IONDV Documentation

Release latest

IONDV LLC and The IONDV Community

Jun 07, 2021

Оглавление

ГЛАВА 1

1. About IONDV. Framework

IONDV. Framework is an node.js opensource framework for developping data accounting applications and functional modules based on JSON/YAML metadata. Visual constructor IONDV. Studio allows you to create applications with "no code" and build applications using REST-API web-services (module rest). The basic module registry is a universal tool for viewing and editing data, processing it according to workflows.

See the video for the application development and building process

1.1 Free demo apps

Check out our demo apps right now:

- Studio is a specialized IDE, created as an iondv application for visual (no code) development of IONDV.Framework applications. Tutorial and video on creating an application using IONDV. Studio. See on GitHub.
- Telecom is an application for accounting, storing and viewing data on the availability of communications (internet, cellular, TV, post offices etc.) in different localities. See on GitHub
- DNT is an application for developping and testing the framework functionality, in which every accounting entity represents the type of data, for example the "string" class or the "collection" class. It allows to explore the framework's features through the app. See on GitHub.
- War Archive is a software solution based on the IONDV. Framework, developped for the project "Remember everyone" which aims to digitize the archive documents, upload them to the database and allow everyone access to them. See on GitHub.
- Project Management is an application for the management of project activities of the regional authorities aiming to control the results of the activities, fitting and diminishing the terms of their accomplishment; effective usage of time, human and financial resources; making up-to-date and justified management solutions. See on GitHub
- CRM is a software solution developed for registring, accounting, storing and viewing buisness data (such as requests, incoming calls, clients, product, services and so on). See on‘GitHub <https://github.com/iondv/crm-ru>‘_

Login: demo, password: ion-demo. No signing up required.

1.2 Standard applications

IONDV. Framework is a wide range web-applications constructor as the subject area is defined by the metadata structure. For example, applications such as following may be created:

- CRM for the clients relationships management;
- accounting and managing the company resources;
- automatization of company workflows and document management;
- collecting and storing any data, for example equipment metrics tracking (IoT);
- representation of data in the form of portals;
- REST-API for SPA applications;
- REST-API and background for mobile apps;

1.3 The structure of the framework

The framework-based application schema: core + metadata + modules = application

The following components are shown at the figure:

- ION Core is the core of the application in the form of the IONDV. Framework;
- meta class, meta view, meta navigation, meta workflow, meta security are the functional metadata of the application's structures, views, navigations, workflows and security respectively;
- registry module shows the plug-in functional modules, for example the Registry module for data viewing and editing;

Below are additional meta types and modules. They bring additional functionality and are applied accordingly with the application specifics. The application dependencies are represented in the package.json file.

The application is a meta-description of its behaviour in the files of the JSON (YAML) formats + functional code + HTML templates, expanding standard functionality -> it's comfortable to work with using git version repository. See the examples at Github .

More details on functional features of the IONDV. Framework and its modules in documentation.

1.4 Functional features

IONDV. Framework provides the following functionality:

- descriptive metadata into the data storage structure in the DBMS;
- functionality to work with various DBMS (ORM technology);
- authorization in a system with different policies, by default oath2, with an open, configurable API for connecting passport library authorization modules which provides up to 500 different authorization policies;

- securing access to data - static securing to data types, to navigation, to stages of business processes, to actions on a form; dynamic securing- through the conditions in the data that the profile of the current user must correspond to (belonging to the unit or organization specified in the object, group or other conditions); through url; providing exceptions in authorization and security by url or for a special user;
- connecting modules that provide additional functionality and are implemented through access to the kernel interfaces (APIs) ;
- providing import, export of data in the system, metadata and security from files;
- providing interaction with the file system for storing data, including external file storages, such as nextcloud;
- calculating values with formulas and caching this data;
- providing eager loading and data filtering in connected collections;
- caching requests and sessions in memcached, redis;
- scheduled jobs;
- notifying users about events.

1.5 Documentation

The IONDV.Framework documentation is available in two languages — english and russian.

1.6 Links

Links to the additional information on web application development using the IONDV. Framework.

- Documentation
- Web site
- Video tutorials on Youtube
- Feedback on Facebook

ГЛАВА 2

2. Getting started

2.1 How to deploy

2.1.1 Step 1 Environment Setup

Environment is a list of programs required to start.

Environment required for launching the IONDV. Framework with applications:

- DBMS MongoDb version 3.6.
- Development environment Node.js version 10.x.x.

DBMS

1. DBMS installation required MongoDB. Verified version 3.6.9 and 4.0.0.
2. Then create folder data on the C: drive and a subfolder ‘‘db‘‘in it.
3. For launching the database move to the folder of the MongoDblocation, then to the server\bin folder and launch the file mongod.exe. If you need to use the DB catalog different from c:\data\db, then the file mongod.exe should be launched with the parameter --dbpath + path to the folder.

Node.js runtime environment

Node.js is an environment in which the components are executed.

1. Node.js runtime environment setup required. Verified version node.JS 10.14.2.
2. By default, the installer prescribes the paths to the Node.js in PATH, and also installs the npm package manager.

Install global dependencies

Install global dependencies using cmd.execommand prompt ran as administrator, after installation of node.js.

NB: command node -v will show the node.js version.

Environment Setup for Windows

1. Install globally the package node-gyp required to build various libraries using command npm install -g node-gyp.
2. To run the library under the Windows operating system, you must additionally install the windows-build-tools package npm install -g --production windows-build-tools.

Project builder package

To organize testing and build distributions during development Gulp is used. Install globally using command npm install -g gulp@4.0. 4.0 is a supported version of Gulp.

Frontend libraries installer

To install frontend libraries, use‘bower <https://bower.io>‘_. Install globally using command npm install -g bower.

2.1.2 Step 2 Install core, modules and application

Clone application and its components

NB: paths should not contain Russian letters and spaces. We recommend that you place your application in c:\workspace.

Let’s examine the creating a project with modules using the develop-and-test application as an example.

1. Find the application on github. Search for develop-and-test and click it.
2. Go to the file repository on the version branch.
3. Open package.json file to check the dependencies.

```
"engines": {
  "ion": "3.0.0"
},
"ionModulesDependencies": {
  "registry": "3.0.0",
  "geomap": "1.5.0",
  "portal": "1.4.0",
  "report": "2.0.0",
  "ionadmin": "2.0.0",
  "dashboard": "1.1.0",
  "soap": "1.1.2"
},
"ionMetaDependencies": {
```

(continues on next page)

(continued from previous page)

```
  "viewlib": "0.9.1"
}
```

1. engines": "ion": 3.0.0 - version of the core 3.0.0.
2. ionModulesDependencies - list of modules and their versions.
3. ionMetaDependencies - list of other metadata required for the project; in this case the ‘‘viewlib‘‘exception is a view library.

NB: to switch to tag version number, see versions in package.json file.

Getting repository of the core

The core is located in the repository framework. The field containing the path to the repository is at the main page.

1. Run the command prompt as Administrator.
2. Copy the repository path, go to the workspace folder using command cd c:\workspace and run git clone https://github.com/iondv/framework. The command will create the framework folder and clone the repository into it.

Getting Modules

1. Go to the modules flofer using command cd framework\modules.
2. For each module from the package.json list in the ionModulesDependencies property find the module repository among the group of modules‘‘https://github.com/iondv/ION-MODULES‘‘.
3. Clone all the modules from the ionModulesDependencies list using command git clone https://github.com/iondv/registry.
4. Go to the folder of the installed module, switch to thetag version number git checkout tags/v1.27.1. For example 1.27.1 is a number of registry module version.
5. Repeat for each module.

Getting Application

1. Go to the application folder. If you are in the modules folder, run cd ..\applications.
2. Next, go back to the develop-and-testrepository page, copy the path and clone it using command git clone https://github.com/iondv/develop-and-test.
3. Go to the folder of the installed application, switch to the tag version number git checkout tags/v1.17.0.
4. The dependencies installation in ionMetaDependencies is carried out in the applicationsfolder. For installation make sure you’re in the applications folder. Clone the applications from the list in the ionMetaDependencies parameter. For‘‘viewlib‘‘ application clone using command‘‘git clone https://github.com/iondv/viewlib‘‘.
5. Go to the folder of the installed application, switch to the tag version number git checkout tags/v0.9.1. Repeat for each application.
6. The application is compiled.

NB: we recommend creating a project for it in an IDE, for example Visual Studio Code, and creating the configuration file in it.

Config file

Config file is for setting the main parameters of the app environment and configuring the additional start parameters.

1. Create setup config file with ini extension in config folder.
2. Open the file in the editor and paste the contents.

```
auth.denyTop=false
auth.registration=false
auth.exclude[]=/files/**
auth.exclude[]=/images/**
db.uri=mongodb://127.0.0.1:27017/iondv-dnt-db
server.ports[]=8888
module.default=registry
fs.storageRoot=./files
fs.urlBase=/files
```

The main parameter is db.uri=mongodb://127.0.0.1:27017/db. It indicates the name of the database we'll use for the application. The database will be created authomatically.

2.1.3 Step 3 Build and run application

For all further commands, run command prompt as Administrator.

Go to the application folder cd c:\workspace\framework``and set the environment variable ``NODE_PATH equal to the path to the application. For Windows: command set NODE_PATH=c:\workspace\framework, for Linux: export NODE_PATH=/workspace/framework.

Build application

Building the application provides installing all the libraries, importing the data into the database, and preparing the application to run.

1. At the first launch, you need to run npm install command to install key dependencies, including the gulp task runner locally. Make sure that the Gulp version is 4.0. This command will install all the libraries from the ‘‘dependencies‘‘property of ‘‘package.json‘‘file of the core.
2. After that, and in all the next times, run the command gulp assemble to build the application.

   NB: Make sure that the environment variable NODE_PATHis set, the MongoDBdatabase is running, the Gulp is globally and locally installed and its version is‘‘4.0‘‘.
3. Before launching the application, add a basic user for logging in. Open‘‘Mongo Compass‘‘ app and find the ion-user``table in the database. Delete all the records you see there. Then go back to the console and execute the commands below. Add ``admin user with 123 password using command node bin/adduser.js --name admin --pwd 123. Add administrator rights to the user using command node bin/acl.js --u admin@local --role admin --p full.

Launch application using scripts

The bin folder contains the scripts to run applications based on the IONDV.Framework, such as:

- acl.js
- export.js
- import.js and import-data.js
- setup.js
- www.js

NB. Запускаются локально из папки framework, шаблоны команд указаны в разделах с описанием назначения скрипта.

More details about application launch scripts

Run application

After finishing the build you can launch the app. Make sure that the environment variable NODE_PATH is set. If skipping this step, the system will show the missing components error.

The system is launched by the command npm start, alternative launch is node bin/www.

After launching the system, open the browser with http://localhost:8888``adress and log in to the application, where ``8888 is a port specified in the server.ports parameter in the launch configuration.

2.1.4 Application launch scripts

The bin folder contains the scripts to run applications based on the IONDV.Framework, such as:

- acl.js
- export.js
- import.js and import-data.js
- setup.js

NB. Запускаются локально из папки framework, шаблоны команд указаны в разделах с описанием назначения скрипта.

acl.js

Launch command template: node %NODE_PATH%\bin\acl.js --d %NODE_PATH%\applications\%IONAPP%\acl where NODE_PATH is a path to the platform directory, %IONAPP% is a name of the application.

Adds rights to the system objects, specified in the acl folder.

Also the commands for creating roles and its rights in the system are available in case of missing settings in the aclfolder:

- Setting the login and the password node bin/adduser.js --name admin --pwd 123
- Setting the access node bin/acl.js --u admin@local --role admin --p full

- Rights to generate a token for the rest/token service node bin/acl.js --role admin --p USE --res ws:::gen-ws-token

По такому же принципу можно задавать пользователей и права на отдельные ресурсы системы

export.js

Launch command template: node bin/export --ns %IONAPP%, where“NODE_PATH“ is a path to the platform directory, %IONAPP% is a name of the application (namespace).

Exports data and meta out of the current built and locally running application. Export files are formed in the folder structure, located at the same point as the directory platform in the out folder.

import.js and import-data.js

When importing meta, the data import is not implemented by default. For importing meta along with the data call the command:

node bin/import.js --src %NODE_PATH%/applications/%IONAPP% --with-data --ns %IONAPP% , where“NODE_PATH“ is a path to the directory platform, %IONAPP% is a name of the application (namespace).

For importing the data call:

node %NODE_PATH%/bin/import-data.js --src %NODE_PATH%/applications/%IONAPP%/data --ns %IONAPP%

Besides, when importing meta with the data specify the directory of the application(and the data for import will be located in the data subdirectory), but when importing the data specify the data directory.

setup.js

Launch command template: node %NODE_PATH%\bin\setup %IONAPP% , where NODE_PATH is a path to the directory of the platform, %IONAPP% is a name of the application.

Installs the application, runs the application deployment script, what includes importing and recording to the database the application modules meta.

This section explains how to deploy the system. Several steps are required to deploy the system.

2.1.5 Steps

1. Environment Setup
2. Install core, modules and application
3. Build and Run application

2.2 Quick start

2.2.1 IONDV. Studio

IONDV. Studio is the IONDV. Framework application. It can be used as standalone node.js application or as the desktop application. The IONDV. Studio can be deployed with the use of the studio demo version. Registration and account are not required.

Description

IONDV. Studio is an application for visual development and editing metadata (such as classes, navigation, views, workflows, portal forms), which are deployed as an IONDV.Framework web application.

How to launch application in the Studio

Watch video on how to deploy the application from the repositry in the IONDV. Studio.

- Choose a ready-made application on github. For example, Nutrition-Tickets.
- Download the zip file and open it in the Studio application.
- Click the play button in the upper right corner of the screen to launch the application.
- The application will start deploying. It takes around 80 seconds.
- Get the link to the application and click it.
- Login to the application using demo as the login and ion-demo as the password.

The application is running! The application in the demo Studio is stored for 1 hour, you get to know the technologie and try to make your own changes to the structure of the application and see the result. After 1 hour the application will be removed from the server.

How to create an application in the Studio

В Студии также можно создавать приложения. Смотрите видео о создании простого приложения IONDV. Nutrition-tickets в IONDV. Studio. Инструкция доступна в репозитории IONDV. Nutrition-Tickets.

- Press + to start creating an application. Fill out the required fields in the popped-up window. The tab for the application you created will appear in the left upper corner of the screen and can be managed as a browser tab.
- The side menu will appear, which is the application work panel. The Classes section is for creating Classes and Attributes.
- Start creating an application with a Class. Click on the Class and in the work space click +Class.
- Now we've got the Class and the Attribute ID is automatically created.
- When the Class is selected, you can add its Attributes by clicking on +Attribute. Attribute properties and types description here.
- When you've created at least the 2 Classes you can set up the connections between them. It can be made through the data type setting when creating Class attribute. Main types are the Collection and Reference. The specified type of data of the Attribute will be shown in the workspace as a linking line.

The created Classes and their Attributes will appear in the Class section. It is called the project tree, which will make it easy to navigate the project when the amount of Classes will get bigger. These are the basic things about creating an application.

Application in the IONDV.Studio

To develop an application in the Studio, first launch it using one of the following methods:

- Use the demo of the Studio.
- Launch locally as the IONDV. Framework application with the source code from github. After that build and deploy the application according to the instruction for the Framework applications.
- Open in the browser by the link http://localhost:8888.
- Launch locally as a standalone node.js application according to the instruction below.
- Launch as a desktop application according to the instruction below.
- Launch in docker-container using command docker run -d -p 8888:8888 –name studio iondv/studio.

Open in the browser by the link http://localhost:8888.

Then:

- Develop your application in the Studio by creating classes and navigation.
- Note that the data is stored in the local reporitory of the browser. Export the application as zip.
- Download the latest version of the IONDV. Framework and the IONDV. Registry module from GitHub repositories Framework and Registry.
- Follow the standard instruction on deploying the application by git, excluding the application. Instead of the application deploy your zip file in the applications folder.
- Then build and deploy the application according to the instruction IONDV. Framework.

Ways to use the Studio

Standalone node.js application

The advantage of using the standalone application is no need of the database and the IONDV. Framework.

- Run the command git clone https://github.com/iondv/studio.git. Change the local directory to studio.
- Execute the command npm install to install all necessary dependencies, including the local application for building gulp.
- Please, make sure that you have the Gulp of version‘4.0‘ globally installed.
- Execute the command gulp build to build the application.
- Launch the application using the command npm start or node www‘(use ‘node standalone to launch the application as [standalone](/readme-standalone_ru.md).)
- In the browser go to the http://localhost:8888.

The Desktop application of the Studio (node-webkit)

Before forming the desktop application of the Studio, build Standalone node.js application

Launch new studio at the node-webkit local server

1. Download the latest NORMAL version the node-webkit from the website https://nwjs.io/.
2. Unpack the archive contents into a convenient folder.
3. Use one of the available methods to connect the application and the node-webkit.

The examples are described in the article https://github.com/nwjs/nw.js/wiki/How-to-package-and-distribute-your-apps in the 2a and 2b sections.

A more convenient option is to use the package nw-builder:https://github.com/nwjs-community/nw-builder. Example of command: nwbuild ./studio -p win64 -v 0.34.0 -o ./destination. It's worth noting that nw-builder will itself download the required node-webkit version.

As a result, you'll get your application in the folder with dll files that nwjs uses. Launch the application using the nw.exe file (the name may differ).

Formation of a single executable file

1. Download Enigma virtual box from the website https://enigmaprotector.com/en/downloads.html, install and run
2. Enter the path to the executable file of your application in the first field. (Choose)
3. Enter the path to save the executable file in the second field.
4. Enter ALL files and folders from your application directory except the executable file .exe in the field Files.
5. In the Files options menu, check Compress.
6. Click Process and wait for the results.

External App Tracker

All settings in deploy.json -> globals -> externalAppTracker

```
{
  "items": [{
    "name": "dnt",
    "title": "Develop and test",
    "url": "https://github.com/iondv/develop-and-test/archive/master.zip"
  }, {
    "name": "crm-en",
    "title": "CRM EN",
    "url": "https://github.com/iondv/crm-en/archive/master.zip",
    "language": "en"
  }, {
    "name": "crm-ru",
    "title": "CRM RU",
    "url": "https://github.com/iondv/crm-ru/archive/master.zip",
    "language": "ru"
  }],
  "front": "/themes/portal/static/archives/",
  "storage": "applications/studio/themes/portal/static/archives/",
  "tempZip": "applications/studio/temp.zip",
  "enableUpdate": false,
```

(continues on next page)

(continued from previous page)

```
  "updateInterval": 86400
}
```

- item.name - sets the file name when saving the archive
- item.title - is displayed on the client when the application is selected
- item.url - remote address of the application archive
- item.front - address of the archive for the client, if not specified, it is created according to the general setting and the name
- item.language - if not specified the application will be displayed in any language
- storage - the place to store the applications archives
- front - link to the archives from the client
- tempzip - temporary file for remote download from another server
- enableUpdate - turn on/off the synchronization with a remote server. When starting the server, the presence of the archives is checked, and if not found, then they are downloaded from the specified URL. After the updateInterval is expired the archives are updated
- updateInterval - the period of re-uploading the archive to the server (seconds)

You may specify the custom URL on the client, but take into consideration that the browser only allows the explicitly permitted Access-Control-Allow-Origin downloads from other hosts

Read the original instruction on the website https://github.com/nwjs/nw.js/wiki/How-to-package-and-distribute-your-apps in the section An alternative way to make an executable file in Windows

Links

- The application repository
- Node-webkit
- Node-webkit wiki
- Package for generating the executable file
- Program for linking dll files
- User guide
- Launch application as standalone
- Instruction for creation of information systems with the IONDV. Studio

2.2.2 Build application from the repository

For quick start of the applications with the use of the Github repository implement the following steps:

1. Set up the system environment and global dependencies.
2. Clone the core, module and application.
3. Build and deploy the application.
4. Run it.

Below is the detailed guide on how to build the application from the Github repository.

Detailed guide

System environment and global dependencies

The framework is launched in the Node.js runtime. Version 10.x.x.

To store the data, you need to install and run MongoDb of version later than 3.6.

To build all the components and libraries of the framework, you need to install the following components globally:

- package node-gyp npm install -g node-gyp. For the Windows operating system, it is necessary to additionally install the windows-build-tools package npm install -g --production windows-build-tools.
- Gulp <http://gulpjs.com/>‘_installation package ‘‘npm install -g gulp@4.0‘. 4.0 is supported version of Gulp.
- for versions 3.x.x and earlier of the IONDV. Framework the manager of the frontend libraries packages Bower is required‘‘npm install -g bower‘‘. It’s not required for versions 4.x.x and later .

Manual core, modules and application installation

Let’s take the develop-and-test application as an example. Find the application‘‘develop-and-test‘‘ in the repository. Check the dependencies specified in the file package.json.

```
 "engines": {
  "ion": "1.24.1"
},
"ionModulesDependencies": {
  "registry": "1.27.1",
  "geomap": "1.5.0",
  "graph": "1.3.2",
  "portal": "1.3.0",
  "report": "1.9.2",
  "ionadmin": "1.4.0",
  "dashboard": "1.1.0",
  "lk": "1.0.1",
  "soap": "1.1.2",
  "gantt-chart": "0.8.0"
},
"ionMetaDependencies": {
  "viewlib": "0.9.1"
  "viewlib-extra": "0.1.0"
```

- First, install the core, which version is specified in the parameter "engines": {"ion": "1.24.1"}. Copy the core repository address and run the command git clone https://github.com/iondv/framework in the command prompt. Go to the core folder, switch to the tag of version number git checkout tags/v1. 24.1. Since the compatibility is provided at the metadata level, and new versions were released due to the changes in the building technologie, you may use the latest version, for example 4.0.0.
- After that the required modules, specified in the "ionModulesDependencies" parameter, are to be installed. The modules are installed to the modules folder of the core. To do that, go to the folder using the cd modules command. Clone the modules from the list "ionModulesDependencies". For the registry module use the command git clone https://github.com/iondv/registry. Go to the folder of the installed module, switch to the tag of the version number git checkout tags/v1.27.1. Repeate for each

module. For the majority of applications, you can use the latest versions of modules compatible with the core.

- The installation of the application is carried out in the applications folder. If you are in the modules folder, move to the folder using the command cd ..\applications. Install by cloning the repository using the command git clone https://github.com/iondv/dnt_ru.
- Finally, install all necessary applications listed in the "ionMetaDependencies" parameter in the applications folder. Make sure that you're inside this folder. Clone the dependencies in ionMetaDependencies. For the‘‘viewlib‘‘application execute the command‘‘git clone https://github.com/iondv/viewlib‘‘. Go to the folder of installed application and switch to the tag of the version number git checkout tags/0.9.1. Repeat for each application.

Build, configure and deploy the application

Build of the application provides installation of all dependent libraries, importing data into the database and preparing the application for launch.

Create the configuration file setup.ini in the config folder, where the framework was cloned to set the main parameters of the application environment.

```
auth.denyTop=false
auth.registration=false
db.uri=mongodb://127.0.0.1:27017/db
server.ports[]=8888
module.default=registry
fs.storageRoot=./files
fs.urlBase=/files
```

Open the file in the editor and paste the contents. The main parameter db.uri=mongodb://127.0.0.1:27017/ion-dnt specifies the name of the database which will be used for the application. The database will be created automatically.

Set the NODE_PATH environment variable to the path to the core of the application using command set NODE_PATH=c:\workspace\dnt``for Windows and ``export NODE_PATH=/workspace/dnt for Linux, where workspace\dnt is the folder with the cloned framework.

At the first launch, you need to run npm install. It will install the key dependencies, including the gulp task runner locally.

Next, execute the command to build the application gulp assemble.

If you want to implement the data import in your project, check the folder data in the application and execute the command: node bin/import-data --src ./applications/develop-and-test/data --ns develop-and-test

Add the admin user with the 123 password by executing the command node bin/adduser.js --name admin --pwd 123.

Add admin rights to the user executing the node bin/acl.js --u admin@local --role admin --p full command.

Start

Launch the application from the core folder using command npm start or node bin\www .

Open the link http://localhost:8888 in a browser and log in. 8888 is a port in the server.ports parameter of the launch configuration.

2.2.3 Docker-container

Applications can also be launched using a docker container.

Docker is a software platform for rapid development, testing and application deployment.

Steps

1. Run mongodb DBMS: docker run --name mongodb -v mongodb_data:/data/db -p 27017:27017 -d mongo
2. Launch IONDV. App docker run -d -p 80:8888 --link mongodb iondv/app, where App is a name of the application and the path to the corresponding repository.
3. Open the link ‘‘http://localhost‘‘in a browser in a minute (the time required for data initialization). For log in use the standard login: demo, password: ion-demo

2.2.4 Installer for Linux

You can also use the IONDV. Framework application installer for Linux, which requires node, mongodb and git installed. During the installation all other dependencies will be checked or installed, and the application itself will be built and launched.

Install in one command:

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) -t git -q -i -m␣
↪localhost:27017 app
```

Where the parameters for iondv-app localhost:27017` `is MongoDB address and ` `app is the name of the application. After launch open the link http://localhost:8888, back office account demo, password ion-demo.

The other way is to clone git clone https://github.com/iondv/iondv-app.git and install the application using the command bash iondv-app -m localhost:27017 app, where app is the name of the application.

You can also build the application in the docker-containers. In that case you only need the docker and the mongodb DBMS in the docker-container.

Environment requirements

The MongoDB DBMS should be running for the system to work. You can launch it in the docker-container using the command:

docker run --name mongodb -v mongodb_data:/data/db -p 27017:27017 -d mongo

System preparation is described below.

Install, build and run the application using a single command in the docker-container

The examples below assume that the DBMS is running in the container named mongodb.

Taking the simple application Nutrition-Tickets as an example. The command downloads the installer from the github repositor, which builds and runs the appliction in the docker-container in the current repository.

- from the git repository

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \   -t docker -q -i -
↪l mongodb nutrition-tickets
```

- from the zip archive file, for example, downloaded from github curl -L https://github.com/iondv/nutrition-tickets/archive/master.zip > ./nutrition-tickets.zip or created in the IONDV. Studio. Note, that you need to specify the module for data display in the‘‘package.json‘‘file of the created application in the attribute "ionModulesDependencies", usually it’s "registry".

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \   -t docker -q -i -
↪l mongodb ./nutrition-tickets.zip
```

- from the folder, while the original application folder is not modified. Note, that the name of the folder must match the namespace of the application (if the folder is unzipped from the github archive, the branch code is usually added in the name, so you need to rename it)

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \   -t docker -q -i -
↪l mongodb ./nutrition-tickets
```

The default address of the built application is http://localhost:8888, login demo, password ion-demo.

Install, build and run the application on your local file system

The examples below assume that the DBMS is running locally and is available at localhost:27017.

Installation on the local file system allows to get the application, which is ready to be modified and refined by a developper tools, such as IDE, just open the application folder. The folder of the application is created in the launch folder or in the folder specified by the ‘‘-p‘‘parameter

- from the git repository in the folder /workspace

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \  -t git -p /
↪workspace -m localhost:27017 https://github.com/iondv/nutrition-tickets.git
```

- from the zip archive in the current folder, for example, downloaded from github curl -L https://github.com/iondv/nutrition-tickets/archive/master.zip > ./nutrition-tickets.zip or created in the IONDV. Studio. Note, that you need to specify the module for data display in the attribute "ionModulesDependencies" of the package.json file of the created app , it’s usually "registry" module

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \ -t git -
↪p /workspace -m localhost:27017 ./nutrition-tickets.zip
```

- from the folder, while the original application folder is not modified. Note, that the name of the folder must match the namespace of the application (if the folder is unzipped from the github archive, the branch code is usually added in the name, so you need to rename it)

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) \   -t git -q -i -m␣
↪localhost:27017 ./nutrition-tickets
```

Launch parameters

iondv-app [OPTION]... IONDV_APP_NAME|IONDV_APP_NAME@VERSION|GIT_URL|IONDV_APP_ZIP|IONDV_

Parameters	Type of the application build
-t [value]	git: cloning the repositories to the file system (git installed required) docker: building in docker-containers, no environment is required on the computer
-c [value]	launch application as a cluster with the amount of installations [value]
-m [value]	uri for mongodb, examples: mongodb:27017. localhost:27017 by default (when building in docker you'll get the error of connecting to the DB(!). For docker use the parameter -l, or specify the external DBMS address
-r	check and remove the folder with the application name in the build directory
-i	data import when initializing the application
-a	roles and user accounts import when initializing the application
-y	setting all values to default (yes to all)
-q	silent mode. Only basic information, warnings, and errors are displayed
-l [value]	the name of the MongoDB container for linking to the built container (docker build type or the parameter -d when building with git), also forms the configuration with the value of mongo uri as [value]:27017
-p [value]	path to the directory where the application name folder will be created and application build will be held
-s [value]	the full path to the script that runs in the application folder after the build, but before the application is deployed. Can be used for additional processing of application files
-n [value]	the parameter defining the launch, changing the application namespace to a new one, before the deployment
-h	skipping switching to application dependency versions, installing the latest versions
-x	exit without launching the application
Parameters fo the git method:	
-d	prepare a docker container based on the compiled version. Also stop and delete a container, an image with that name
-k	skip the environment check
Parameters for docker build method:	save temporary versions of containers, it allows you to speed up subsequent builds. But caching is skipped if the ignore dependency versions flag is set
-v	prepare a docker container based on the compiled version. Also stop and delete a container, an image with that name
Environment variables:	
IONDVUrlGitFramework	URL of the framework repository, by default https://github.com/iondv/framework.git You can set the login and the password to your version in the private repository. For example: https://login:password@git.company-name.com/iondv/framework.git
IONDVUrlGitModules	URL to modules, by default https://github.com/iondv
IONDVUrlGitApp	URL to applications, is used if only the name of the application is specified for the building, by default https://github.com/iondv
IONDVUrlGitExtApp	URL to extension applications, by default https://github.com/iondv

Preparing the environment

Installation of docker

It is recommended not to do it under root

- Installing the latest version of docker for CentOS:

1. Обновляем систему sudo yum update

2. Устанавливаем необходимые библиотеки yum install -y yum-utils device-mapper-persistent-data lvm2

3. Регистрируем репозиторий yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo

4. Установка последней версии yum -y install docker-ce docker-ce-cli containerd.io

5. Запускаем докер systemctl start docker

6. Для автоматического запуска докера systemctl enable docker

- Install the latest docker version for Ubuntu:

1. Add GDP key curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
2. Check the key apt-key fingerprint 0EBFCD88
3. Add the repository

```
sudo add-apt-repository \    "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
```

4. Update the repositories sudo apt-get update
5. Install the latest version sudo apt-get install docker-ce docker-ce-cli containerd.io

Add the current user to the docker group:

sudo groupadd docker sudo usermod -aG docker $USER

You can check docker run hello-world

Lunch Mongo in docker

Launch with mapping to the local port:

docker run --name mongodb -v mongodb_data:/data/db -p 27017:27017 -d mongo

Install node

To speed up the build, it's recommended to download docker-image node:10 in advance, since it takes 900Mb.

docker pull node:10

Check it using the command docker images | grep node, the list of node local images will be displayed.

We recommend that you start getting to know the IONDV. Framework with the quick start of applications.

This section contains the instructions on how to deploy the IONDV. Framework applications. There are several ways to quickly launch the applications. You can choose the most convenient for you.

Deploy the application in following ways:

- use IONDV. Studio
- clone the repository of the application and install all the components see
- use :doc:‘docker-container <quickstart/3_docker_container>‘with built applications
- use installer for Linux operating system

2.2.5 Links

Links for additional information on developing applications using the IONDV.Framework.

- Documentation
- Framework Homepage
- Feedback on Facebook
- Video tutorials on Youtube

ГЛАВА 3

3. Development

3.1 Expand functionality

3.1.1 Development of functional utilities in the application

The application's functional utilities are designed to solve specific application tasks, without implementing separate logic in the form of a module.

For example, the utilities of a scheduled job or the utilities called upon transition in a workflow.

Utilities for action buttons

Utilities for action buttons are designed to automate the execution of certain actions when you click a button in a web form. The button is connected for the web form of creating and editing the object through the commands array of the main form object:

```
{
  "commands": [
    {
      "id": "SAVE",
      "caption": "Save",
      "visibilityCondition": null,
      "enableCondition": null,
      "needSelectedItem": false,
      "signBefore": false,
      "signAfter": false,
      "isBulk": false
    },
    {
      "id": "SAVEANDCLOSE",
      "caption": "Save and close",
      "visibilityCondition": null,
```

(continues on next page)

(continued from previous page)

```
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
  },
  {
    "id": "CREATE_INDICATOR_VALUE",
    "caption": "Form the collected values",
    "visibilityCondition": null,
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
  }
],
```

The three buttons are available on the form: SAVE, SAVEANDCLOSE and CREATE_INDICATOR_VALUE.

Before connecting to the form, the custom buttons must be set in deploy.json in the object modules.registry.globals.di.actions.options.actions:

```
{
  "modules": {
    "registry": {
      "globals" : {
        "di": {
          "actions": {
            "options": {
              "actions": [
                {
                  "code": "CREATE_INDICATOR_VALUE",
                  "handler": "ion://createIndicatorValueHandler"
                },
                {
                  "code": "ASSIGNMENT_TO_EVENT_ONLY",
                  "handler": "ion://assignmentToEventOnly"
                },
                {
                  "code": "CREATE_PROJECT_REPORTS",
                  "handler": "ion://createProjectReportsHandler"
                }
              ]
```

Also the parameters of the handler module used by the button must be specified:

```
{
  "modules": {
    "registry": {
      "globals" : {
        "di": {
          "createIndicatorValueHandler": {
            "module": "applications/sakh-pm/lib/actions/createIndicatorValueHandler",
            "initMethod": "init",
            "initLevel": 2,
```

(continues on next page)

(continued from previous page)

```
        "options": {
          "data": "ion://securedDataRepo",
          "workflows": "ion://workflows",
          "log": "ion://sysLog",
          "changelogFactory": "ion://changelogFactory",
          "state": "onapp"
        }
      },
```

In this example, clicking the button"CREATE_INDICATOR_VALUE" launches the script ./applications/sakh-pm/lib/actions/createIndicatorValueHandler.js.

The contents of the script:

```
/**
 * Created by kras on 08.09.16.
 */
'use strict';

const ActionHandler = require('modules/registry/backend/ActionHandler');
const edit = require('modules/registry/backend/items').saveItem;
const ivc = require('../indicator-value-creator');

/**
 * @constructor
 * @param {{}} options
 * @param {DataRepository} options.data
 * @param {WorkflowProvider} options.workflows
 * @param {Logger} options.log
 * @param {ChangelogFactory} [options.changelogFactory]
 * @param {String} [options.state]
 */
function CreateIndicatorValueHandler(options) {

  options = options || {};

  const work = ivc(options);

  this.init = function () {
    if (options.workflows && options.state) {
      options.workflows.on(
        'indicatorBasic@sakh-pm.' + options.state,
        (e) => {
          let logger = null;
          if (options.changelogFactory && e.user) {
            logger = options.changelogFactory.logger(() => e.user.id());
          }
          return work(e.item, e.user, logger).then(() => null);
        }
      );
    }
  };

  /**
   * @param {{metaRepo: MetaRepository, securedDataRepo: SecuredDataRepository}} scope
   * @param {ChangelogFactory} scope.changelogFactory
   * @param {Request} req
```

(continues on next page)

(continued from previous page)

```
   * @returns {Promise}
   */
  this._exec = function (scope, req) {
    let logger;
    let user = scope.auth.getUser(req);
    if (options.changelogFactory) {
      logger = options.changelogFactory.logger(() => user.id());
    }
    return edit(scope, req, null, logger, true)
      .then(item => scope.dataRepo.getItem(item, null))
      .then((item) => {
        if (item.get('status') !== 'edit') {
          throw new Error('Создать значения показателей, можно только при редактировании!');
        }
        return work(item, user, logger);
      })
      .then((count) => {
        return {$message: 'Создано ' + count + ' значений для ввода по периодам!'};
      });
  };
}

CreateIndicatorValueHandler.prototype = new ActionHandler();

module.exports = CreateIndicatorValueHandler;
```

Utilities for scheduled jobs

Utilities for schedueled jobs are designed to automate the regular execution of some actions at certain intervals.

Each utility must be specified in deploy.json of the application in the globals.jobs object, for example:

```
{
  "globals": {
    "jobs": {
      "ticketClose": {
        "description": "Ночной перевод билетов в статус \"проверен\"",
          "launch": {
            "timeout": 3600000,
            "hour": 24
          },
          "worker": "ticketCloser",
          "di": {
            "ticketCloser": {
              "executable": "applications/khv-ticket-discount/lib/overnightTicketClose",
              "options": {
                "dataRepo": "ion://dataRepo",
                "log": "ion://sysLog",
                "workflows": "ion://workflows"
...
```

di should contain a field with a name equal to worker value - this is the job that will be launched.

Here the script applications/khv-ticket-discount/lib/overnightTicketClose.js is launched on the schedule. launch can be an object containing the following fields:

month, week, day, dayOfYear, weekday, hour, min, minute, sec, second - set the interval for job executions;

check - interval for checking the execution condition, in milliseconds, by default - 1000.

For example, if check is equal to 5000, and sec is 2, the job will be implemented only every 10 seconds, when the interval between checks coincides with the interval of execution

If the job execution interval is not specified, then it will be executed when the application is launched and at each check interval.

timeout - the time in milliseconds after which the running job is interrupted by timeout;

launch can also be an integer - the interval of the job in milliseconds, while the job will also be performed immediately when the application starts. The timeout will be set equal to the execution interval.

In the options field, any variables and their values, which will become available in the script through the fields of the object passed as an argument to the main function of the module, can be specified.

The script is compiled in the module format, for example:

```
"use strict";
const Logger = require("core/interfaces/Logger");

module.exports = function (options) {
  return options.dataRepo
    .getList(
      "ticket@khv-ticket-discount",
      "ticketYear@khv-ticket-discount"
    )
    .then((tickets) => {
      let p = Promise.resolve();
      tickets.forEach((ticket) => {
        p = p
          .then(() => options.dataRepo.editItem(ticket.getClassName(), ticket.getItemId(), {"state": "close"}))
          .then(item => (item.name === "ticketYear" ?
            options.workflows.pushToState(item, "ticketYear@khv-ticket-discount", "close") :
            options.workflows.pushToState(item, "ticket@khv-ticket-discount", "close")))
          .catch((err) => {
            if(options.log instanceof Logger) {
              options.log.error(err);
            } else {
              console.error(err);
            }
          });
      });
      return p;
    });
};
```

Utilities for printed forms

Utilities for printed forms (injectors) are designed to process data output to the template, including performing intermediate calculations and formatting.

The printed form for which the injector will be used must be defined in deploy.json, for example:

```
"registry": {
  "globals": {
    "di": {
```

(continues on next page)

(continued from previous page)

```
            "pmListToDocx": {
                "module": "modules/registry/export/listToDocx",
                "initMethod": "init",
                "initLevel": 0,
                "options": {
                    "tplDir": "applications/khv-ticket-discount/export/list",
                    "log": "ion://sysLog",
                }
                ...
            }
            ...
        }
        ...
    }
    ...
}
```

In this case, the listToDocx module is used, so a list of all objects of a certain class will be uploaded to the printed form.

For each such class in tplDir, create a folder with the name of the namespace, into which then place a file with the name of the class you need to unload from this namespace, for example:

```
...\applications\khv-ticket-discount\export\list\khv-ticket-discount\ticketYear.docx
```

Thus, a list of all objects of the ticketYear@khv-ticket-discount class will be uploaded to the document.

The utility itself is a .js script connected to the application in the module format in deploy.json, for example:

```
"registry": {
    "globals": {
        "di": {
            "weekTicketStatsInjector": {
                "module": "applications/khv-ticket-discount/export/injectors/monthTicketStats",
                "options": {
                "dataRepo": "ion://dataRepo"
                }
                ...
            }
            ...
        }
        ...
    }
    ...
}
```

The .js file here is located at the “module” path.

After connecting, the utility must be included in the options of the printed form:

```
"registry": {
    "globals": {
        "di": {
            "pmListToDocx": {
                "module": "modules/registry/export/listToDocx",
                "initMethod": "init",
                "initLevel": 0,
```

(continues on next page)

(continued from previous page)

```
            "options": {
                "tplDir": "applications/khv-ticket-discount/export/list",
                "log": "ion://sysLog",
                "injectors": [
                    "ion://monthTicketStatsInjector"
                ]
            }
            ...
        }
        ...
    }
    ...
}
```

The injector script is compiled in the module format, provided that it must contain the this.inject function, into the parameter of which the object with the list of objects of the class specified earlier will be transferred to, for an example from this reference:

```
ticketYear@khv-ticket-discount
```

Example of the monthTicketStats.js file:

```
function monthTicketStatsInjector() {
  this.inject = function (value) {
    if (value && value.className === "ticketYear") {
      let expValueList = [];
      const periodBegF = value.periodBegF;
      const periodEndF = value.periodEndF;
      const areaF = value.areaF;
      let i = 0;
      value.list.forEach((vectorparams) => {
        if (vectorparams.person.area.code === areaF && vectorparams.dateAirGo >= periodBegF &&␣
↪vectorparams.dateAirGo <= periodEndF && ((vectorparams.state !== "canceled") && (vectorparams.state !
↪== "returned"))) {
          expValueList[i++] = vectorparams;
        }
      });
      value.list = expValueList;
    }
    return value;
  };
}

module.exports = monthTicketStatsInjector;
```

Example of an export configuration for this form in deploy.js:

```
"registry": {
    "globals": {
        "di": {
            "export": {
                "options": {
                    "configs": {
                        "ticketYear@khv-ticket-discount": {
```

(continues on next page)

(continued from previous page)

```
                    "pmListToDocx": {
                        "type": "list",
                        "caption": "Ежемесячный отчет",
                        "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.
↪document",
                        "extension": "docx",
                        "params": {
                            "periodBegF": {
                            "caption": "Период с",
                            "type": "date"
                        },
                        "periodEndF": {
                            "caption": "по",
                            "type": "date"
                        },
                        "areaF": {
                            "caption": "Район",
                            "type": "reference",
                            "className": "area@khv-ticket-discount"
                        }
                    },
                    "preprocessor": "ion://pmListToDocx",
                    "eagerLoading": [
                        "person",
                        "person.documents",
                        "person.area",
                        "route.pointDeparture",
                        "route.pointArrival",
                        "route.flight"
                    ],
                    "fileNameTemplate": "Ежемесячный отчет"
                }
            }
        }
    }
    ...
  }
  ...
}
```

Here you should pay attention to the params field, you can specify the parameters available in the export form in the application web service in it . The following types of parameters are possible:

“string” - a string for entering text,

“date” - interactive calendar where you can select a required date

“reference” - a reference to the class, in this case, the export window will display a drop-down list of all objects of the class.

The passed parameters will be available in the script through the this.inject function parameter.

Utilities for web service (rest)

Web service utilities are designed to implement the processing of various types of requests to the server.

The service connects to the application in deploy.json in the modules.rest.globals.di object, for example like this:

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "acceptor": {
            "module": "modules/rest/lib/impl/acceptor",
            "options": {
              "dataRepo": "ion://dataRepo",
              "metaRepo": "ion://metaRepo"
...
```

In this case, the service acceptoris connected , it will be available for requests by url https://dnt.iondv.com/rest/acceptor.

Functional description of interaction with requests should be contained in the script modules/rest/lib/impl/acceptor.js.

In the options field, any variables and their values, which will become available in the script through the fields of the object passed as an argument to the main function of the module, can be specified.

The script is compiled in the module format, for example:

```
const Service = require('modules/rest/lib/interfaces/Service');

/** Simple app service - REST module
 * @param {{dataRepo: DataRepository, metaRepo: MetaRepository}} options
 * @constructor
 */
function EchoRest(options) {
  this._route = function(router) {
    this.addHandler(router, '/', 'POST', (req) => {
      return Promise.resolve({
        echo: 'peekaboo'
      });
    });
    this.addHandler(router, '/', 'GET', (req) => {
      return Promise.resolve({
        echo: 'peekaboo'
      });
    });
  };
}
EchoRest.prototype = new Service();
module.exports = EchoRest;
```

A detailed description of the principles of creating the service can be found in https://github.com/iondv/rest/blob/main/README.md/ section Development of the service handler in the app

Utilities for workflow

Utilities for workflow are designed to automate the execution of some actions when the status of a workflow changes. The utility is connected to the application in deploy.json in the globals.plugins object, for example like this:

```
{
  "globals": {
    "plugins": {
      "wfEvents": {
        "module": "applications/sakh-pm/lib/wfEvents",
        "initMethod": "init",
        "initLevel": 1,
        "options": {
          "workflows": "ion://workflows",
          "metaRepo": "ion://metaRepo",
          "dataRepo": "ion://dataRepo",
          "log": "ion://sysLog",
          "AIPConfigPath": "applications/sakh-pm/paths_config/eventOnlyAIP.json"
```

Here the wfEvents utility is connected. The script containing the description of actions when changing the status of the workflow is located on the path applications/sakh-pm/lib/wfEvents.js.

In the options field, any variables and their values, which will become available in the script through the fields of the object passed as an argument to the main function of the module, can be specified.

The script is compiled in module format, provided that it must include the init method, for example:

```
'use strict';
const ivc = require('./indicator-value-creator');

function WorkflowEvents(options) {
  this.init = function () {
    options.workflows.on(
      ['assignmentBasic@sakh-pm.fin'],
      (e) => {
        if (e.transition === 'toKT') {
          return options.dataRepo.getItem(e.item, null, {forceEnrichment: [['meeting','basicObj'],['basicObj']]})
            .then((item) => {
              const data = {
                basicObj: item.property('meeting.basicObj').evaluate() || item.property('basicObj').evaluate(),
                name: item.get('name'),
                owner: item.get('owner'),
                datePlannedEnd: e.item.get('datePlannedEnd'),
                priority: e.item.get('priority'),
                descript: e.item.get('descript')
              };
              return options.dataRepo.createItem('eventControl@sakh-pm', data, null, {user: e.user});
            });
        }
        return Promise.resolve();
      }
    );
    options.workflows.on(
      ['proposal@sakh-pm.cancel'],
      (e) => {
        if (e.transition === 'curatorToCancel') {
          return options.dataRepo.editItem(e.item.getMetaClass().getCanonicalName(), e.item.getItemId(),
↪{archive: true})
            .then(
              item => collectionToArchive(item, 'proposals')
                .then(() => collectionToArchive(item, 'eventBlock'))
                .then(() => collectionToArchive(item, 'project'))
            );
```

(continues on next page)

(continued from previous page)

```
        }
        return Promise.resolve();
      }
    );
  };
}

module.exports = WorkflowEvents;
```

This script describes the actions to be performed when changing the status of the workflow assignmentBasic@sakh-pm to fin, and the status of the workflow proposal@sakh-pm to‘‘cancel‘‘.

3.1.2 Core key functions

dataRepo

//dataRepo info from

1. coreimpldatarepositoryionDataRepository.js
2. coreinterfacesDataRepositoryDataRepository.js
3. coreinterfacesMetaRepositoryMetaRepository.js
4. coreimplmetaDsMetaRepository.js

#. coreiterfacesDataSource.js #.

coreimpldatasourcemongodb.js // supported calls:

1. wrap(className, data, [version], [options]) supported options: user . . .
2. setValidators(validators[]) . . .
3. getCount(obj, [options]) supported options: filter

Returns the number of objects of the obj class in the database. . . .

1. getList(obj, [options]) supported options: filter offset count sort countTotal nestingDepth env user

Returns a list of objects of the obj class in the database. . . .

1. getIterator(obj, [options]) supported options: filter offset count sort countTotal nestingDepth env user

presumably https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/ . . .

1. aggregate(className, [options]) supported options: user expressions filter groupBy

presumably https://docs.mongodb.com/manual/aggregation/

. . .

1. rawData(className, [options]) supported options: user filter attributes distinct

https://docs.mongodb.com/manual/reference/method/db.collection.find/ . . .

1. getItem(obj, [id], [options]) supported options: filter nestingDepth user . . .
2. createItem(className, data, [version], [changeLogger], [options]) supported options: nestingDepth skipResult adjustAutoInc user . . .
3. editItem(className, id, data, [changeLogger], [options]) supported options: nestingDepth skipResult adjustAutoInc user . . .

4. saveItem(className, id, data, [version], [changeLogger], [options]) supported options: nestingDepth autoAssign skipResult adjustAutoInc user ...
5. deleteItem(className, id, [changeLogger], [options]) supported options: user ...
6. put(master, collection, details, [changeLogger], [options]) supported options: user ...
7. eject(master, collection, details, [changeLogger], [options]) supported options: user ...
8. getAssociationsList(master, collection, [options]) supported options: filter offset count sort countTotal nestingDepth user ...
9. getAssociationsCount(master, collection, [options]) supported options: filter offset count sort countTotal nestingDepth user ...
10. bulkEdit(classname, data, [options]) supported options: filter nestingDepth forceEnrichment user ...
11. bulkDelete(classname, [options]) supported options: filter user
12. recache(item, [options]) ...

MetaRepo

TODO

Workflow

TODO

3.1.3 Using templates for data entry fields in a web form

Templates are used to set custom parameters for constructing data entry fields in a web form.

To connect a template to a web form field, you must specify it in the options of the corresponding field of the json form:

```
{
  "tabs": [
    {
      "caption": "General info",
      "fullFields": [
        {
          "property": "surname",
          "caption": "Surname",
          //...
          "options": {
            "template": "capitalize"
          },
          "tags": ""
        },
```

.ejs‘ templates are loaded by the path specified in modules.registry.globals.templates from deploy.json.

For example, if the path applications/khv-ticket-discount/templates/registry is specified, then for the previous example in the property field the script applications/khv-ticket-discount/templates/registry/capitalize.ejs will be loaded.

The script will be executed when the field is loaded in the web form. The syntax is standard for ejs.

Some elements of the web form are available inside the script, for more details see: Options </3_development/metadata_structure/meta_view/meta_view_attribute/options.rst> _.

Example of a script for automatically replacing lowercase letters with uppercase letters and the letters “ö” with “o” in a text field:

```
<% wfState = item.base.state %>
<div class="form-group <%= wfState === 'edit' || item.id === null ? (field.required ? 'required' : '') : ''
↪%> " style data-type="<%= wfState === 'edit' || item.id === null  ? 'input' : 'static' %>" data-name="
↪<%= prop.getName().toLowerCase() %>" data-prop="<%= JSON.stringify(field) %>" >
   <label for="a_khv-ticket-discount_<%= item.getClassName().split('@')[0] %>_<%= prop.getName().
↪toLowerCase() %>" class="col-md-2 col-sm-3 control-label"><%= prop.getCaption() %>
   </label>
    <div class="col-sm-9">
       <input id="a_khv-ticket-discount_<%= item.getClassName().split('@')[0] %>_<%= prop.getName().
↪toLowerCase() %>" type="<%= wfState === 'edit' || item.id === null ? 'text' : 'hidden' %>" class=
↪"form-control attr-value" name="<%= prop.getName().toLowerCase() %>" data-mask="{&amp;quot;regex&amp;quot;:&
↪quot;[öÖa-zA-Z .-]{1,50}&amp;quot;}" placeholder="<%= prop.getCaption() %>" value="<%= prop.getValue() !
↪== null ? prop.getValue() : "" %>" im-insert="true">
       <% if(wfState === 'done' && item.id !== null) { %>
          <div class="form-control-static"><%= prop.getValue() %></div>
       <% } %>
       <script>
          if (typeof inputField !== 'object') {inputField = [];}
          propName = '<%= prop.getName().toLowerCase() %>';
          inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'] =␣
↪document.getElementById(`a_khv-ticket-discount_<%= item.getClassName().split('@')[0] %>_${propName}
↪`);
          inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪addEventListener('focusout',applyStyle)
          inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪addEventListener('keyup', applyStyle)
          inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪addEventListener('keydown', applyStyle)
          inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪addEventListener('paste', applyStyle)
          function applyStyle() {
             while ((/[Ö]/).test(inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value)) {
                inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>
↪'].value = inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪value.replace(/[Ö]/, 'O');
             }
             while ((/[ö]/).test(inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value)) {
                inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>
↪'].value = inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪value.replace(/[ö]/, 'o');
             }
             while ((/[a-z]/).test(inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value)) {
                inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>
↪'].value = inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().toLowerCase() %>'].
↪value.toUpperCase();
             }
             //(/[а-я]/).test(inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value[0]) ? inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value = inputField['<%= item.getClassName().split('@')[0] %><%= prop.getName().
↪toLowerCase() %>'].value[0].toUpperCase() + inputField['<%= item.getClassName().split('@')[0] %><%=
↪prop.getName().toLowerCase() %>'].value.substring(1) : ""; - только первая буква}
```

(continues on next page)

(continued from previous page)

```
        }
      </script>
    </div>
</div>
```

3.2 Functionality

Application functionality - is a set of features (functions) that are implemented separately and independently of metadata but within an application.

Name	Description
Module templates	Divided into system and project templates.
Printed forms	Display formatting in printed form.
Scheduled jobs	Subsystems for running scheduled jobs.
Notifications	Feed of new notifications.
Data caching	Data caching provides quick access to the requested information.
Filters on form	Query for filter on the list view form.
Connect two namespaces	Connection of two projects using “namespace”.
EDS	Electronic digital signature.
Virtual attributes	Special type of attributes that allows you to display the full code and name of the class.
Utilities	Additional programs for more specialized application.

3.2.1 Module templates

Themes

Themes is a directory of the following structure:

```
/static/css     директория стилей
/static/js      директория скриптов
/templates      директория шаблонов ejs
```

Themes can be located:

- In the view directory of modules and platform - the system themes
- In the applications directory of application as a platform - the projects themes
- In the themes directory application - the project themes

Setting the current theme:

1. For a platform
    - Setting theme in the config.json of the platform
    - Setting globals.theme in the deploy.json of the application
2. For a module
    - Setting theme in the config.json of the module

- Setting Module name.globals.theme in the deploy.json of the application

The default system theme is default (in the platform and modules “registry”, “geomap”, “report”).

Set the path to the theme directory in the theme field according to the following rules:

1. The absolute path is taken as it is.
2. A relative path is taken relative to the system paths in the following order:
   - Relative to the view directory of module and platform
   - Relative to the applications directory of the platform
   - Relative to the platform directory

Example in dnt

```
"geomap": {
    "globals": {
      "theme": "develop-and-test/themes/geomap",
```

3.2.2 Printed forms

Printed forms Word

- The docxtemplater library is used
- See the examples of the connecting and using docxtemplater library.

Format transfer options

The table_col parameter is to transfer the formatting. See the rulles and examples here.

Type:

```
${table_col:коллекция:разделитель:формат}
```

Example:

```
${table_col:list.instructions.limit:;:DD.MM.YYYY}
```

Result:

```
30.08.2017;06.09.2017
```

The format allows the use of the : symbol.

Displays the value of the sum in upper case

There is a toWordsfilter for docx templates, which converts to text by default. If you add “true” as the second parameter , then a ruble format will be added (rubles - kopecks).

Example:

```
{costing.costExp | toWords:true}
```

As a result, the value of the “costExp” attribute equals 345.52. The result in written form is = Three hundred and forty five rubles fifty two kopecks.

Conversion between date and string

The following functions are available:

- date - convert string to date
- upper - string to upper case
- lower - string to lower case

In the export to docx, the following filters are available in expressions:

- lower - to lower case
- upper - to upper case
- dateFormat - convert date to string, examples of use:
  - {now | dateFormat:en}
  - {since | dateFormat:en}
  - {date | dateFormat:en:YYYYMMDD}
- toDate - string to date

Current date value - _now

```
{_now} г.
```

Setting to display field values from an array of objects

If you need to display fields from an array of objects (collection for example), use the tag:

```
${table_col:list.collection.attrFromCollection}
```

By default, the values will be connected by a semicolon. To indicate another separator, specify it after the second colon:

```
${table_col:list.collection.attrFromCollection:разделитель}
```

3.2.3 Scheduled jobs subsystem

The launch of the scheduled jobs is performed in two ways:

1. in a separate process using the bin/schedule.js script
2. within the ION web application process (bin/www) by specifying the option jobs.enabled=true in the ini-file

In the second case, job management is possible to implement in a web application. Jobs are configured in the deploy.json file of the applications in the global settings section as a jobs parameter.

Example:

```
"jobs": {
  "dummy": {
    "launch": { // Периодичность запуска задания
      "month": [2,5], // в феврале и мае
      "week": 3, // каждую третью неделю (month и week  - взаимоисключающие настройки),
      "weekday": [1, 3, 5], // по понедельникам, средам и пятницам
      "dayOfYear": 5, // раз в 5 дней в течение года,
      "day": 10, // раз в 10 дней в течение месяца
      "hour": 3, // раз в 3 часа
      "minute": [10, 15, 35], // на 10-ой, 15-ой и 35-ой минуте
      "sec": 10 // раз в 10 секунд
    },
    "di": { // скоуп задания
      "dummy": {
        "module": "applications/develop-and-test/jobs/dummy",
        "options": {
        }
      }
    },
    "worker": "dummy", // имя компонента из скоупа задания, который будет исполняться
  }
}
```

A component can be specified as a starting job, in this case it should have the run method. A function can also be specified as a starting job. Then in di it is described similarly to the component, but using the executablesetting:

```
"di": {
  "dummy": {
    "executable": "applications/develop-and-test/jobs/dummy",
    "options": {}
  }
}
```

Example:

The jobs field in the global settings.

```
...
"jobs": {
      "dummy": {
```

(continues on next page)

(continued from previous page)

```
      "launch": {
        "sec": 30
      },
      "worker": "dummy",
      "di": {
        "dummy": {
          "executable": "applications/develop-and-test/jobs/dummy"
        }
      }
    }
...
```

3.2.4 Notifications

Notifications - is displayed on the form as a red bell, when clicked, the notification feed (the last 10 unread) is displayed, when clicked on the notification, it disappears as read. The feed is updated every 15 seconds. If the bell is missing, then there are no new notifications.

Setting up notifications

The feature of sending notifications to mail is implemented. Configure it in the project - TODO.

The API for sending notifications programmatically is implemented. That means that you can send a notification in the event handler. In the admin panel when sending notifications the list of full user names (user @ local) separated by a space is indicated in the "Recipients" field.

Notifications are stored in the following collections: ion_notification, ion_notification_recievers.

3.2.5 Data caching

Principle of work

Between the basic data repository dataRepo and the data repository with security checks securedDataRepo, a data repository with support for caching cachedDataRepo has been implemented. The data is loaded by chain: BD -> dataRepo -> cachedDataRepo -> securedDataRepo.

However, when running a query, the cachedDataRepofirst checks for the availability of data in the cache (if the cache is enabled), and if there is no data in the cache, it requests it from dataRepo, then it places it in the cache and returns it to securedDataRepo.

Setting caching of objects of individual classes

In the configuration file (deploy.json), in the options cachedDataRepo you can individually specify the list of classes whose data should be stored in the cache.

Example:

```
...
"options": {
  "cachedClasses": ["class1@namespace", "class2@namespace", "class3@namespace"]
}
...
```

If cachedClasses is not specified, then all data are cached.

Caching a list of objects

Both individual objects and lists are cached.

When a list is cached based on its selection conditions (including pagination), a key is generated, then a selection is performed and all received objects are cached individually, and the list is cached as an array of object identifiers. When selecting a list from the cache, on the basis of this array, the corresponding objects are selected from the cache and the resulting list is formed.

Similarly, all objects by reference and eagerly loaded collections are cached.

Setting the adaptive cache

For advanced configuration of cache (such as, query storage time, depth of storage of objects in the cache, etc.) the setting of adaptive object caching is applied. Adaptive cache settings are specified in the deploy.json file in the "di"property, in the memcached and redis components in the connectOptions option. All the necessary placeholders are listed in the config.json file of the platform repository.

NB: The storage depth of objects in the cache corresponds to the depth of the query for objects from the database. So, the object stores information about its references, and then in the cache the object graph is obtained, and not the tree.

Setting the cache by use of the ini-file

The ini-file parameters:

```
...
cache.module=redis // - модуль используемый репозиторием данных для кеширования.
//(варианты: ion://memcached, ion://redis, ion://innerCache)

cache.memcached.enabled=true // активируем memcached
cache.memcached.location1 // сервер1
cache.memcached.location2 // сервер2
cache.memcached.timeout // таймаут обращения к кэшу

cache.redis.enabled=true // активируем redis
cache.redis.host=127.0.0.1 // хост redis
cache.redis.port=6379 // порт redis

session.type=redis // хранение сессий авторизации в redis
```

Cache is not used by default.

3.2.6 Filters on the list view form

```
Если для даты значение в поле фильтра и значение в поле атрибута имеют разный формат, то фильтр по
↪такому полю работать НЕ БУДЕТ
```

The query for the filter is specified by an expression (search query).

The available operations:

- using brackets for grouping
- logical: AND, OR, NOT
- comparison: =, <, >, <=, >=, <>
- arithmetic: +, -, *, /
- string: like
- over collections: size

Create a query

Choose attribute from the drop-down list using the > _button located at the bottom of the filter request field. The name of the attribute is shortened in “backticks” i.e.:

```
`Наименование атрибута` != 2
```

Combination options of attribute values for a query:

- and - necessarily both (or more) values,
- or - any of the values of both (or more) values.

Example of a combination:

```
`Атрибут1` = 1 AND `Атрибут2` != 2
```

String values of attributes when forming the request, are wrapped in double quotes:

```
`Название поля` != "привет"
```

Accessing attributes by reference:

```
`Атрибут1`.`Атрибут по ссылке из Атрибут 1` = "значение"
```

Hints:

At the end of the filter query field, there is a ?sign, when clicked, a model window opens describing how the filter and the query syntax for it works.

This library (https://nearley.js.org/) is used to parse search expressions.

Options for using

In addition to the button next to the search bar at the top of the page, you can call the filter by clicking on the similar icon located in each column of the table.

To create a request for the filter, select a value from the drop-down list, or start typing a value in a row. As soon as the value is selected, you need to press the Enter key - the result of the query is displayed in the value column.

3.2.7 Connection of two namespaces

Connection of two projects using “namespace”

It is used if you need to link two projects together and deploy them on the same platform, with references and collections not only within the project but also to others deployed in the same context. Implemented using the project namespace, so it would be clear to which project the classes refer by reference. For example, we associate the test project develop-and-test with the project fias:

Example


```
{
  "namespace": "develop-and-test",
  "code": "projectJoin.addressExt",
  "orderNumber": 0,
  "type": 1,
  "title": "",
  "caption": "Адрес ФИАС",
  "classname": "address@fias",
  "container": null,
  "collection": null,
  "url": null,
  "hint": null,
  "conditions": [],
  "sorting": [],
  "pathChains": [],
  "metaVersion": "2.0.61.21119"
}
```


In the develop-and-test project menu, there is a navigation node, which is a class representation from the fias project. Thus, when navigating through the menu item of one project, we obtain data from another project, if the required namespace is specified.

Description

For meta:

When importing, the namespace is taken not from the parameter, but from the meta class. We remove namespace as a navigation parameter, etc.

When importing the navigation meta, if the system names match, the meta that is rolled last takes precedence, but taking into account the fullness of the attributes. i.e., an empty attribute value does not override a non-empty one.

For Studio:

The namespace property is added in the class meta. In the studio, the “namespace” parameter has been added to the ION project settings. By default, this namespace is entered in the corresponding field on the class creation form.

The “namespace” attribute is also set to separate fields on the forms for creating and editing a class. In the studio, the system class name is now displayed everywhere with a namespace through a @sign if the class is declared in a namespace other than the namespace of the current project.

Namespace field is in the form of a drop-down list, from where you can choose either a project’s namespace or project-related namespaces (Project references). At the same time there is an opportunity to set it manually.

The rest of the meta objects has no namespaces, but links to classes everywhere must be affixed with the account of namespaces (in the reference attributes too). This means that the class selection list must contain the full name of the class with a namespace.

Also, the class selection lists should include classes from all Project References. At the physical level, the namespace is included in the file name of the class meta (and directories, if they are involved in the formation of the class binding logic, for example, in the view meta).

3.2.8 Electronic digital signature

Description

Electronic digital signature (EDS) - is a detail of an electronic document intended to protect this electronic document from forgery, obtained as a result of cryptographic transformation of information using the private key of an electronic digital signature. It allows you to identify the owner of the certificate key signature and establish that there is no distortion of information in the electronic document.

Purpose of use

In the application the EDS may be used for:

- Data integrity checking
- Data authorship establishment

There are three types of digital signatures that differ in their use:

- Simple electronic digital signature
  - to establish the authorship of the data
  - created with the use of codes, passwords or other instruments
- Reinforced unqualified electronic digital signature
  - to check data integrity
  - to establish the authorship of the data
  - created using electronic signature tools
- Reinforced qualified electronic digital signature
  - to check data integrity
  - to establish the authorship of the data

  - to create and verify an electronic signature, electronic signature tools are used that have received confirmation of compliance with the requirements of the legislation

Work specifics

The EDS utility works on the cryptoPro basis, so it should be installed on the computer:

- install cryptopro
- install cryptopro plugin
- issue a certificate for testing

Implementation

EDS can be attributed to the application utilities, since its main implementation is in the application. Usually the implementation of EDS is located in the lib/digest application folder (for example, the project-management app):

- lib/digest/digestData.js - check the loading object form to the need for an electronic signature (_applicable) and check the signature process when performing a WF transition (_process)
- lib/digest/signSaver.js - attachment of the signature to the object

Add the signedClasses setting in the deploy file for the registry module, so that EDS status can be displayed.

Example

```
"modules": {
   "registry": {
     "globals": {
       "signedClasses": [
        "class@application"
       ],
...
```

In the workflows/indicatorValueBasic.wf.json workflow add a transition with the "signBefore": true property.

Example

```
{
    "name": "needAppTrs_sign",
    "caption": "На утверждение",
    "startState": "edit",
    "finishState": "onapp",
    "signBefore": true,
    "signAfter": false,
    "roles": [],
    "assignments": [
      {
        "key": "state",
        "value": "onapp"
      }
    ],
```

(continues on next page)

(continued from previous page)

```
  "conditions": []
}
```

3.2.9 Virtual attributes

Description

Virtual attributes - a special type of attributes, specified in the property attribute name starting with‘‘__‘‘. The functionality of each virtual attribute is different, there are the following virtual attributes:

- __class - displays the full code of the class in which the attribute was set.
- __classTitle - displays the name of the class in which the attribute was set.

3.2.10 Utilities

Description

Utilities - additional programs for more specialized use.

Utilities of core

The utilities of the core are the most necessary utilities from the installation to the operation of the application and are located in the bin folder. Before running them, specify the environment variable NODE_PATH, if it is not set in advance (with an indication of the core folder).

The core includes utilities:

- bin/acl.js - to import and edit application security settings (resources, roles, users, access rights). Launch parameters:
  - --u - user
  - --res - resource
  - --role - role
  - --p - access right
  - --m - access application method
  - --d - directory with the security settings for import
- bin/adduser.js - to add new users to the application's security settings. Launch parameters:
  - --name - user login (admin - by default)
  - --pwd - user password (admin - by default)
- bin/bg.js - to run low priority background procedures that require high power from the processor or run relatively long time
- bin/export.js - to export the application to a local directory. Launch parameters:

  - --dst - path to the directory where the export result will be written (by default ../out)
  - --ns - application namespace
  - --file-dir - path to the directory to which files from file attributes will be exported
  - --acl - optional export of security settings
  - --nodata - skip export for all created objects in the application
  - --nofiles - skip file attribute export
  - --ver - version (latest version - last)
- bin/import.js - to import the application metadata. Launch parameters:
  - --src - path to the directory from which the import will take place (by default ../in)
  - --ns - application namespace
  - --ignoreIntegrityCheck - data integrity ignored during import
- bin/import-data.js - to import the application data. Launch parameters:
  - --src - path to the directory from which the import will take place (by default ../in)
  - --ns - application namespace
- bin/job-runner.js - to run scheduled jobs
- bin/job.js - to run the job component from the job-runner utility
- bin/meta-update.js - to convert application meta from one version to another
- bin/schedule.js - for manual start of scheduled jobs
- bin/setup.js - to set the deploy settings from the application. Launch parameters:
  - --reset - preliminary reset of all deploy settings in the application
  - --sms - when resetting the settings, the deploy settings that are marked as important are not deleted
  - --rwa - to override, not add to arrays in deploy settings

Application utilities

Application utilities implement specific application functionality during the operation phase, which has not yet been implemented in the kernel in a unified form for various applications. Usually, these utilities are stored in the lib directory and connect to the app via deploy.

Examples of implemented utilities for the sakh-pmapplication:

- lib/actions/createIndicatorValueHandler.js - utility that creates values in the collection for the selected period
- lib/actions/createProjectReportsHandler.js - the utility that automatically creates printed forms for the project saving files in the cloud
- lib/actions/assignmentToEventOnly.js - utility that forms a checkpoint from an instruction

App utility creation using an example of createIndicatorValueHandler

Implementation

For implementation, as a rule, JavaScript is used with the use of the available functionality of the modules included in the application.

When implementing utilities in an application with relatively large functionality, the files themselves can be split into several dependent files.

In this example in the utility main file lib/actions/createPlanIndicatorsHandler.js the export should be the last line:

```
module.exports = CreatePlanIndicatorsHandler;
```

Connecting to the application

Set the connection parameters in the deploy to start the utility when using the application.

For example, you first need to add in the meta view an interface element for the utility that will launch the utility. Add the CREATE_INDICATOR_VALUEbutton in the views/indicatorFinancial/item.json file:

```
{
    "id": "CREATE_INDICATOR_VALUE",
    "caption": "Сформировать собираемые значения",
    "visibilityCondition": null,
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
}
```

Then add the settings in the deployfile to connect the CREATE_INDICATOR_VALUE button in the UI with the createIndicatorValueHandlerutility:

```
"modules": {
  "registry": {
    "globals": {
      "di": {
        "createIndicatorValueHandler": {
          "module": "applications/sakh-pm/lib/actions/createIndicatorValueHandler",
          "initMethod": "init",
          "initLevel": 2,
          "options": {
          "data": "ion://securedDataRepo",
          "workflows": "ion://workflows",
          "log": "ion://sysLog",
          "changelogFactory": "ion://changelogFactory",
          "state": "onapp"
          }
        },
        "actions": {
          "options": {
          "actions": [
            {
```

(continues on next page)

(continued from previous page)

```
            "code": "CREATE_INDICATOR_VALUE",
            "handler": "ion://createIndicatorValueHandler"
            }
        ]
    }
}
```

In this example all settings are held for the registry module, since the utility will be called out of it in the form of the indicatorFinancial object by clicking the CREATE_INDICATOR_VALUE button .

Additional info

Module settings in deploy.json

Meta view - Commands

3.3 Localization

The localization of the framework, modules and applications is organized in the i18n folders in the corresponding directories.

To translate a string for a specific locale, you need to add it and its translation to:

```
<корень фреймворка, приложения или модуля>/i18n/<код локали>/index.js
```

to the exported object, for example:

```
module.exports = {
    'Контрольная панель': `Control panel`
};
```

The string ‘Контрольная панель’ in Russian, when loaded will be translated as ‘Control Panel’

3.4 Metadata structure

3.4.1 Meta class - general part

Criteria of abstraction

Criteria of abstraction for a class is required when you need to display a list of the selection of its heirs based on the attribute reference to the base class. When forming the selection list of classes for creating an object, do not include abstract classes in the list. Set the true value in the “abstract” field.

Example

```
{
  "name": "SomeClassName",
  "abstract": true
}
```

The class becomes unavailable for initialization at the UI level.

Example

```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "value|[|plannedValue|](|dateStart|-|dateEnd|)",
  "name": "indicatorValueBasic",
  "version": "",
  "caption": "Значения показателей на период",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "abstract": true,
  "compositeIndexes": [
    {
      "properties": [
        "indicatorBasic",
        "dateStart",
        "dateEnd"
      ],
      "unique": true
    }
  ],
  "properties": [
...
```

Inheritance

Inheritance allows you to create a new meta class based on a parent one with all its attributes. A parent class may have several heirs with different attribute structures. Each heir can have its own individual set of attributes, plus attributes of the parent class.

Attributive structure

In the meta of the parent class the attributive structure is formed in such a way that each attribute can be used in any heir classes. Whereas the attributive structure in a heir class is individual for each heir class and has no influence on other classes.

Example

Parent class:


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "name",
  "name": "organization",
  "version": "",
  "caption": "Организация",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": [],
  "properties": [
```


Child class:


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "medicalOrg",
  "version": "",
  "caption": "Медицинская организация",
  "ancestor": "organization",
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": [],
  "properties": [
```


"id" is a unique identifier for all heirs. It is stored in the parent class.

"name" - the class has the “Name” attribute, which can be displayed in all the heirs, or in one of them. If the "name" attribute is set in the parent class, then it can be displayed in any of the heirs. But if "name" is set in the class of the heir, it is displayed only in the class in which it was set.

Class view:

If the abstractness criteria is set for the parent class, then it is not necessary to set the representation for it.

The class view is set for each heir separately specifying the desired attribute structure (attributes of the new class plus attributes of the parent class, if needed).

Setting the list of the heirs for creating objects by reference

Set in the meta class for the attribute of "reference"/\"collection" type after specifying the reference/collection class:

```
"itemsClass": "event",
"allowedSubclasses": [
        "Subclasses1",
        "Subclasses2"
    ],
```

itemsClass - collection for the parent class - [event]

Subclasses1 is a child class of the parent class [event], which will be displayed in the list when creating an object by reference (hereinafter you can list all the heir classes that need to be displayed in the list).

NB: If this setting is not specified, then upon creation, all child classes will be displayed in the list.

Conditions for applying the setting:

- The attribute type is “Collection” or “Reference”;
- For the attribute of the “Collection” or “Reference” type, the class of the collection/referece to the parent (base) class is specified (when creating an object of the reference class, a window for selecting several classes is displayed);
- In addition to hiding the base class, when creating an object, you do not need to display all child classes in the class selection list for creating an object by reference.

Example

The parent class [Events] has several classes of heirs ([Event1], [Event3], [Event2]). In the [Project] class, there is an attribute of the “Collection” type that refers to the parent class [Event]:

```
{
    "namespace": "ns",
    "isStruct": false,
    "key": [],
    "semantic": "",
    "name": "project",
    "version": "",
    "caption": "Проект",
    "ancestor": "",
    "container": null,
    "creationTracker": "",
    "changeTracker": "",
    "creatorTracker": "",
    "editorTracker": "",
    "history": 0,
    "journaling": true,
    "compositeIndexes": [],
    "properties": [
      {
        "orderNumber": 80,
        "name": "event",
```

(continues on next page)

(continued from previous page)


```
        "caption": "Мероприятия",
        "type": 0,
        "size": null,
        "decimals": 0,
        "allowedFileTypes": null,
        "maxFileCount": 0,
        "nullable": true,
        "readonly": false,
        "indexed": true,
        "unique": false,
        "autoassigned": false,
        "hint": null,
        "defaultValue": null,
        "refClass": "",
        "itemsClass": "event@ns",
        "allowedSubclasses": [
            "event1",
            "event2"
        ],
        "backRef": "",
        "backColl": "",
        "binding": "",
        "semantic": null,
        "selConditions": [],
        "selSorting": [],
        "selectionProvider": null,
        "indexSearch": false,
        "eagerLoading": false,
        "formula": null
    }
  ...
```


If the abstractness setting is set for the class, then when creating an object of the [Event] class in the collection, the selection list will display those inheritors of the [event] class that are specified in the "allowedSubclasses" property. That is, based on the example, only objects of the class “Event1” and “Event2” can be created in the “Events” collection.

Multilevel hierarchy

The child classes can inherit the set of the attributes not only from their direct parent classes but also from those that are higher in the inheritance hierarchy.

Example

[basicObj] - parent class ->> [eventBasic] - heir class of the [basicObj] class ->> [eventBlock] - heir class of the [eventBasic] class.


```
{
  "isStruct": false,
  "key": [
    "guid"
  ],
  "semantic": "name",
```


(continues on next page)

(continued from previous page)


```
  "name": "basicObj",
  "abstract": true,
  "version": "31",
  "caption": "Учетный объект",
  "ancestor": null,
  "container": null,
  "creationTracker": "createDatet",
  "changeTracker": "modifeDate",
  "creatorTracker": "creator",
  "editorTracker": "editor",
  "history": 0,
  "journaling": true,
  "compositeIndexes": null,
  "properties": [
```

```
{
  "isStruct": false,
  "key": [
    "guid"
  ],
  "semantic": "name| ( |code| )",
  "name": "eventBasic",
  "version": "31",
  "caption": "Базовое мероприятие",
  "ancestor": "basicObj",
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": null,
  "abstract": true,
  "properties": [
```

```
{
  "isStruct": false,
  "key": [
    "guid"
  ],
  "semantic": "name| ( |code| )",
  "name": "eventBlock",
  "version": "31",
  "caption": "Блок мероприятий",
  "ancestor": "eventBasic",
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": null,
  "properties": [
```


The [eventBlock] heir will inherit the set of the attributes of the [basicObj]parent class, as well as the [eventBasic] heir class.

Indexing

Indexing is compound unique fields. Used to search and manage the data integrity.

Composite indexing is located in the "compositeIndexes"field, which allows you to set the requirements of the unique field combinations.

Description

Specify the composite index by listing its attributes and indicating the "unique": true property. When a composite index is present in a class, when the object is saved, the DB checks the identical combinations of the listed fields. That is, the values of the "protocol" and "family" fields from the example below can be repeated, but a pair of values is always unique.

Example:

```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "protocol|family",
  "name": "refusal",
  "version": "",
  "caption": "Письменные отказы граждан",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": [
    {
      "properties": [
        "protocol",
        "family"
      ],
      "unique": true
    }
  ],
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 0,
      "size": 24,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": true,
      "indexed": false,
      "unique": true,
```

(continues on next page)

(continued from previous page)


```
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "protocol",
    "caption": "Протокол заседания комиссии",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 31,
    "name": "family",
    "caption": "Семья, поставленная на учет",
    "type": 13,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
```


(continues on next page)

(continued from previous page)


```
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "family",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```

Journaling

Journaling indicates the need to log all actions performed on the object. It is located in the journaling field of the main part of the class meta. A value of true indicates that logging is required. If the field is set to false or not present, then journaling of object changes is not required.

Example:


```
{
  "isStruct": false,
  "key": "okato",
  "semantic": "name",
  "name": "naselenniyPunkt",
  "caption": "Населенный пункт",
  "journaling": true,
  "ancestor": null,
  "container": "",
  "creationTracker": "",
  "changeTracker": "",
  "properties": []
}
```

Key fields

Each class must have a key attribute (the field "key" of the main part of the meta class). Without this, the application will not function.

Types of key fields

1. Guid - “Global identifier [12]”.
2. “String [0]”.
3. “Integer [6]”.

Key attribute requirements

When creating the key attribute meta, set the "unique" and "autoassigned" fields to true. Disable the empty value by setting "nullable" to false.

If the attribute is not generated automatically, then "autoassigned" can be set to false, then the field must be set by the operator when creating. For example, if it is a classifier code or registration number that is specified externally (on paper).

Versioning

Versioning allows you to store multiple versions of metadata. When you change and save the object, it gets the version that contains proceeded changes. So, versioning provides the ability to work with different versions of the same objects. It greatly reduces the work required to manage the changes that occur with an object.

Versioning is set in the "version" field of the general part of the meta class. Add the version property (” version “: 2) to the attribute to change the meta version.

Principle of work

When loading metadata, if there is a version attribute (“version”: 2), the meta with the version will be uploaded, otherwise version = 1.

```
{
  "isStruct": false,
  "key": "id",
  "semantic": "caption",
  "name": "ion_filter",
  "caption": "Фильтры",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "version" : 2
}
```

When creating objects, the latest version of the metadata from the current database will be added to them, and when editing objects, they will be edited based on the saved version.

Example of saved objects with different versions in the database:

```
{
    "_id" : ObjectId("567cfa1eb869fc2833690ea4"),
    "id" : 24006,
    "class" : "ALL",
    "caption" : "11",
    "html" : "",
    "filter" : "[\"{\\\"property\\\":\\\"contact\\\",\\\"operation\\\":20,\\\"value\\\":\\\"11\\\",\\\"title\\\
↪":\\\"Контактная информация содержит 11\\\",\\\"type\\\":7}\"]",
    "period" : "2015-12-08,2016-02-05",
    "version" : 1,
    "semanticTitle" : "11 "
}

{
    "_id" : ObjectId("56944e5cb73f51ec182c7369"),
    "class" : "ALL",
    "caption" : "ffffff",
    "filter" : "[\"{\\\"property\\\":\\\"class\\\",\\\"operation\\\":0,\\\"value\\\":\\\"fff\\\",\\\"title\\\":\\\
↪"Класс фильтра равно fff\\\",\\\"type\\\":1}\"]",
    "version" : 2,
    "id" : NaN,
    "semanticTitle" : "ffffff "
}
```


Realization in code

When reading the class meta, the data is separated by version. The names of the versioned classes have names that look like "<nameof class><version number>". For example, ion_filter_1, ion_filter_2 is the ion_filter class of versions 1 and 2, respectively.

When selecting objects, the data is taken based on the version. The version of the object is passed as the version parameter of the request to open the object.

Semantics

Semantics - is used to output a class object as one string of the class title.

The "semantic" field is used twice in the meta class:

1. in the general part of the meta class, where it forms a string view for this class;
2. in the class attribute meta, where it forms a string representation of the objects of the class referenced by the attribute, i.e. used for reference attributes.

Purpose of use

"Semantics" is used to adjust the display of attributes and attribute properties in the list. In attributes that display tabular data, semantics are used to limit the output of columns.

Examples of use in reference attributes

For example, there is a class, which has attributes: id, name, link, date. There is a second class - classTable, which has a reference attribute table on the class class. Without using the semantics in objects of the classTable class, in the table attribute only the values of identifiers of objects of the class will be displayed. Attributes used as identifiers are listed in the class meta class.

To display the values of the name and link attributes in the tableattribute, and not the values of identifiers, you need to write "semantic": "name|link". Depending on the type of attribute, the result will be different:

- if the table attribute is a reference, then the values of the name and link attributes separated by spaces will be filled in. Here you can use additional words and expressions using the || sign, for example " semantic ":" name|, |link" or "semantic":"The object has 2 attributes:|name|, |link";
- If the table attribute is a collection of objects of the classclass, then it will display the name and link columns.

Display format in semantics

- You can trim the output with:[].

```
"name[0,50]|..."
Указываем позицию и количество выводимых букв из семантики атрибута. Из атрибута name␣
↪выводим 50 символов семантики (значение атрибута), начиная с первого.
```

- Dereferencing is available via ‘‘. ‘‘ i.e. access to the nested object.

```
"semantic": "digitalTV|kachestvoCTB|analogTV.name|kachestvoAnal|period"
где ``analogTV`` - ссылочный атрибут класса, для которого задается семантика, а ``name`` -␣
↪атрибут класса по ссылке.
```

Display semantics on form

1. In the first-level lists (opened directly by the navigation node), only the value from the “caption” field of the navigation node is displayed as the title.
2. In the selection lists we display only the value from the “caption” field of the class of the list objects as a title.
3. In the edit form we display only the semantics of the object as a title.
4. In the creation form we display only the value from the “caption” field of the class as a title.
5. In the selection lists we display the following line in fine print “Selecting the value of the attribute <...> of the object <...>” above the title.
6. In the creation form, if an object is created in a collection or a reference, we display the following line in fine print “Creating an object in the collection/by reference <...> object <...>” above the title.

Time tag of created objects and committed changes

These are the following fields of the main part of the meta class:

1. "creationTracker" - Creation time tracker: allows you to save the date/time of object creation in the class, requires the corresponding class attribute "name" of which is entered in this field.
2. "changeTracker" - Метка времени изменения: позволяет сохранять в классе дату/время изменения объекта, требует наличия соответствующего атрибута класса, "name" которого и вносится в данное поле.

Example


```
{
  "isStruct": false,
  "key": "id",
  "semantic": "rtrs",
  "name": "digitTv",
  "caption": "Цифровое ТВ",
  "ancestor": null,
  "container": null,
  "creationTracker": "createDate",
  "changeTracker": "modifeDate",
  "properties": [
    {
      "orderNumber": 60,
      "name": "createDate",
      "caption": "Метка времени создания",
      "type": 9,
      "size": null,
      "decimals": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "selConditions": [],
      "selSorting": [],
      "selection_provider": null,
      "indexSearch": false,
      "eagerLoading": false
    },
    {
      "orderNumber": 70,
      "name": "modifeDate",
      "caption": "Метка времени изменения",
      "type": 9,
      "size": null,
      "decimals": 0,
      "nullable": true,
      "readonly": false,
      "indexed": false,
      "unique": false,
      "autoassigned": false,
```


(continues on next page)

(continued from previous page)

```
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "selConditions": [],
      "selSorting": [],
      "selection_provider": null,
      "indexSearch": false,
      "eagerLoading": false
    }
  ]
}
```

Метки пользователя создавшего и измененившего объект

1. "creatorTracker" - Метка пользователя создавшего: позволяет сохранять в классе имя пользователя, создавшего объект, требует наличия соответствующего атрибута класса, "name" которого и вносится в данное поле.
2. "editorTracker" - Метка пользователя изменившего: позволяет сохранять в классе имя пользователя, изменившего объект, требует наличия соответствующего атрибута класса, "name" которого и вносится в данное поле.

```
{
  "isStruct": false,
  "key": [
    "guid"
  ],
  "semantic": "name",
  "name": "basicObj",
  "abstract": true,
  "version": "31",
  "caption": "Учетный объект",
  "ancestor": null,
  "cacheDependencies": [
    "basicObj"
  ],
  "container": null,
  "creatorTracker": "creator",
  "editorTracker": "editor",
  "history": 0,
  "journaling": true,
  "compositeIndexes": null,
  "properties": [
    ...
    {
      "orderNumber": 20,
      "name": "creator",
      "caption": "Метка пользователя, создавшего объект",
      "type": 18,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
```

(continues on next page)

(continued from previous page)


```
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": true,
    "hint": "",
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "editor",
    "caption": "Метка пользователя, изменившего объект",
    "type": 18,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": true,
    "hint": "",
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
],
"metaVersion": "2.0.61"
  }
```

Тип Структура [16]

Структура - способ отображения связанных объектов (ссылок). Если у поля указано структура, то данный тип атрибута нужен для уменьшения действий при заведении модели, при заранее известных типовых классах, атрибуты которых используются во многих других классах. Для класса-структуры в основной части меты класса задается значение true в поле "isStruct".

In the meta view of creation and change use the type “Group [0]” for the attributes of the “Structure [16]” type. If you do not specify the fields in the group, they are created automatically according to the meta. In list views, there is no need to create columns for structure attributes, the object will not have such a property, but the corresponding attributes of the structure-class will be present. For them, you can add columns.

** NB: ** Objects of the class-structure are not created!

Example

Класс-структура:


```
{
  "isStruct": true,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "is_struct",
  "version": "",
  "caption": "\"isStruct\" класс-структура",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
   {
     "orderNumber": 10,
     "name": "id",
     "caption": "Идентификатор",
     "type": 12,
     "size": null,
     "decimals": 0,
     "allowedFileTypes": null,
     "maxFileCount": 0,
     "nullable": false,
     "readonly": false,
     "indexed": false,
     "unique": true,
     "autoassigned": true,
     "hint": null,
     "defaultValue": null,
     "refClass": "",
     "itemsClass": "",
     "backRef": "",
     "backColl": "",
     "binding": "",
```


(continues on next page)

(continued from previous page)


```
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "last_name",
    "caption": "Фамилия",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "first_name",
    "caption": "Имя",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
```


(continues on next page)

(continued from previous page)


```
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 40,
    "name": "patronymic",
    "caption": "Отчество",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 50,
    "name": "date",
    "caption": "Дата рождения",
    "type": 9,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
```


(continues on next page)

(continued from previous page)


```
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```


Класс с атрибутом типа “Структура [16]”:


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "struct",
  "version": "",
  "caption": "Класс \"Структура [16]\" (класс с типом атрибута 16 - структура)",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
```


(continues on next page)

(continued from previous page)


```
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
   },
   {
    "orderNumber": 20,
    "name": "struct",
    "caption": "Класс \"Структура [16]\"",
    "type": 16,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "is_struct",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
   }
  ]
}
```


Class object with an attribute of structure type in DB:


```
{
    "_id" : ObjectId("57c3e46fd53ecd50123cc4f5"),
    "struct$id" : "5f421610-6dba-11e6-874f-1b746e204b07",
    "struct$last_name" : "Мирошниченко",
    "struct$first_name" : "Ирина",
    "struct$patronymic" : "Львовна",
    "struct$date" : ISODate("1978-07-13T14:00:00.000Z"),
    "id" : "5f41ef00-6dba-11e6-874f-1b746e204b07",
    "_class" : "struct@develop-and-test",
    "_classVer" : ""
}
```

Type Structure

Structure - a way to display related objects (references). If the field has a structure specified, then this type of attribute is needed to reduce actions when creating a model, with previously known type classes, whose attributes are used in many other classes.

For the class structure in the main part of the meta class, the value true is set in the "isStruct" field.

In the meta view of creation and change use the type “Group [0]” for the attributes of the “Structure [16]” type. If you do not specify the fields in the group, they are created automatically according to the meta. In list views, there is no need to create columns for structure attributes, the object will not have such a property, but the corresponding attributes of the structure-class will be present. For them, you can add columns.

** NB: ** Objects of the class-structure are not created!

Example

Class-structure:


```
{
  "isStruct": true,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "is_struct",
  "version": "",
  "caption": "\"isStruct\" класс-структура",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
```


(continues on next page)

(continued from previous page)


```
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "last_name",
    "caption": "Фамилия",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "first_name",
    "caption": "Имя",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
```


(continues on next page)

(continued from previous page)


```
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 40,
    "name": "patronymic",
    "caption": "Отчество",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 50,
    "name": "date",
    "caption": "Дата рождения",
    "type": 9,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
```


(continues on next page)

(continued from previous page)


```
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```


Meta class with an attribute of the “Structure [16]” type


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "struct",
  "version": "",
  "caption": "Класс \"Структура [16]\" (класс с типом атрибута 16 - структура)",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
```


(continues on next page)

(continued from previous page)


```
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
   },
   {
    "orderNumber": 20,
    "name": "struct",
    "caption": "Класс \"Структура [16]\"",
    "type": 16,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "is_struct",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
   }
  ]
}
```


Class object with an attribute of structure type in DB:


```
{
    "_id" : ObjectId("57c3e46fd53ecd50123cc4f5"),
    "struct$id" : "5f421610-6dba-11e6-874f-1b746e204b07",
    "struct$last_name" : "Мирошниченко",
    "struct$first_name" : "Ирина",
    "struct$patronymic" : "Львовна",
    "struct$date" : ISODate("1978-07-13T14:00:00.000Z"),
    "id" : "5f41ef00-6dba-11e6-874f-1b746e204b07",
    "_class" : "struct@develop-and-test",
    "_classVer" : ""
}
```

The general part of the class meta - contains the fields of the class parameters that are related to the structure itself and the methods for handling data in it.

JSON


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "class_integer",
  "abstract": true,
  "version": "",
  "caption": "Класс \"Целое [6]\"",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "creatorTracker": "",
  "editorTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [...]
}
```

Field description

Code	Name	Acceptable values	Description
"isStruct"	It a structure	Logical	If the value is “true” - this class is a structure and can be used in other classes in attributes of a special kind - “Structure [16]”
"key"	Key attributes	Array of strings, at least one value.	Specify a key field that uniquely identifies the object in the collection.
"semantic"	Semantic attribute	String.	Sets the semantics - the rule of forming the row view for this class.
"name"	System name	String, only the latin characters with no spaces.	Sets the first part of the name of the meta class file, the system name.
"abstract"	Criteria of abstraction for class	Logical	Used only for parent (base) classes.
"version"	Versioning	String.	Allows to set the versioning of the meta to operate the data created in different meta versions in the same collection.
"caption"	Logic name	String.	The class name displayed in the UI
"ancestor"	Inheritance	Null or string.	A set of attributes, created in the class is inherited by successor classes. It is a way to reduce the number of entities when it is possible to use the same set of attributes. All classes-heirs will inherit the attribute set of the parent + you can make attributes belonging individually to this class-heir (if necessary).
"container"	Container reference attribute	Null or string.	Select the reference attribute that will be used to automatically build hierarchical navigation. The object to which the selected attribute will refer will be perceived by the environment as a container of the domain class instance, and automatically will build a hierarchy of objects.
"creationTracker"	Creation time tracker	String	Allows to save data/time of the object creation, requires the presence of the corresponding class attribute, the "name of which is entered into this field.
"changeTracker"	Change time tracker	String	Allows to save data/time of the object change, requires the presence of the corresponding class attribute, the "name" of which is entered into this field.
"creatorTracker"	Object creator tracker	String	Allows to save the name of the user who created the object, requires the presence of the corresponding class attribute, the "name" of which is entered into this field.
"editorTracker"	Object editor tracker	String	Allows to save the name of the user who changed the object, requires the presence of the corresponding class attribute, the "name" of which is entered into this field.
"history"	Data image	0 - none	Stores the images of data
		1 - arbitrarily	
		2 - up to an hour	
		3 - up to a day	
		4 - up to a week	
		5 - up to a	

3.4.2 Meta class - attribute part

Типы атрибутов

Collection

Общее описание

Collection - is a data type that allows lists of other objects to be displayed in one object. The data of the object can be objects of any class including the initial.

Ways to define collections:

с точки зрения используемых полей атрибутивной части меты классов

1. один-ко-многим - означает наличие контейнера и вложенного объекта с ссылкой на контейнер. В контейнере необходимо указать коллекцию, а у нее указать ссылочный атрибут вложенного объекта по которому формируется связь. См. Обратные ссылки в контексте коллекций.
2. многие-ко-многим - необходимо определить коллекцию без ссылок и указать класс вложенных элементов - связи создаются при помещении в коллекцию и хранятся как отдельные сущности в БД. См. Коллекции.
3. обратная коллекция - если коллекция многие ко многим задается на основании коллекции в другом объекте, то другая сторона связи задается через backColl. См. Обратные коллекции.

См. полное описание типа Коллекция в атрибутивной части меты класса - Атрибут коллекции и обратной коллекции.

Тип атрибута дата/время

Description

Тип атрибута дата/время - представляет собой дату в формате ISODate. Может быть отображена либо как дата, либо как дата-время.

Режимы даты и времени

Режимы хранения даты задаются в параметре mode атрибутивной части меты класса. Доступно 3 режима хранения даты - реальный, локализованный, универсальный:

- 0 - Реальный (по умолчанию). Дата хранится без информации о часовом поясе. При отображении приводится к часовому поясу пользователя. Т.е. 01.01.2017 заданное на Камчатке в Хабаровске отобразится как 31.12.2016.

- 1 - Локализованный. Дата хранится вместе с часовым поясом, в котором была задана. При отображении приводится к этому часовому поясу. Т.е. 01.01.2017 заданное на Камчатке, в Хабаровске отобразиться так же - 01.01.2017. Но правильно его отображать с указанием часового пояса, т.е. 01.01.2017 (+11). При редактировании часовой пояс обновляется часовым поясом новой даты. При этом в БД хранится РЕАЛЬНЫЙ момент времени, что нужно учитывать в условиях выборки задаваемых хардкодом в мете.
- 2 - Универсальный. Дата сохраняется как если бы она задавалась в UTC. Т.е. не приводится к UTC, а сохраняется в UTC так же, как была введена. Т.е. если мы в Хабаровске ввели 01.01.2017 по Хабаровску, то сохранится она как “01.01.2017 00:00 UTC”, а не как “31.12.2016 14:00 UTC”. Отображается в любом часовом поясе так же, как была введена. Использовать для дат в реквизитах (дата рождения, дата выдачи документа и т.д.), т.е. когда важен не реальный, а формальный момент времени. Либо, когда не нужно учитывать время вообще.

Example


```
{
  "isStruct": false,
  "metaVersion": "2.0.7",
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "class_datetime",
  "version": "",
  "caption": "Класс \"Дата/Время [9]\"",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": 24,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
```


(continues on next page)

(continued from previous page)


```
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "data_data",
    "caption": "Выбор даты [120]",
    "type": 9,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "data_datatime",
    "caption": "Реальная дата",
    "type": 9,
    "mode": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
```


(continues on next page)

(continued from previous page)


```
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "data_datatime1",
    "caption": "Дата с часовым поясом",
    "type": 9,
    "mode": 1,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "data_datatime2",
    "caption": "Универсальная дата",
    "type": 9,
    "mode": 2,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
```


(continues on next page)

(continued from previous page)


```
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```


Geodata

Тип атрибута геокоординаты - является атрибутом, который хранит координаты с уникальными представлениями для создания и редактирования. Значение свойства “type” для атрибута, имеющего тип геокоординаты равно 100.

Описание:

Если атрибут имеет тип геокоординаты и на форме создания/редактирования объекта атрибут заполнен (координаты заданы), то отображается поле карты с координатами и кнопка “Изменить”. Если атрибут не заполнен – то отображается только кнопка “Задать координаты”. Доступно только в студии

Особенности работы

Если задано свойство "autoassigned": true и не заданы данные при создании формы - то нужно определять координаты автоматически по данным адреса из атрибутов указанных в свойствах геометки.

Способы отображения на форме

Для отображения атрибута с типом геокоординаты используется тип представления геообъект. Геообъект может быть отображён на форме в одном из трёх режимов:

- Карта (0) - геообъект отображается на карте;
- Поиск по адресу (1) - геообъект отображается на карте, на которой по адресу можно найти место и переместить туда геообъект. Или же можно просто задать координаты геообъекта;
- Холст (2) - геообъект отображается на карте, можно задать интерактивное отображение геообъекта на карте.

Режим отображения задаётся указанием соответствующей цифры в поле "mode" в мете представлений класса:

“mode”: 0, – Режим отображения Карта

“mode”: 1, – Режим отображения поиск по адресу

“mode”: 2, – Режим отображения Холст

Способы хранения в БД

Данные атрибута с типом геокоординаты сохраняются в БД в виде строки со значениями через запятую, строки в формате JSON-массива или строки в формате JSON-объекта.

Пример структуры атрибута в режиме Карта (0) в JSON:

В логическом имени атрибута (caption) указан режим отображения (mode) для разделения атрибутов с одинаковым типом.


```
{
    "orderNumber": 20,
    "name": "class_geodata",
    "caption": "Геоданные [100] mode [0] - Карта",
    "type": 100,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
```


Пример структуры атрибута в режиме Поиск по адресу (1) в JSON:


```
{
    "orderNumber": 30,
    "name": "class_geodata1",
    "caption": "Геоданные [100] mode [1] - Поиск по адресу",
    "type": 100,
```


(continues on next page)

(continued from previous page)


```
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
```


Пример структуры атрибута в режиме Холст (2) в JSON:


```
{
    "orderNumber": 40,
    "name": "class_geodata2",
    "caption": "Геоданные [100] mode [2] - Холст",
    "type": 100,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
```


(continues on next page)

(continued from previous page)

```
    "formula": null
  }
```

Тип “Структура [16]”

Структура - способ отображения связанных объектов (ссылок). Если у поля указана структура, то данный тип атрибута нужен для уменьшения действий при заведении модели, при заранее известных типовых классах, атрибуты которых используются во многих других классах.

For the class structure in the main part of the meta class, the value true is set in the "isStruct" field.

In the meta view of creation and change use the type “Group [0]” for the attributes of the “Structure [16]” type. If you do not specify the fields in the group, they are created automatically according to the meta. In list views, there is no need to create columns for structure attributes, the object will not have such a property, but the corresponding attributes of the structure-class will be present. For them, you can add columns.

** NB: ** Objects of the class-structure are not created!

Example

Class-structure:

```
{
  "isStruct": true,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "is_struct",
  "version": "",
  "caption": "\"isStruct\" класс-структура",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
```

(continues on next page)

(continued from previous page)


```
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "last_name",
    "caption": "Фамилия",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "first_name",
    "caption": "Имя",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
```


(continues on next page)

(continued from previous page)


```
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 40,
    "name": "patronymic",
    "caption": "Отчество",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 50,
    "name": "date",
    "caption": "Дата рождения",
    "type": 9,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
```


(continues on next page)

(continued from previous page)


```
      "nullable": true,
      "readonly": false,
      "indexed": false,
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```


Meta class with an attribute of the “Structure [16]” type


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "struct",
  "version": "",
  "caption": "Класс \"Структура [16]\" (класс с типом атрибута 16 - структура)",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
      "unique": true,
      "autoassigned": true,
```


(continues on next page)

(continued from previous page)


```
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    },
    {
      "orderNumber": 20,
      "name": "struct",
      "caption": "Класс \"Структура [16]\"",
      "type": 16,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": true,
      "readonly": false,
      "indexed": false,
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "is_struct",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ]
}
```


Class object with an attribute of structure type in DB:


```
{
    "_id" : ObjectId("57c3e46fd53ecd50123cc4f5"),
    "struct$id" : "5f421610-6dba-11e6-874f-1b746e204b07",
    "struct$last_name" : "Мирошниченко",
    "struct$first_name" : "Ирина",
    "struct$patronymic" : "Львовна",
    "struct$date" : ISODate("1978-07-13T14:00:00.000Z"),
```


(continues on next page)

(continued from previous page)

```
    "id" : "5f41ef00-6dba-11e6-874f-1b746e204b07",
    "_class" : "struct@develop-and-test",
    "_classVer" : ""
}
```

Links

Общее описание

Ссылка - тип данных, который хранит в себе ссылку на другой объект. Данный объект может быть объектом исходного класса, если ссылка указывает на класс, в котором она определена.

Значение в поле атрибута типа “Ссылка” выводятся в соответствии с семантикой, указанной для класса по ссылке. Главное отличие от типа “Коллекция” в том, что в поле атрибута типа “Ссылка” может отображаться и храниться только один объект класса по ссылке.

См. полное описание типа Ссылка в атрибутивной части меты класса - Атрибут ссылки и обратной ссылки.

Способы задания ссылок:

Способы задания ссылок с точки зрения используемых полей атрибутивной части меты классов:

1. один-к-одному - означает наличие ссылки и вложенного объекта с ссылкой-связкой на исходный объект. В ссылке необходимо указать ссылку-связку, а у нее указать ссылочный атрибут вложенного объекта по которому формируется связь. Обязательно в атрибуте-ссылке указать свойство уникальности - true. См. Обратные ссылки в контексте ссылок.
2. один-ко-многим - необходимо определить ссылку и указать класс вложенного элемента - связи создаются при помещении в ссылку и хранятся как отдельная сущность в БД. См. Ссылки.

Расписание

Расписание - тип данных реализующий хранение периодов времени или периодичность выполнения регулярных событий.

Режим отображения атрибута на форме:

В представлении для расписания предусмотрено два типа полей: SCHEDULE = 210 – расписание отображается в табличном виде CALENDAR = 220 – расписание отображается в виде календаря

Пример структуры атрибута в табличном виде:

```
{
        "caption": "Расписание [210]",
        "type": 210,
        "property": "schedule",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": null,
        "orderNumber": 20,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": "",
        "historyDisplayMode": 0,
        "tags": null
    }
```


Пример структуры атрибута в виде календаря:


```
{
        "caption": "Календарь [220]",
        "type": 220,
        "property": "calendar",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": null,
        "orderNumber": 30,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": "",
        "historyDisplayMode": 0,
        "tags": null
      }
```

Пример хранения расписания в БД:

Периоды времени задаются как объекты периодичности в атрибуте occurs с указанием атрибута duration. Пропуски во временном периоде указываются в атрибуте skipped.

```
[
  {
    "description": "Рабочие часы",
    "item": "develop-and-test@WorkTime@12345", // Ссылка на объект данных
    "occurs": [ // происходит
        {
          "hour": 9, // на 9 час суток
          "duration": 14400 // длится 4 часа (4 * 60 * 60)
        },
        {
          "hour": 14, // на 14 час суток
          "duration": 14400 // длится 4 часа (4 * 60 * 60)
        }
    ],
     "skipped": [ // пропускается
        {
          "weekday": 6 // по субботам
        },
        {
          "weekday": 7 // по воскресеньям
        }
    ]
  },
// ...
]
```

Периодичность

В объекте периодичности атрибуты задаются в рамках своих обычных значений, кроме атрибута year - год. Атрибут year, задаётся в виде частоты, так как является не периодической характеристикой.

Пример:

```
{
  "second": 30, // 1 - 60
  "minute": 20, // 1 - 60
  "hour": 9, // 0 - 23
  "day": 5, // 1 - 31
  "weekday": 1 // 1 - 7
  "month": 3 // 1 - 12
  "year": 2,
  "duration": 30 //
}
```

Описание примера:

В примере определён временной интервал длительностью 30 секунд, который повторяется один раз в два года, пятого марта в 9 часов 20 минут 30 секунд и только если день выпадает на понедельник.

Пользовательские типы

Пользовательский тип - "type": 17, задает значение пользовательского типа на основе базового типа. Находится в директории meta, types + [название типа].type.json. Используется в случаях, когда необходимо применить маску на значения определенного атрибута в различных классах.

Допустимые базовые типы

При создании пользовательского типа доступны следующие базовые типы:

- Строка [0]
- Целое [6]
- Действительное [7]
- Дата/Время [9]
- Десятичное [8]

Пример пользовательского типа userPassport.type.json

```
{
  "name": "userPassport",
  "caption": "Номер паспорта",
  "type": 0,
  "mask": "99 99 999999",
  "mask_name": "passport",
  "size": 12,
  "decimals": null
}
```

Применение

Пользовательские типы подключаются путем указания типа атрибута “Пользовательский [17]” - "type": 17 и указанием наименования пользовательского типа в поле “refClass”.

Пользовательский тип в JSON

```
{
    "orderNumber": 20,
    "name": "passport",
    "caption": "Номер паспорта (Пользовательский тип [17])",
    "type": 17,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
```

(continues on next page)

(continued from previous page)


```
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "userPassport",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
```


Таким образом, при вводе значения для атрибута "Номер паспорта (Пользовательский тип [17])" будет применяться маска, заданая для типа "userPassport" по ссылке свойства "refClass".

Тип атрибута - указывает на тип поддерживаемых атрибутом данных, как например размер допустимых значений и другие.

Code	Name	Тип в БД	Description
0	String	String	Тип данных, значениями которого является произвольная последовательность (строка) символов алфавита. Каждая переменная такого типа (строковая переменная) может быть представлена фиксированным количеством символов.
1	Text	String	Хранит текстовые данные.
2	HTML	String	Форматированный текст, содержащий гипертекстовую разметку с возможностью редактирования с учетом возможных начертаний.
3	URL	String	Хранит ссылку, позволяет сохранить любую строку.
4	Image	String	Изображение, сохраняемое в файловом хранилище с предпросмотром в представлениях.
5	File	String	Файл, сохраняемый в файловом хранилище. В процессе реализации в регистри.
6	Integer	Int32	Целое число
7	Real	Double	Любое положительное число, отрицательное число или ноль.
9	Дата/время	Date	Дата в формате ISODate. Может быть отображена как дата, либо как дата-время.
8	Decimal	Double	Число, представленное в десятичной системе счисления. Алфавит этой системы счисления состоит из 10 цифр от нуля до 9, отсюда и название - десятичная.
10	Logical	Boolean	Принимает два возможных значения, называемых истиной (true) и ложью (false).
11	Password	String	Хеш пароля
12	Global identifier	String	Тип предназначенный для ключевого поля класса. Предполагает выставление атрибутов уникальности и автозаполнения.
13	Ссылка	String	Тип данных, хранящий в себе ссылку на объекты другого класса.
14	Коллекция	Array	Коллекция - тип данных, который хранит в себе ссылки на другие объекты. Каждая ссылка содержит значение идентификатора объекта, определенного в мете класса. Разделяются ссылки через запятую. Все значение из последовательности ссылок и запятых хранится строкой в базе.
15	Multiplicity	String	Храним набор дискретных значений из предопределенного списка выбора.
16	Структура	String	Тип данных, хранящий в себе ссылку на объект класса-структуры.
17	Пользовательский тип	String	Дает возможность определения пользовательских типов на основе примитивных типов.
18	User	String	Хранит имя пользователя, для настройки безопасности, в формате имя@local
100	Геоданные	Object	Особый тип данных, хранящий координаты с уникальными представлениями для создания и редактирования.
110	File collection	String	Тип атрибута для хранения комплекта файлов до 5 штук, с общим ограничением размера и возможностью задания допустимых расширений файлов
210	Расписание	Array	Тип данных, предназначеный для хранения данных календаря/расписания

Идентификаторы типов атрибутов:

```
module.exports = {
  STRING: 0,
  TEXT: 1,
  HTML: 2,
  URL: 3,
  IMAGE: 4,
  FILE: 5,
  INT: 6,
  REAL: 7,
  DECIMAL: 8,
  DATETIME: 9,
  BOOLEAN: 10,
  PASSWORD: 11,
  GUID: 12,
  REFERENCE: 13,
  COLLECTION: 14,
  SET: 15,
  STRUCT: 16,
  CUSTOM: 17,
  USER: 18,
  PERIOD: 60,
  GEO: 100,
  FILE_LIST: 110,
  SCHEDULE: 210
};
```

Autocomplete attributes

Autocompletion type - " autoassigned ": true - indicates that the value of this attribute should be filled automatically when creating a class. It is used mainly for attributes of the “Unique values” type ("unique": true) for integers and string attributes, as well as for attributes of the “Date-time” type.

Principle of formation:

1. For the attributes of the “Date-time” type, an attribute should have the value of the current time. It is used in the time tag of created objects and committed changes.
2. For integer attributes, if the value “Unique identifier” ("unique": true) is specified when creating the form, it is filled in with a random set of characters.
3. For strings, if the value “Unique identifier” ("unique": true) is specified, a random hex value should be generated - the size of the string length - 20 characters in the example below.

   ```
   var crypto = require('crypto');
   ID = crypto.randomBytes(20).toString('hex');
   ```

4. For the “global identifier” type, the configuration is the same as for the string.

NB: You need to make a check when saving. The field should be generated automatically for empty values or dates. For all others (integer, string), previously created values must be generated.

Example:

```
{
  "orderNumber": 50,
  "name": "auto",
  "caption": "auto",
  "type": 6,
  "size": null,
  "decimals": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": true,
  "autoassigned": true
}
```

Caching the value of computable attribute

Description

When applying the cached values, the attribute values are calculated during creating and editing an object. Previously calculated values are used for the samples.

If there are two computable attributes A and B referring to the collection C and the A has the cached value and B hasn't, then when editing the object, the collection C will be pulled twice. First time for the attribute B at the securedDataRepo level to verify the access. Second time for the attribute A when calculated into the dataRepo. In this case, when reading an object from the database, the cache of the attribute A simply does not make sense, since in any case the collection will be selected for the attribute B.

Caching semantics

Set the following property to cache semantics of the objects in meta class:

```
semanticCached: true
```

We recommend you not to use the eager loading for the attributes used in the cached semantics. Also, we recommend you not to use the dates, because they cannot be converted to the user's time zone, since they are cached when editing an object at the DBAL level.

Caching the value of computable attribute

Set the following property in the meta class to cache the value of the computable attribute:

```
cached: true
```

Besides, you can update caches of objects by reference when editing the main object.

Set the following property in the meta class:

```
cacheDependencies: ["refAttr1", "refAttr2.refAttr3", "refAttr2.collAttr4"]
```

When configuring the meta class, specify the reference and collections, the caches of the objects in which you need to update when editing an object of this class. Updates are done recursively. If the refAttr1 attribute is set to update caches in the class object, then the update will start. This setting is inherited in the heir classes.

Example:


```
{
    "orderNumber": 40,
    "name": "kolStatOps",
    "caption": "Количество стационарных ОПС",
    "type": 6,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": "count($raionObslu.oktmo_nasPunkta.svyaz.ops,&eq($gops, b), 1)",
    "cached": true
}
```


The value of this attribute obtained from the formula is cached. Additionally, to update the value when editing an object, you need to update the caches of objects by reference: to do this, specify cacheDependencies: in the meta class of each object by reference.

Example:


```
{
  "isStruct": false,
  "key": [
    "okato"
  ],
  "semantic": "name",
  "name": "naselenniyPunkt",
  "version": "",
  "caption": "Населенный пункт",
  "ancestor": null,
```


(continues on next page)

(continued from previous page)

```
  "container": "",
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": null,
  "cacheDependencies": ["supOktmo"],
  "properties": [
...
```

Default value

Default value is set when you need to output the value in the attribute field automatically when you open the object creation form. The default value is set by assigning the default value to the "defaultValue" property . The main use is for selection lists of acceptable values.

Example

```
{
    "orderNumber": 20,
    "name": "defaultValue",
    "caption": "Значение поля",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": "default",
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": {
     "type": "SIMPLE",
     "list": [
       {
         "key": "default",
         "value": "Значение, которое отображается по умолчанию при создании объекта"
       },
       {
         "key": "other",
```

(continues on next page)

(continued from previous page)

```
            "value": "Другое значение"
          }
        ],
        "matrix": [],
        "parameters": [],
        "hq": ""
      },
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
}
```

To implement ** automatic calculation of the default value**you can use the maxfunction:

```
"defaultValue": {max: ["className@namespace", "attr", {"filterAttr": "filterValue"}]}
```

Use the get operation to implement default value for the “Reference” attribute type in the following ways:

```
get(className) // возвращаем id случайно выбранного объекта класса
get(className, id) // проверяем наличие объекта в БД, если объект есть, возвращаем его id
get(className, attr1, val1, attr2, val2, ...) // возвращаем id первого объекта удовлетворяющего критериям
↪поиска: attr1=val1 и attr2=val2 и т.д.
```

Computable attribute (without caching)

Description

Computable attributes (formula) - are used to instantly generate a string expression as a result according to a preset algorithm when accessing a class object through the API. For example, when you open the object.

At the meta class level, the computable attributes store the algorithm for generating a string expression in the formula property.

For example you can access all unique values from the nameattribute, which values are stored in the ownOrg collection. All the unique values, divided by “,” are united in one string expression.

```
"formula": {
  "merge": [
    "$ownOrg",
    "name",
    null,
    1,
    ", "
  ]
}
```

For example, If you have a collection with non-unique values (for example, “ownOrg1”, “ownOrg2” and again “ownOrg1”) and you need to obtain only the unique values of the collection “ownOrg1 and ownOrg2”, then the above described formula for the computable attribute using the merge operation will be useful.

Depending on the function, you can refer to the necessary attribute to get the value through the attributes of the “Reference” and “Collection” types.

When saving changes and closing the form of an object, the result is not saved in the attribute if caching is not configured.

If the meta class has a few computable attributes, then the order of the calculation is set in the orderNumber property. You can use the results of calculations for a given order - orderNumber in the subsequent computable attributes.

You can specify computable attributes in the semantics of a class or an attribute.

If you set Null in the formula property, the attribute will not be computable and caching should not be applied to such an attribute.

Formula record model

Each formula begins with a description of the object and the function in the JSON format .

```
"formula": {
  "function1": [
    {
      "function2": [
        "operand3"
      ]
    },
    "operand1",
    "operand2"
  ]
}
```

In the object, be sure to specify: ref: a suitable function <supported-functions> with the required number of operands to get the result in function1.

The object contains the full description of the algorithm, that controls the calculations except the functions stored in the depended computable attributes.

An array in a function stores the order of the operands that are passed to the function.

The functions operands could be:

- String values, storing the constant

```
"formula": {
  "function1": [
    "string"
  ]
}
```

- String values that store: doc: variables </3_development/metadata_structure/meta_variables>.

```
"formula": {
  "function1": [
    "$attr1"
  ]
}
```

- Numerical values

```
"formula": {
  "function1": [
    3.14
  ]
}
```

- Empty values

```
"formula": {
  "function1": [
    null
  ]
}
```

- Objects

```
"formula": {
  "function1": [
    {
      "function2": [
        "operand1"
      ]
    }
  ]
}
```

Example of applying the formula:

```
{
    "orderNumber": 5,
    "name": "addressString",
    "caption": "",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": {
      "concat": [
        {
          "if": [
            "$zipCode",
            {
              "concat": [
```

(continues on next page)

(continued from previous page)

```
            "$zipCode"
          ]
        },
        ""
      ]
    },
    " ",
    {
      "if": [
        "$subjectFederation",
        "$subjectFederation",
        ""
      ]
    },
    {
      "if": [
        "$federationBorough",
        {
          "concat": [
            ", ",
            "$federationBorough"
          ]
        },
        ""
      ]
    },
    {
      "if": [
        {
          "and": [
            {
              "ne": [
                "$subjectFederation",
                "Санкт-Петербург г"
              ]
            },
            {
              "ne": [
                "$subjectFederation",
                "Москва г"
              ]
            }
          ]
        },
        {
          "concat": [
            ", ",
            "$town"
          ]
        },
        ""
      ]
    },
    {
      "if": [
        "$street",
```

(continues on next page)

(continued from previous page)

```
          {
            "concat": [
              ", ",
              "$street"
            ]
          },
          ""
        ]
      },
      {
        "if": [
          "$houseNumber",
          {
            "concat": [
              ", Дом ",
              "$houseNumber"
            ]
          },
          ""
        ]
      },
      {
        "if": [
          "$flatNumber",
          {
            "concat": [
              ", Квартира (офис) ",
              "$flatNumber"
            ]
          },
          ""
        ]
      }
    ]
  }
}
```

Result: the output of the address with spaces and commas between the values of the attributes.

Supported functions

eq - equal

ne - not equal

lt - less

gt - greater

lte - less or equal

gte - greater or equal

and - and

or - or

not - not

add - arithmetic addition

sub - arithmetical subtraction

mul - arithmetical multiplication

div - arithmetical division

nempty - not empty

empty - empty

pad - additional symbols to the desired string length

next - :ref:‘retrieves the new value of the sequence <autoassigned-attribute> ‘

merge - concatenation of attributes in the collection

size - available attribute values - string and collection. For strings - returns the length, for collections - the number of elements

element - derive an arbitrary element from an array, indexation is from 0 ([array of values], [element index: 0 is the first element, last is the last element])

dateAdd - an addition to the data (in the notation momentjs - [Date], [add interval (number)], [unit (line [d, m, y, h, min, s, ms)]

dateDiff - difference between dates (in the notation momentjs - [unitism], [Date1], [Date2])

now - current date and time

concat - string concatenation

substring - получение подстроки ([Строка], [с какого символа], [сколько символов])

obj - forming an object: odd arguments - property names, even arguments - values

aggregation:

max, min, avg, sum, count

All aggregation operations take the following arguments:

either

[$Имя атрибута коллекции], [Имя агрегируемого атрибута], [функция фильтрации элементов коллекции]

either

[Имя класса], [Имя агрегируемого атрибута], [Объект фильтра сформированный функцией obj ↪соответствующий нотации фильтров mongodb]

1 - indicates the uniqueness of the object, so it allows to count only unique objects for aggregation operations

\n - line break

Example:

```
"formula": {
        "merge": [
          "$atr1",
          "atr2.name",
          null,
```

(continues on next page)

(continued from previous page)

```
        1,
        "\n"
      ]
    },
```

Auto-assignment and getting the value of an attribute in a calculated expression

1. Set the autoassigned: true, so that calculated expression is not executed when opening the create form. The expressions will be calculated before saving. This is relevant when using the next operation in calculations, since it is not always necessary to derive the next value each time opening the creation form.
2. The default values are calculated before the object is written to the DB (so there is nothing yet in simple auto-assigned attributes at the stage of their calculation).
3. The next($id) operation (if the $id has a value) will always return 1, since for each object a separate sequence will be created, from which only the first value will be selected.

Attribute indexation

Attribute indexation is required to accelerate the search. Indexation is set manually by assigning the true value for the "indexed"property, that is:

```
"indexed": true
```

Except for the following attribute types:

1. Indexation is set automatically (regardless of what is specified in the indexation field):
   - for the key field;
   - for the objects with field "compositeIndexes": true in the main part of the meta class.
2. When importing a meta, all attribute objects of the “Reference” type are indexed.
3. Objects of attributes of the “Collection” type are not indexed, since collections with a back reference are not stored in the object, there is no need for indexing.

```
NOTE. Запрещается индексация атрибутов типа "Текст" [1] и "HTML" [2].
  Связано с ограничением MongoDB на размер индексируемого значения,
  т.к. размер значения атрибутов данных типов может превышать допустимый размер.
```

Collection

Collection - is a data type that allows lists of other objects to be displayed in one object. The data of the object can be objects of any class including the initial.

References are separated by commas. All values from a sequence of references and commas are stored as a string in the database.

Ways to define collections:

in terms of the used fields of the attribute part of the class meta

1. one-to-many is the classic relationship of a child object to a parent object. Indicates the presence of a container and a nested object with a reference to the container. In the container, you must specify the collection, and for the collection, specify the reference attribute of the nested object by which the link is formed. See Back References
2. many-to-many is determined through a collection without references and a class of nested elements — connections are created in the collection and stored as separate entities in the DB. See Collections
3. back collection is similar to the one-to-many connection but in the opposite direction - connection from the reference object. Set the connection using the backColl property. See Back collection

Collection attribute in JSON format

Example


```
{
    "orderNumber": 50,
    "name": "table",
    "caption": "Таблица",
    "type": 14,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "collRefCatalog@develop-and-test",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
}
```


NB: If the collection refers to a class that contains many descendants, then when the collection is populated, it will be possible to create objects of both parent and child classes.

Collections together with the object are loaded according to the semantics specified in the meta class of a collection or reference attribute.

Back reference in the context of collections

The back reference in the context of collection is formed as follows:

- create a regular collection specifying the reference attribute
- The reference class must have a reference attribute that references the source class and has the unique property set to false. The attribute-reference value is assigned immediately when creating a link to the collection, without the need to save the form
- specify the "backRef" property in the initial class of the collection. In this property, write down the code of the reference attribute from the reference class

Back reference attribute in JSON format

Example


```
{
    "orderNumber": 30,
    "name": "coll",
    "caption": "Коллекция с обратной ссылкой",
    "type": 14,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": null,
    "itemsClass": "ref_backcoll_ref@develop-and-test",
    "backRef": "ref_backcoll_ref",
    "backColl": "",
    "binding": "",
    "semantic": "backcoll_data",
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": true,
    "formula": null
}
```


Display modes of the “Collection” type attribute on the form:

Display modes are set in the view meta. They can be defined using the "mode" property, or set by a template in the "options" property.

- mode: 4 - “Tag Cloud” stores the values of one or several objects by reference in the form of tags, the name of which is determined by the semantics of the object by reference.

- mode: 3 - “Table” stores the values of one or several objects by reference in a table, the columns of which are predefined for the view form.

Example

```
{
    "caption": "Таблица",
    "type": 3,
    "property": "table",
    "size": 2,
    "maskName": null,
    "mask": null,
    "mode": 3,
    "fields": [],
    "columns": [],
    ...
},
...
```

- “Comment” - is set in the same way as the “Table” display mode, but with the template specified in the "options" property. It is a field that contains data that was predefined in the "columns" property for an object by reference. It is intended to discuss information on an object at a certain stage of a workflow.

Example

```
{
    "caption": "Коментарий",
    "type": 3,
    "property": "coment",
    "size": 2,
    "maskName": null,
    "mask": null,
    "mode": 3,
    "fields": [],
    "columns": [
      {
        "sorted": true,
        "caption": "Дата",
        "type": 120,
        "property": "date",
        ...
      },
      {
        "sorted": true,
        "caption": "Подтверждение (Обоснование)",
        "type": 7,
        "property": "descript",
        ...
      },
      {
        "caption": "Ведущий",
        "type": 2,
        "property": "owner",
```

(continues on next page)

(continued from previous page)

```
      ...
    }
  ],
  "actions": null,
  "commands": [
    {
      "id": "CREATE",
      "caption": "Создать",
      "visibilityCondition": null,
      "enableCondition": null,
      "needSelectedItem": false,
      "signBefore": false,
      "signAfter": false,
      "isBulk": false
    },
    {
      "id": "EDIT",
      "caption": "Править",
      "visibilityCondition": null,
      "enableCondition": null,
      "needSelectedItem": true,
      "signBefore": false,
      "signAfter": false,
      "isBulk": false
    }
  ],
  "orderNumber": 80,
  ...
  "tags": null,
  "options": {
    "template": "comments",
    "comments": {
      "textProperty": "descript",
      "userProperty": "owner",
      "parentProperty": "answlink",
      "photoProperty": "owner_ref.foto.link",
      "dateProperty": "date"
    }
  }
}
```

Back collection

The example of the collection above is converted for the back reference as follows:

```
{
    "orderNumber": 30,
    "name": "backcoll",
    "caption": "Обратная коллекции",
    "type": 14,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
```

(continues on next page)

(continued from previous page)

```
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "coll_backcoll_coll",
    "backRef": "",
    "backColl": "coll",
    "binding": "",
    "semantic": "backcoll_data",
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": true,
    "formula": null
}
```

Pay attention to the indication of the additional value in the property "backColl"- the name of the attribute from the class in the collection (from the example it is coll)

Thus, a many-to-many relationship is implemented without an intermediate class. Not only can the "backcoll" attribute with the “Collection” type contain multiple references, but objects on references can also contain multiple references to objects of the source class in their “coll” collection.

Attention:

- "type": 14 - the attribute type “Collection”
- "backColl" - the name of the collection type reference attribute referencing the original collection class
- "itemsClass" - the name of the class, the objects of which can store their identifiers in the collection and, thus, form a link to the object by identifier
- "backRef" - reference-attribute from the reference class specified in "itemsClass"
- When specifying a parent class, it is possible to create objects of the parent and child classes
- Collections with objects are loaded according to the semantics specified in the collection class or the collection attribute

Collection processing scheme and DB storage format

To save the collection, transfer the array of actions (the example below) in the corresponding attribute of the object:

```
"collection": [
  {"action": "put", "id": "1234"},
  {"action": "put", "id": "1235"},
  {"action": "put", "id": "1236"},
  {"action": "eject", "id": "1230"}
]
```

The order of the objects must correspond to the order of relevant actions. Available operations: put - add to the collection, eject - extract from the collection. The algorithm for creating and editing is the same. Actions on collections are performed after the container is created or saved.

The working principle of collections on the form of creation and editing is fundamentally different:

- On the creation form, interconnection with the server is required only to receive and display in the table the selected/created object of the collection
- On the editing form, it is possible to get a server response if necessary, and to change the select parameters upon request, depending on the actions performed in the collection.

Reference

Description

Reference is a data type that stores a simple value and is interpreted by the system as a reference to a key attribute of an object of another class. This object can be an object of any class, including the source one. When you specify a parent class, you can create objects of the parent and child classes. References together with the object are loaded according to the semantics specified in the meta of the reference class or the reference attribute.

Values in the attribute of the reference type are displayed in accordance with the semantics, specified in the reference class of this attribute.

The ability to replace an object by a back reference is determined by the nullable parameter that binds the reference attribute. When you replace an object, the reference will be lost, and the referenced object will be deleted when you try to change the reference from the back reference collection.

Types of links implemented by the “Reference” type:

Reference type in context of the attribute part of the meta class:

1. ‘‘one-to-many ‘‘- the classic connection of the heir object to the ancestor object. It is necessary to define the reference and indicate the class of the nested element - the connections are created when placed in the reference and stored as a separate entity in the database.
2. one-to-one - similar to the one-to-many connection, means the presence of a reference and a nested object with a bonded reference to the source object. In the reference, you must specify a bonded reference, and in the bonded reference you must indicate the reference attribute of the nested object by which the connection is formed. Be sure to specify the "unique ": true property in the reference attribute.

Reference in JSON format

Example

```
{
    "orderNumber": 20,
    "name": "ssylka",
    "caption": "Ссылка",
    "type": 13,
```

(continues on next page)

(continued from previous page)


```
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "collRefCatalog@develop-and-test",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
}
```


Display modes of the “Reference” type attribute on the form:

You can set the display mode in the meta view. Set the mode using the "mode" property or the "options" property to set it as a template.

- “mode”: 0 - displaying only the semantics of an object by reference
- “mode”: 1 - displaying the reference to the form of the object by reference
- “mode”: 3 - hierarchical object search
- “mode”: 4 - refining object search

Back reference

Back reference in the context of reference is obtained as follows:

- Create an attribute with type 13 and specify the reference class refClass and specify the property "backRef" (where the attribute code from the reference class is written).
- the reference class should contain the reference attribute, which refers to the initial class and has the "unique": true property.

Back reference in JSON format

Example


```
{
    "orderNumber": 30,
    "name": "backref",
    "caption": "Обратная ссылка",
    "type": 13,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "otorbrRef@develop-and-test",
    "itemsClass": "",
    "backRef": "ref",
    "backColl": "",
    "binding": "",
    "semantic": "data",
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": true,
    "formula": null
}
```


Attention:

- "type": 13 - attribute type “Reference”
- "refClass" - the name of the class whose objects can store their identifiers in the reference and, thus, form a relationship with an object by its identifier.
- "backRef" - specifies the name of the attribute that belongs to the class specified in the property "refClass". The attribute must have the type “Reference” and a reference to the source class.
- When specifying a parent class, it is possible to create objects of the parent and child classes.
- References with object are loaded according to the semantics specified in the meta of the reference class or reference attribute.

Example

```
Employee: {
   property: {
      aaa: {
         refClass: Post,
         backRef: bbb,
         ...
```

(continues on next page)

(continued from previous page)

```
        },
        ...
    }
}

Post: {
    property: {
        bbb: {
            refClass: Employee,
            ...
        },
        ...
    }
}
```

Conditions of sorting the valid values

Description

Conditions for selecting allowed values are conditions that allow you to limit the selection of objects by reference that are allowed for binding in this reference attribute.

Filter of sorting the valid values is used in the meta class for the attributes of the "Reference" and "Collection" types. The filter sets the conditions to limit the sample of objects. Conditions are imposed as a list of consecutive operations.

Available operations:

- EQUAL: 0, // равно =
- NOT_EQUAL: 1, // не равно <>
- EMPTY: 2, // пусто '' или null
- NOT_EMPTY: 3, // не пусто !'' или !null
- LIKE: 4, // похож
- LESS: 5, // меньше <
- MORE: 6, // больше >
- LESS_OR_EQUAL: 7, // меньше или равно <=
- MORE_OR_EQUAL: 8, // больше или равно >=
- IN: 9, // элемент входит в коллекцию/массив (IN)
- CONTAINS: 10 // содержит

```
module.exports = {
  AND: 0,
  OR: 1,
  NOT: 2,
  MIN: 3,
  MAX: 4,
  AVG: 5,
  SUM: 6,
  COUNT: 7
};
```

Description of operations

All operations can be divided into groups according to the properties in the condition:

The attribute is not set in the condition and the condition is the object

- nestedConditions does not contain following conditions
  - Aggregation operations AgregOpers
    1. MIN
    2. MAX
    3. AVG
    4. SUM
    5. COUNT
- nestedConditions contains following conditions
  - Logical operations of comparison of nested conditions BoolOpers
    1. OR
    2. NOT

The attribute is set in the condition and the condition is the object: operations of comparison of the attribute value in the condition with the value in the “value” field

1. EMPTY
2. NOT_EMPTY
3. CONTAINS
4. EQUAL
5. NOT_EQUAL
6. LESS
7. MORE
8. LESS_OR_EQUAL
9. MORE_OR_EQUAL
10. LIKE
11. IN

Condition in the form of an array

- Use the logic operation “AND” to compare the results of conditions (objects in array).

In the operation of the key-expression type, the key is the attribute name in the reference class or in the collection class. Adjacent conditions are combined by a logical “AND” operation (unless another operation is specified) and filters are added to the “selConditions” property.

Application of operations and other features

Use the “nestedConditions” to perform the attribute inquiry. A separate operation for each attribute. Do not specify nested reference attributes by a point in the “property” field.

To inquire attribute values that are not equal to zero, use the nemptyoperation and specify null in the “value” field.

The CONTAINS operation is applied to the following attribute types:

- string - the LIKE operation is applied to the data string
- collection
  - the IN operation is applied if the compared value is an array and contains at least one element
  - transition to nested conditions nestedConditionsoccurs if the compared value is not an array or does not contain at least one element in the array

JSON

```
{
  "selConditions": [
    {
      "property": "region",
      "operation": 10,
      "value": "Хабаровский край",
      "nestedConditions": [
        {
          "property": "town",
          "operation": 0,
          "value": "г Хабаровск",
          "nestedConditions": []
        }
      ]
    }
  ]
}
```

```
{
  "selConditions": [
    {

          "property": "town",
          "operation": 3,
          "value": null,
          "nestedConditions": []
        }
      ]
}
```

Field description

Field	Name	Acceptable values	Description
"property"	Attribute	String only the latin characters with no spaces	The attribute of the reference class by which the values are filtered
"operation"	Operation	Operation code (see above)	The operation according to which the filtration is performed
"value"	Value	Depends on the type of operation	The second value for binary filtering operations
"nestedConditions"	Nested selection conditions	Object, the structure is similar to the structure of the selection conditions object itself.	

Example

Attention

Field “selection_provider”. See for more detailes Selection list of valid values.

- “type”: “SIMPLE” - simple type,
- “list”: [] - an array of acceptable values.


```
{
  "orderNumber": 80,
  "name": "type",
  "caption": "Тип организации",
  "type": 0,
  "size": null,
  "decimals": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "selConditions": [],
  "selSorting": [],
  "selectionProvider": {
    "type": "SIMPLE",
    "list": [
      {
        "key": "customer",
        "value": "Заказчик"
      },
      {
        "key": "executor",
        "value": "Исполнитель"
      }
```


(continues on next page)

(continued from previous page)


```
    ],
    "matrix": [],
    "parameters": [],
    "hq": ""
  },
  "indexSearch": false,
  "eagerLoading": false
}
```


Example

In the reference attribute, it is necessary to show only those objects that have the “selConditions” attribute set in the reference class and in the property field of which this attribute specifies the field of the related class whose value in the “value” field corresponds to the “operation” condition.

The aim is to show in the “Organization” attribute only those organizations (“refClass”: “organization”) in which the type field (“property”: “type”) is equal (“operation”: 0) to the customer value (“value”: “customer”).

All conditons in the "selConditions" are united by “AND” condition.


```
{
  "orderNumber": 120,
  "name": "customer",
  "caption": "Заказчик",
  "type": 13,
  "size": null,
  "decimals": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "defaultValue": null,
  "refClass": "organization",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "selConditions": [
    {
      "property": "type",
      "operation": 0,
      "value": "customer",
      "nestedConditions": []
    }
  ],
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false
},
{
  "orderNumber": 130,
  "name": "executor",
  "caption": "Исполнитель",
```


(continues on next page)

(continued from previous page)


```
  "type": 13,
  "size": null,
  "decimals": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "defaultValue": null,
  "refClass": "organization",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "selConditions": [
    {
      "property": "type",
      "operation": 0,
      "value": "executor",
      "nestedConditions": []
    }
  ],
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false
}
```


Conditions of sorting the valid values for “Data” attribute type

The core implements the context attribute - $$ now, which returns the current date. $$ now is available everywhere if you specify the conditions.

For more details see variables.

Example

Condition: to display the objects with the attribute value [dataStart] less than the current data:


```
{
      "property": "dateStart",
      "operation": 5,
      "value": [
        "$$now"
      ],
      "nestedConditions": []
 }
```

Selection list of valid values

Selection list of valid values specifies a selection list of valid values for an attribute input field and is located in the attribute part of the class meta "selectionProvider". The list is formed in the form of an array of

objects of the “key-value” type and is a selection list of values for the attribute with the “String”, “Real”, “Integer”, “Decimal”, and “Text” types.

You have three types of a selection list. Set the type in the ("type") field using one of the following keys:

- "SIMPLE" - simple selection list;
- "MATRIX" - matrix selection list.

Description of the structure fields

Structure of the selection list object

```
"selectionProvider": {
  "type": "SIMPLE",
  "list": [...],
  "matrix": [],
  "parameters": [],
  "hq": ""
},
```

Field	Name	Acceptable values	Description
"type"	Type	"SIMPLE", "MATRIX", "HQL"	Selection list type
"list"	Simple type	An array of objects of the “key-value” type	Selection list of the “SIMPLE” type is stored here.
"matrix"	Matrix	An array of vectors. Each of vectors consists of: name, selection conditions and key-values.	Selection list of the “MATRIX” type.
"parameters"	Parameters of request	An array of objects of the “key-value” type	Parameters of request, to be realized
"hq"	Request	String of request in accordance with the handler format not used in the current version	String of request, to be realized

The "list" field - an array of objects of the following structure:

```
"list": [
  {
    "key": "2001-03-23 09:00:00.000Z",
    "value": "Затопление орбитальной станции «Мир» (23 марта 2001 г. 09:00 мск)"
  },
  {
    "key": "1957-10-04 19:28:00.000Z",
    "value": "Запуск первого в мире искусственного спутника (4 октября 1957 г. в 19:28 гринвич)"
  },
  {
    "key": "1970-04-17 12:07:00.000Z",
    "value": "Завершение полёта «Аполлон-13» (17 апреля 1970 г. 12:07 Хьюстон)"
  }
],
```

Field	Name	Acceptable values	Description
"key"	Key	Any value corresponding to the attribute type of the selection list	When saving an object the key value is written in the DB
"value"	Value	Any string, but there may be problems if there are control sequences	The value of this field is displayed in the user interface

The "matrix" field is an array of objects of the following structure:

```
"matrix": [
  {
    "comment": "Оба отрицательные",
    "conditions": [
      {
        "property": "matrix_base_1",
        "operation": 5,
        "value": "0",
        "nestedConditions": []
      },
      {
        "property": "matrix_base_2",
        "operation": 5,
        "value": "0",
        "nestedConditions": []
      }
    ],
    "result": [
      {
        "key": "Оба отрицательные",
        "value": "Оба отрицательные"
      }
    ]
  },
  {
    "comment": "Оба неотрицательные",
    "conditions": [
      {
        "property": "matrix_base_1",
        "operation": 8,
        "value": "0",
        "nestedConditions": []
      },
      {
        "property": "matrix_base_2",
        "operation": 8,
        "value": "0",
        "nestedConditions": []
      }
    ],
    "result": [
      {
        "key": "Оба неотрицательные",
        "value": "Оба неотрицательные"
      }
    ]
  },
```

(continues on next page)

(continued from previous page)


```
    {
      "comment": "Первое неотрицательное второе отрицательное",
      "conditions": [
        {
          "property": "matrix_base_1",
          "operation": 8,
          "value": "0",
          "nestedConditions": []
        },
        {
          "property": "matrix_base_2",
          "operation": 5,
          "value": "0",
          "nestedConditions": []
        }
      ],
      "result": [
        {
          "key": "Первое неотрицательное второе отрицательное",
          "value": "Первое неотрицательное второе отрицательное"
        }
      ]
    },
    {
      "comment": "Первое отрицательное, второе неотрицательное",
      "conditions": [
        {
          "property": "matrix_base_1",
          "operation": 5,
          "value": "0",
          "nestedConditions": []
        },
        {
          "property": "matrix_base_2",
          "operation": 8,
          "value": "0",
          "nestedConditions": []
        }
      ],
      "result": [
        {
          "key": "Первое отрицательное, второе неотрицательное",
          "value": "Первое отрицательное, второе неотрицательное"
        }
      ]
    }
  ],
  "parameters": [],
  "hq": ""
},
```


Each object of the "MATRIX" array contains the following mandatory fields:

Field	Name	Acceptable values	Description
"comment"	Comment	Any string	Comment to the vector
"conditions"	Conditions	Array of objects	Defines the conditions under which the list of objects described in "result" of this vector is displayed
"result"	Results	Array of objects, similar to the structure of the "list" field	Sets the selection list that is displayed when the conditions are set correctly in the "conditions" field

The "conditions" field of the "MATRIX" array

Field	Name	Acceptable values	Description
"property"	Attribute	String only the latin characters with no spaces	Class attribute, the field value of which is checked for compliance with the given condition of the given vector
"operation"	Operation	Operation code	The operation according to which the determination is made
		0 - equal to (AND)	
		1 - not equal to (OR)	
		2 - empty (NOT)	
		3 - not empty (MIN FROM)	
		4 - (MAX FROM)	
		5 - < ()	
		6 - >	
		7 - <=	
		8 - >=	
		9 - IN /Similar/	
		10 - contains	
"value"	Value	Depends on the type of operation	Second value for binary operations
"nestedConditions"	Nested selection conditions	Object, the structure is similar to the structure of the selection conditions object itself.	

NB: The operation code corresponds to different operation values, depending on whether the attribute is selected or not. If the "property" field is null, then a boolean condition is encoded, according to which the nested selection conditions are combined. (Indicated in parentheses in the table above)

Description

Selection list of the “SIMPLE” type

This selection list allows you to create a preset of field values hardwired into the application, thereby limiting the user’s choice.

For the field, it is mandatory to set the “Drop-down list [5]” view type.

It implies the ability to save data in the database in a type different from the type of data displayed to the user.

For example: If you set the date-time value selection list items in ISODate as the key fields, and the event description as the value field, then we will give the user the opportunity to select the event, but work with ISODate data inside the application.

NB: If an attribute with a selection list is allowed to have an empty value: ‘‘ “nullable”: true‘‘, then an empty default value is added to the selection list!


```
{
  "orderNumber": 50,
  "name": "sp_date",
  "caption": "Сохраняем ключ дата-время",
  "type": 9,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "semantic": null,
  "selConditions": [],
  "selSorting": [],
  "selectionProvider": {
    "type": "SIMPLE",
    "list": [
      {
        "key": "2001-03-23T09:00:00.000Z",
        "value": "Затопление орбитальной станции «Мир» (23 марта 2001 г. 09:00 мск)"
      },
      {
        "key": "1957-10-04T19:28:00.000Z",
        "value": "Запуск первого в мире искусственного спутника (4 октября 1957 г. в 19:28 гринвич)"
      },
      {
        "key": "1970-04-17T12:07:00.000Z",
        "value": "Завершение полёта «Аполлон-13» (17 апреля 1970 г. 12:07 Хьюстон)"
      }
    ],
    "matrix": [],
    "parameters": [],
    "hq": ""
  },
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
}
```


Principle of formation:

You need to:

1. choose the most appropriate attribute type,

2. choose the identifiers of the type ("key") in order to operate with values in the DB as efficiently as possible if automated processing is required,
3. set the phrase to each key, that will be displayed in the "value"field,
4. make sure to set the “Drop-down list [5]” view type.

Selection list of the “MATRIX” type

All that falls under the conditions is the resulting selection list in matrix. If there is no conditions - the system will always apply the selection list. For the predictability of the application, respect two conditions:

1. The vectors should not overlap each other.
2. Array of values of the initial attribute, as the matrix base (an array of combinations of the initial (reference) attributes values) must be completely closed by the described vectors.

   The system takes the value of the reference field (s) and consistently applies the conditions of the vectors to this field. Each vector is the set of conditions and its own selection list. As soon as the system reaches the vector with satisfied conditions, it takes its selection list and defines the output in the UI. It is assumed that the system will find the corresponding vector for any value of the reference field.

Example 1: Matrix of two integer values

JSON of the class:


```
{
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "selection_provider_matrix_dc",
  "version": "",
  "caption": "\"MATRIX\" от двух оснований",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": [],
  "properties": [
    {
      "orderNumber": 10,
      "name": "id",
      "caption": "Идентификатор",
      "type": 12,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": false,
      "readonly": false,
      "indexed": false,
```


(continues on next page)

(continued from previous page)


```
    "unique": true,
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "matrix_base_1",
    "caption": "Первое целое основание матрицы",
    "type": 6,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "matrix_base_2",
    "caption": "Второе целое основание матрицы",
    "type": 6,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
```


(continues on next page)

(continued from previous page)


```
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 40,
    "name": "selection_provider_matrix",
    "caption": "Список выбора типа \"MATRIX\"",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": {
      "type": "MATRIX",
      "list": [],
      "matrix": [
        {
          "comment": "Оба отрицательные",
          "conditions": [
            {
              "property": "matrix_base_1",
              "operation": 5,
              "value": "0",
              "nestedConditions": []
            },
```


(continues on next page)

(continued from previous page)


```
          {
            "property": "matrix_base_2",
            "operation": 5,
            "value": "0",
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "Оба отрицательные",
            "value": "Оба отрицательные"
          }
        ]
      },
      {
        "comment": "Оба неотрицательные",
        "conditions": [
          {
            "property": "matrix_base_1",
            "operation": 8,
            "value": "0",
            "nestedConditions": []
          },
          {
            "property": "matrix_base_2",
            "operation": 8,
            "value": "0",
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "Оба неотрицательные",
            "value": "Оба неотрицательные"
          }
        ]
      },
      {
        "comment": "Первое неотрицательное второе отрицательное",
        "conditions": [
          {
            "property": "matrix_base_1",
            "operation": 8,
            "value": "0",
            "nestedConditions": []
          },
          {
            "property": "matrix_base_2",
            "operation": 5,
            "value": "0",
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "Первое неотрицательное второе отрицательное",
```

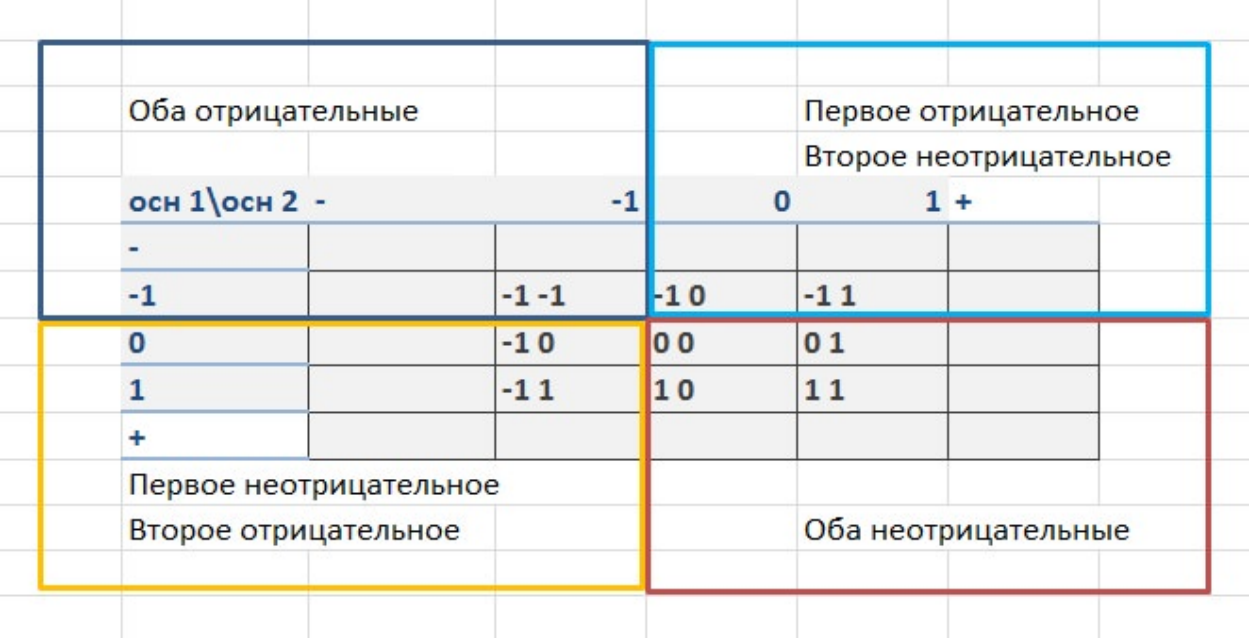

(continues on next page)

(continued from previous page)


```
            "value": "Первое неотрицательное второе отрицательное"
          }
        ]
      },
      {
        "comment": "Первое отрицательное, второе неотрицательное",
        "conditions": [
          {
            "property": "matrix_base_1",
            "operation": 5,
            "value": "0",
            "nestedConditions": []
          },
          {
            "property": "matrix_base_2",
            "operation": 8,
            "value": "0",
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "Первое отрицательное, второе неотрицательное",
            "value": "Первое отрицательное, второе неотрицательное"
          }
        ]
      }
    ],
    "parameters": [],
    "hq": ""
  },
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
 }
]
}
```


Procedure of development

Divide all possible combinations of attribute pairs - "matrix_base_1" and "matrix_base_2" into 4 vectors. Each field can be either negative or non-negative. See the scheme below:



Choose vectors and their conditions:

1. Both negative: (matrix_base_1 < 0) && (matrix_base_2 < 0)
2. Both non-negative: (matrix_base_1 >= 0) && (matrix_base_2 >= 0)
3. First non-negative, second negative: (matrix_base_1 >= 0) && (matrix_base_2 < 0)
4. First negative, second non-negative: (matrix_base_1 < 0) && (matrix_base_2 >= 0)
5. If in the 3 and 4 conditions the equality to zero is not correctly set, there will be drop-down elements and overlapping of vectors as a result.

In the example above, for each vector the selection list is limited to one item, but there may be more.

Example 2: matrix of free real value with compound conditions


```
{
  "isStruct": false,
  "metaVersion": "2.0.7",
  "key": [
    "id"
  ],
  "semantic": "",
  "name": "selection_provider_matrix_real",
  "version": "",
  "caption": "\"MATRIX\" с векторами \u003c, \u003e, \u003c\u003d, \u003e\u003d, \u003d от
↪действительного",
  "ancestor": null,
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "history": 0,
  "journaling": false,
  "compositeIndexes": null,
  "properties": [
```


(continues on next page)

(continued from previous page)


```
  {
    "orderNumber": 10,
    "name": "id",
    "caption": "Идентификатор",
    "type": 12,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": false,
    "readonly": false,
    "indexed": false,
    "unique": true,
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 20,
    "name": "matrix_base",
    "caption": "Действительное основание для списка выбора матричного типа",
    "type": 7,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
```


(continues on next page)

(continued from previous page)


```
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "selection_provider_matrix",
    "caption": "Список выбора со сложными условиями",
    "type": 6,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": false,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": {
      "type": "MATRIX",
      "list": [],
      "matrix": [
        {
          "comment": "matrix_base \u003c 3",
          "conditions": [
            {
              "property": "matrix_base",
              "operation": 5,
              "value": [
                "3"
              ],
              "nestedConditions": []
            }
          ],
          "result": [
            {
              "key": "1",
              "value": "Сохраним 1 при основании меньше 3"
            },
            {
              "key": "2",
              "value": "Сохраним 2 при основании меньше 3"
            }
          ]
        },
        {
          "comment": "matrix_base \u003d 3",
          "conditions": [
```


(continues on next page)

(continued from previous page)


```
        {
          "property": "matrix_base",
          "operation": 0,
          "value": [
            "3"
          ],
          "nestedConditions": []
        }
      ],
      "result": [
        {
          "key": "3",
          "value": "Сохраним 3 при основании 3"
        }
      ]
    },
    {
      "comment": "matrix_base \u003e 3 и matrix_base \u003c\u003d 15",
      "conditions": [
        {
          "property": "matrix_base",
          "operation": 6,
          "value": [
            "3"
          ],
          "nestedConditions": []
        },
        {
          "property": "matrix_base",
          "operation": 7,
          "value": [
            "15"
          ],
          "nestedConditions": []
        }
      ],
      "result": [
        {
          "key": "5",
          "value": "Сохраним 5 при основании \u003e 3 и \u003c\u003d 15"
        },
        {
          "key": "10",
          "value": "Сохраним 10 при основании \u003e 3 и \u003c\u003d 15"
        },
        {
          "key": "15",
          "value": "Сохраним 15 при основании \u003e 3 и \u003c\u003d 15"
        }
      ]
    },
    {
      "comment": "matrix_base \u003e\u003d16",
      "conditions": [
        {
          "property": "matrix_base",
```


(continues on next page)

(continued from previous page)


```
            "operation": 8,
            "value": [
              "16"
            ],
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "50",
            "value": "Сохраним 50 при основании \u003e\u003d 16"
          },
          {
            "key": "100",
            "value": "Сохраним 100 при основании \u003e\u003d16"
          },
          {
            "key": "1000",
            "value": "Сохраним 1000 при основании \u003e\u003d16"
          },
          {
            "key": "5000",
            "value": "Сохраним 5000 при основании \u003e\u003d16"
          }
        ]
      },
      {
        "comment": "matrix_base \u003e 15 и matrix_base \u003c 16",
        "conditions": [
          {
            "property": "matrix_base",
            "operation": 6,
            "value": [
              "15"
            ],
            "nestedConditions": []
          },
          {
            "property": "matrix_base",
            "operation": 5,
            "value": [
              "16"
            ],
            "nestedConditions": []
          }
        ],
        "result": [
          {
            "key": "0",
            "value": "Сохраним 0, если основание где-то между 15 и 16"
          }
        ]
      }
    ],
    "parameters": [],
    "hq": ""
```


(continues on next page)

(continued from previous page)


```
    },
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  }
 ]
}
```


Vectors and their conditions:

1. matrix_base < 3
2. matrix_base = 3
3. (matrix_base > 3) && (matrix_base <= 15)
4. matrix_base >= 16
5. (matrix_base > 15) && (matrix_base < 16)

Sorting a sample of valid values

Description

Sorting a sample of valid values is a filter that specifies how to sort objects. It is used for attributes of the "Reference" type. Set it in the "selSorting" field when creating the meta class.

Available types of sorting:

- Sorting in ascending order
- Sorting in descending order

JSON


```
{
    "orderNumber": 20,
    "name": "ref",
    "caption": "Ссылка",
    "type": 13,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": false,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "selSortingCatalog@develop-and-test",
```


(continues on next page)

(continued from previous page)


```
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "semantic": null,
  "selConditions": [],
  "selSorting": [
    {
      "property": "string",
      "mode": 1
    }
  ],
```


Field description

Field	Name	Acceptable values	Description
"property"	Attribute	String only the latin characters with no spaces	The attribute to sort by.
"mode"	Sort order	0 - ascending	Sort order
		1 - descending	

Collection base

Collection base - an attribute in the object, by the value of which the search of objects in the collection is performed. The search is based on the comparison of collection base with the back reference attribute.

Purpose of use

The base of the collection can be used to create dynamic collections. It means that the objects in the collection can be loaded depending on the entered data or calculated values by the formula, unlike regular back reference with search only by key attributes.

Example

For example in the develop-and-test project there are two classes:

- “ searchRefs“ has the string attribute “ code_binding ” added as a back reference for the binding.


```
{
  "orderNumber": 30,
  "name": "code_binding",
  "caption": "Код для binding",
  "type": 0,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
```


(continues on next page)

(continued from previous page)


```
  "readonly": false,
  "indexed": true,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": null,
  "backRef": null,
  "backColl": "",
  "binding": "",
  "semantic": "",
"selConditions": null,
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
}
```


- In the backref_searchRefs class the backref_searchRefs_binding and backref_searchRefs_text attributes are used. backref_searchRefs_binding is a dynamic collection, and the backref_searchRefs_text is used as a filter for the backref_searchRefs_binding collection. In the backref_searchRefs_binding collection the selection is occured by the equality of backref_searchRefs_text values from the backref_searchRefs class and code_binding values from the searchRefs class. When adding objects to the collection manually, the code_binding attribute is automatically filled.


```
{
  "orderNumber": 25,
  "name": "backref_searchRefs_binding",
  "caption": "Обратная ссылка (с подключенным binding) на класс searchRefs",
  "type": 14,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "searchRefs",
  "backRef": "code_binding",
  "backColl": "",
  "binding": "backref_searchRefs_text",
  "semantic": null,
"selConditions": null,
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
```


(continues on next page)

(continued from previous page)


```
},
{
  "orderNumber": 30,
  "name": "backref_searchRefs_text",
  "caption": "Значение",
  "type": 0,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": true,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "semantic": null,
"selConditions": null,
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null
}
```


Eager loading

Prefetching of load objects is the process that allows you to specify in a request which related data, in an amount sufficient for the correct display of the semantics of the related object, must be loaded when executing the request.

Eager loading helps in displaying objects of the system in cases where the data is at a high level of nesting, for example, in the attributes of the “Reference” and “Collection” types.

In meta, it is used only in special cases mainly to save time resources and as an alternative to fine-tuning.

Example

You can set the eager loading for an attribute by using eagerLoading property with true or false values.

```
"properties": [
   {
     ...
     "eagerLoading": true,
     "formula": null
   }
```

Configuration

We recommend you to configure the eager loading using the deploy.json file of the project. It is appropriate for the attributes of the meta class and meta navigation. This way of configuration allows you to centrally define pre-selected attributes for many classes.

Example

```
"eagerLoading": {
        "node1": {
          "class1": {
             "list": ["attr1", "attr2.attr3"],
             "item": ["attr1", "attr2.attr3"]
           }
        }
      }
```

NB: If instead of "node1" put "*", then when you get from any navigation to this application object, you can use one class setting to export the object.

Configuration for export in lists and forms

The configuration of the eager loading for export in lists and forms coincides with the configuration in the deploy.json file, with only one exception. Instead of list and item, specify exportList and exportItem.

Example

```
"eagerLoading": {
  "class1@ns": {
    "exportList": ["attr1", "attr2.attr3"]
  },
  "class1@ns": {
    "exportItem": ["attr1", "attr2.attr3.attr4"]
  }
}
```

The depth of eager loading of load objects

You can specify the depth of the eager loading in the maxEagerDepth: 1 property in the deploy.json file of the project.

The maximum depth of the eager loading determines the maximum permissible level of nesting of an object.

Semantics

Semantics - is used to output a class object as one string of the class title.

The "semantic" field is used twice in the meta class:

1. in the general part of the meta class, where it forms a string view for this class;
2. in the class attribute meta, where it forms a string representation of the objects of the class referenced by the attribute, i.e. used for reference attributes.

Purpose of use

“Semantics” is used to adjust the display of attributes and attribute properties in the list. In attributes that display tabular data, semantics are used to limit the output of columns.

Examples of use in reference attributes

For example, there is a class, which has attributes: id, name, link, date. There is a second class - classTable, which has a reference attribute table on the class class. Without using the semantics in objects of the classTable class, in the table attribute only the values of identifiers of objects of the class will be displayed. Attributes used as identifiers are listed in the class meta class.

To display the values of the name and link attributes in the tableattribute, and not the values of identifiers, you need to write "semantic": "name|link". Depending on the type of attribute, the result will be different:

- if the table attribute is a reference, then the values of the name and link attributes separated by spaces will be filled in. Here you can use additional words and expressions using the || sign, for example " semantic ":" name|, |link" or "semantic":"The object has 2 attributes:|name|, |link";
- If the table attribute is a collection of objects of the classclass, then it will display the name and link columns.

Display format in semantics

- You can trim the output with:[].

```
 "name[0,50]|..."
Указываем позицию и количество выводимых букв из семантики атрибута. Из атрибута name␣
↪выводим 50 символов семантики (значение атрибута), начиная с первого.
```

- Dereferencing is available via ‘‘. ‘‘ i.e. access to the nested object.

```
 "semantic": "digitalTV|kachestvoCTB|analogTV.name|kachestvoAnal|period"
где ``analogTV`` - ссылочный атрибут класса, для которого задается семантика, а ``name`` -␣
↪атрибут класса по ссылке.
```

Display semantics on form

1. In the first-level lists (opened directly by the navigation node), only the value from the “caption” field of the navigation node is displayed as the title.
2. In the selection lists we display only the value from the “caption” field of the class of the list objects as a title.
3. In the edit form we display only the semantics of the object as a title.
4. In the creation form we display only the value from the “caption” field of the class as a title.

5. In the selection lists we display the following line in fine print “Selecting the value of the attribute <...> of the object <...>” above the title.
6. In the creation form, if an object is created in a collection or a reference, we display the following line in fine print “Creating an object in the collection/by reference <...> object <...>” above the title.

Attribute part of the meta class describes the properties field of the meta class. There are usually at least two attributes in a class — a key field and a data field. The attributes are contained in the ‘‘”properties”‘‘field of the main part of the meta class. Each attribute is an object of the following structure:

JSON


```
{
  "orderNumber": 20,
  "name": "integer_integer",
  "caption": "Редактор целых чисел [14]",
  "type": 6,
  "size": null,
  "decimals": 0,
  "allowedFileTypes": null,
  "maxFileCount": 0,
  "nullable": true,
  "readonly": false,
  "indexed": false,
  "unique": false,
  "autoassigned": false,
  "hint": null,
  "defaultValue": null,
  "refClass": "",
  "itemsClass": "",
  "backRef": "",
  "backColl": "",
  "binding": "",
  "semantic": null,
  "selConditions": [],
  "selSorting": [],
  "selectionProvider": null,
  "indexSearch": false,
  "eagerLoading": false,
  "formula": null,
  "cached": true
}
```

Field description

Code	Name	Acceptable values	Description
"orderNumber"	Serial number	Non-negative integer	Sets the position of the attribute regarding the other attributes of the same class.
"name"	System name	String only the latin characters with no spaces	The name of the attribute that the system will work with, which means it cannot be empty, it can contain only Latin characters, without spaces (set once when crating the attribute). You cannot change the name later.
"caption"	Logic name	String	Display attribute name in user interface.
"type"	Type	Integer - identifier (code) of the type	Attribute data type. См. Attribute types
"size"	Size	Positive integer	The maximum size of the attribute data, the valid values depend on the type of attribute.
"decimals"	Number of decimal places	Non-negative integer	The number of decimal places is specified only for the “Decimal [8]” type.
"allowedFileTypes"	Valid file extension	Array of strings	Allows to specify the valid file extensions that a user can upload to the “File collection [110]” attribute.
"maxFileCount"	Maximum number of files	Number from 1 to 5	Specifies the maximum number of files that a user can upload to the “File collection [110]” attribute.
"nullable"	Valid null value	Logical	Allows or denies an empty attribute value.
"readonly"	Read only	Logical	Allows or denies changing attribute value.
"indexed"	Indexing for search	Logical	Indicates whether to index the value of this attribute to speed up the search.
"unique"	Unique values	Logical	Only unique attribute values (Attention:you cannot create two class objects with the same values in the attribute of the unique type).
"autoassigned"	Auto assignment	Logical	Allows or denies autocompletion of the fields.
"hint"	Help text	String	Specifies the message that will appear in the user interface next to the attribute name.
"defaultValue"	Default value	Depends on attribute type	Specifies the value that will be filled in the create form of the attribute (when an object is created)
"refClass"	Reference attribute	String, only the latin characters with no spaces	Contains the value of the " name " field (System name) of the class to be used in the “Link [13]” type attribute.
"itemsClass"	Collection attribute	String only the latin characters with no spaces	Contains the value of the " name " field (System name) of the class, whose objects can bind to an attribute of the “Collection [14]” type
"backRef"	Back reference attribute	String only the latin characters with no spaces	Specifies the attribute of the “Link[13]” type from the class specified in the Collection Class property, which refers to the original class. It is necessary to filter and bind objects from the Class class of the collection by the value of the reference attribute.
"backColl"	Back collection	String only the latin	Specifies the attribute of the “Collection [14]” type from the class specified in the Collection Class property, which refers to the original

3.4.3 Мета навигации

Мета навигации - регулирует расположение элементов в навигационном блоке.

Мета навигации разделяется на мету узлов навигации и мету секции навигации.

Мета секций навигации

Мета секций навигации состоит из поля "name" + .section.json и находится в директории navigation.

Например: workflow.section.json.

Мета узлов навигации

Мета узлов навигации состоит из:

- Для узлов навигации первого порядка - тех узлов, которые находятся непосредственно в секции навигации: поле "code" + .json и находится в директории, имя которой совпадает с именем файла секции навигации к которому относится узел навигации.

Например: В директории navigation есть файл секции навигации simpleTypes.section.json. И есть узел навигации classString.json, который размещается в секции simpleTypes. Файл узла навигации будет иметь путь: navigation\simpleTypes\classString.json.

- Для узлов навигации второго порядка - тех узлов, которые входят в группу (особый тип узлов навигации, поле "type" в которых содержит значение 0). Разница в том, что поле "code" у таких узлов составное и состоит из поля "code" группы и личного наименования.

Например: navigation\relations\classReference.refBase.json. Это файл узла навигации refBase, который находится в группе classReferense секции навигации relations.

Условия выборки - "conditions"

Условия выборки "conditions" - это фильтры при открытии списка объектов.

Сохраненные фильтры

Перед открытием любой страницы, в реестрах происходит считывание фильтров, которые подходят для данного окна. Подходящие фильтры состоят из двух частей:

1. Общие фильтры, которые применимы для всех классов.
2. Фильтры, сохраненные конкретно для данного класса.

Общие фильтры для всех классов

Общие фильтры для всех классов - это фильтры, которые отображаются для всех открываемых классов. Их отличие в коде в том, что в атрибуте класса у общего фильтра стоит ключевое слово ALL, у персональных фильтров в этом атрибуте стоит название класса, для которого он применим.

Чтобы сделать фильтр для всех классов, при сохранении поставьте “Для всех классов”.

Фильтры для конкретных классов

Чтобы создать фильтр для конкретного класса, откройте объекты этого класса, сгенерируйте фильтр и сохраните его убедившись, что поле “Для всех классов” не отмечено.

Реализация в коде

Реализована единая спецификация выражений, как для вычисляемых выражений, так и для условий отбора и расчетов в агрегации.

В основном работаем с файлом _list-filter-ui - именно он запускает поиск нужных фильтров, а также разбирает текущие данные для создания новых фильтров. В файле _list-filter-ui описано какие атрибуты могут участвовать в создании фильтров и как именно они должны выглядеть и сохраняться (например, дата и чекбокс выглядят по разному).

В параметре cond находятся данные фильтра, которые в последствие подставляются в условие для поиска (в файле _metaCRUD.js).

Example

```
if (cond !== undefined && cond !== '' && cond !== 'undefined') {
  var condObj = JSON.parse(cond);
  if (Array.isArray(condObj) && condObj.length > 0) {
    for (var k = 0; k < condObj.length; k++) {
      if (condObj[k].type === 6) {
        condObj[k].value = new Date(condObj[k].value);
      }
      where[condObj[k].property] = getwherebyOperation(condObj[k]);
    }
  }
}
```

NB: новые фильтры - min и max расширяют возможности создания фильтров и условий в меню.

Необходимая мета

Необходимая мета для работы - это класс фильтров ion_filter. Он находится в папке calc, которая по умолчанию является папкой с классами и метой для системы. Кроме одного класса ничего более не нужно. Пересмотреть

Поиск по ссылочным объектам

Если поиск идет по принципу равно или содержит - то ищется в семантике этого объекта. Если поиск идет по принципу - максимум/минимум - то ищем уже значение поля. Таким образом, есть возможность поиска в ссылочных атрибутах, при этом без лишних запросов к базе, что повышает производительность.

```
[{property:okatoNasPunkta_title,operation:20,value:Лес,title:Населенный пункт содержит Лес,type:2}]
```

Фильтры в меню

Есть возможность не только выдавать отсортированные данные в списке из условий меню, но и ограничивать выборки, т.е. применять фильтры.

Мета отвечающая за работу фильтров

Пример меню с фильтром

```
{
  "code": "passportObject.naselenie",
  "type": 1,
  "orderNumber": 10,
  "caption": "Население",
  "classname": "naselenie",
  "container": null,
  "collection": null,
  "conditions": [
  {"property": "god"
  ,"operation": 10}
  ],
  "sorting": [],
  "pathChains": []
}
```

Атрибут conditions содержит два объекта:

1. property - свойство, по которому происходит фильтрация
2. operation - операция фильтрации

В данном случае этот фильтр имеет такой смысл - показать все объекты класса ‘‘naselenie‘‘ с минимальным годом.

Если нужно указать значение, то третьим атрибутом пойдет value и значение для поиска.

Example:

```
{"property": "god"
  ,"operation": 0
, "value": 2015}
```

Настройка фильтра для отображения объектов класса-наследника

Страницей класса для узла навигации является родительский класс. Если при переходе по данной навигации необходимо отображать объекты класса наследника данного класса, то применяется фильтр вида:

```
{
   property: "atr1.__class",
   operation: 0,
   value: ["childClass@ns"]
}
```

где atr1.__class - атрибут родительского класса, по которому идет выборка объектов, childClass - наследник, объекты которого отображаются в навигации. То есть - показать на форме списка только те объекты, у которых атрибут ‘‘atr1‘‘ является объектом класса-наследника ‘‘childClass‘‘.

Таблица операций

Field	Name	Acceptable values	Description
"property"	Attribute	String only the latin characters with no spaces	Атрибут класса, значение поля которого проверяется на соответствие данному условию данного вектора.
"operation"	Operation	Operation code	Операция, согласно которой производится определение.
		0 - equal to (AND)	
		1 - not equal to (OR)	
		2 - empty (NOT)	
		3 - not empty (MIN FROM)	
		4 - (MAX FROM)	
		5 - < ()	
		6 - >	
		7 - <=	
		8 - >=	
		9 - IN /Similar/	
		10 - contains	
"value"	Value	Depends on the type of operation	Second value for binary operations
"nestedConditions"	Nested selection conditions	Object, the structure is similar to the structure of the selection conditions object itself.	

NB: код операции соответствует разным значениям операций, в зависимости от того, выбран атрибут или нет. Если поле "property" равно null, то кодируется логическое условие, по которому объединяются вложенные условия отбора (указаны в скобках в таблице выше).

Операции для дат

код	значение	системное имя
8	Создаем дату	DATE
9	Добавляем к дате интервал	DATEADD
10	Находим интервал между датами	DATEDIFF
12	Вычитание	
24	День месяца	

Аргументы DATEADD: дата, интервал, ед.изм интервала [ms, s, min, h, d, m, y] (по умолчанию - день(d))

Аргументы DATEDIFF: конечная дата, начальная дата, ед. изм. результата [ms, s, min, h, d, m, y] (по умолчанию - день(d)), логический флаг приведения к целому числу

Сравнение текущей даты с месяцем

Настройка выборки объектов в списке с возможностью сравнения значения даты с любым месяцем года. Например, настройка фильтра таким образом, чтобы в навигации показывались только те объекты, у которых значение атрибута “Дата окончания” - текущий месяц.

Для этого вычисляется начало текущего месяца. После этого к нему можно добавлять или вычитать произвольное колличество месяцев и сравнивать полученный результат с необходимой датой.

Вычисление конца текущего месяца:

```
{
  "property": null,
  "operation": 9,
  "value": null,
  "nestedConditions": [
    {
      "property": null,
      "operation": 9,
      "value": null,
      "nestedConditions": [
        {
          "property": null,
          "operation": 8,
          "value": ["today"],
          "nestedConditions": []
        },
        {
          "property": null,
          "operation": 12,
          "value": null,
          "nestedConditions": [
            {
              "property": null,
              "operation": null,
              "value": [0],
              "nestedConditions": []
            },
            {
              "property": null,
              "operation": 24,
              "value": null,
              "nestedConditions": [
                {
                  "property": null,
                  "operation": 8,
                  "value": ["today"],
                  "nestedConditions": []
                }
              ]
            }
          ]
        },
        {
          "property": null,
          "operation": null,
          "value": ["d"],
          "nestedConditions": []
```

(continues on next page)

(continued from previous page)

```
        }
      ]
    },
    {
      "property": null,
      "operation": null,
      "value": [1],
      "nestedConditions": []
    },
    {
      "property": null,
      "operation": null,
      "value": ["m"],
      "nestedConditions": []
    }
  ]
}
```

1. Для начала вычисляется значение дня месяца для текущей даты:

```
{
  "property": null,
  "operation": 24,
  "value": null,
  "nestedConditions": [
    {
      "property": null,
      "operation": 8,
      "value": ["today"],
      "nestedConditions": []
    }
  ]
}
```

2. Получено условное значение “d”. Далее необходимо отнять полученное значение от 0 (0-d):

```
{
  "property": null,
  "operation": 12,
  "value": null,
  "nestedConditions": [
    {
      "property": null,
      "operation": null,
      "value": [0],
      "nestedConditions": []
    },
    {
      "property": null,
      "operation": 24,
      "value": null,
      "nestedConditions": [
        {
          "property": null,
          "operation": 8,
          "value": ["today"],
```

(continues on next page)

(continued from previous page)

```
        "nestedConditions": []
      }
    ]
  }
 ]
}
```

3. Получено условное значение “-d”. Далее к текущей дате прибавляется значение “-d” дней:

```
{
 "property": null,
 "operation": 9,
 "value": null,
 "nestedConditions": [
  {
   "property": null,
   "operation": 8,
   "value": ["today"],
   "nestedConditions": []
  },
  {
   "property": null,
   "operation": 12,
   "value": null,
   "nestedConditions": [
    {
     "property": null,
     "operation": null,
     "value": [0],
     "nestedConditions": []
    },
    {
     "property": null,
     "operation": 24,
     "value": null,
     "nestedConditions": [
      {
       "property": null,
       "operation": 8,
       "value": ["today"],
       "nestedConditions": []
      }
     ]
    }
   ]
  },
  {
   "property": null,
   "operation": null,
   "value": ["d"],
   "nestedConditions": []
  }
 ]
}
```

4. Получено начало текущего месяца.

5. Для вычисления конца текущего месяца нужно прибавить к полученному значению начала ме-

сяца 1 месяц:


```
{
  "property": "date",
  "operation": 5,
  "value": null,
  "nestedConditions": [
    {
      "property": null,
      "operation": 9,
      "value": null,
      "nestedConditions": [
        {
          "property": null,
          "operation": 9,
          "value": null,
          "nestedConditions": [
            {
              "property": null,
              "operation": 8,
              "value": ["today"],
              "nestedConditions": []
            },
            {
              "property": null,
              "operation": 12,
              "value": null,
              "nestedConditions": [
                {
                  "property": null,
                  "operation": null,
                  "value": [0],
                  "nestedConditions": []
                },
                {
                  "property": null,
                  "operation": 24,
                  "value": null,
                  "nestedConditions": [
                    {
                      "property": null,
                      "operation": 8,
                      "value": ["today"],
                      "nestedConditions": []
                    }
                  ]
                }
              ]
            },
            {
              "property": null,
              "operation": null,
              "value": ["d"],
              "nestedConditions": []
            }
          ]
        },
        {
```


(continues on next page)

(continued from previous page)

```
        "property": null,
        "operation": null,
        "value": [1],
        "nestedConditions": []
      },
      {
        "property": null,
        "operation": null,
        "value": ["m"],
        "nestedConditions": []
      }
    ]
  }
 ]
}
```

Мета узлов навигации

JSON

```
{
 "code": "classDatetime",
 "orderNumber": 0,
 "type": 1,
 "caption": "Класс \"Дата/Время [9]\"",
 "classname": "classDatetime",
 "container": null,
 "collection": null,
 "url": null,
 "external": true,
 "hint": null,
 "conditions": [],
 "sorting": [],
 "eagerLoading": {
   "list": { // Здесь задается жадная загрузка для списков
      "internet": ["okato"],
      "someClass1": ["refAttr1", "refAttr2.refAttr3"],
      "someClass2": ["colAttr4"]
   },
   "item": { // Здесь задается жадная загрузка для форм редактирования
      "internet": ["okato", "standart"],
      "someClass1": ["refAttr1", "refAttr2.refAttr3", "refAttr5", "colAttr4"],
      "someClass2": ["colAttr4"]
   }
  },
 "pathChains": [],
 "searchOptions": null
 "metaVersion": "2.0.7"
}
```

Field description

Field	Name	Acceptable values	Description
"code"	System name	Строка латиницей, без пробелов	Системное имя узла навигации может быть составным, если принадлежит узлу навигации типа Группа [0].
"orderNumber"	Serial number	Целое число	Задает порядок сортировки пунктов меню в пределах секции навигации
"type"	Type	Целое число	Задает логику работы пункта меню, выводимые при переходе/активации значения. Накладывает ограничения на прочие поля меты узла навигации.
		0	Группа. Объединяет в себе другие узлы навигации по какому-либо общему признаку, не является страницей класса.
		1	Страница класса. Отображает структуру и объекты класса, заданного в поле “classname”.
		2	Страница контейнера. Выполняет отображение списка объектов в коллекции. Для таких узлов навигации нужно указывать класс и идентификатор объекта-контейнера, а также имя атрибута коллекции.
		3	Гиперссылка. Осуществляет переход на ссылку, заданную в поле “url”
"title"	Заголовок	String	Позволяет дополнительно указать заголовок страницы.
"caption"	Logic name	String	Наименование узла навигации отображаемое в интерфейсе.
"classname"	Класс	Строка латиницей, без пробелов	Если “Тип” - “Страница класса (1)”, то поле обязательно к заполнению.
"container"	ID контейнера	Строка или null	Идентификатор объекта содержащего коллекцию отображаемую на странице.
"collection"	Атрибут коллекции	Строка или null	Имя атрибута коллекции, содержимое которого надо вывести на странице.
"url"	URL	Гиперссылка (принимает любые строки)	Если “Тип” - “Гиперссылка (3)”, то поле обязательно к заполнению.
"external"	Признак внешнего ресурса	Logical	Открывает страницу по ссылке в новом окне, если присвоено значение true.
"hint"	Help text	String	Текст, заданный в этой строке, отображается при наведении на узел навигации, которому она принадлежит.
"conditions"	Условия выборки	Array of objects	Фильтр при открытии списка объектов. Используется для узлов типа «Страница класса» и «Страница контейнера».
"sorting"	Сортировка	Array of objects	Используется для узлов типа «Страница класса» и «Страница контейнера». Здесь задаются параметры сортировки объектов в списке. Параметры задаются аналогично настройкам сортировки и выборки допустимых значений в атрибутах.
"eagerLoading"	Жадная загрузка	Объект	Настройка жадной загрузки в навигации. Если нужна ЖЗ для

Настройка поиска в узле навигации

```
"searchOptions": {
 "person": {
  "searchBy": [ // атрибуты по которым ищем, по умолчанию то, что выводится в колонках
    "surname",
    "name",
    "patronymic"
  ],
  "splitBy": "\\s+", // разбивать поисковую фразу на регулярное выражение, части сопоставить с
↪атрибутами
  "mode": ["starts", "starts", "starts"], // режимы сопоставления - в данном случае "начинается с"
↪(доступны like, contains, starts, ends)
  "joinBy": "and" // режим объединения условий на атрибуты (по умолчанию or)
 }
}
```

Структура в mongoDB (registry)

```
{
    "_id" : ObjectId("578f07aa0ce0024ce143e71e"),
    "code" : "classDatetime",
    "orderNumber" : 0,
    "type" : 1,
    "caption" : "Класс \"Дата/Время [9]\"",
    "classname" : "classDatetime",
    "container" : null,
    "collection" : null,
    "url" : null,
    "hint" : null,
    "conditions" : [],
    "sorting" : [],
    "pathChains" : [],
    "itemType" : "node",
    "section" : "simpleTypes",
    "namespace" : ""
}
```

Мета секций навигации

JSON

```
{
  "caption": "Простые типы",
  "name": "simpleTypes",
  "mode": 0,
  "tags": null
}
```

Field description

Field	Name	Acceptable values	Description
"caption"	Logic name	String	Наименование секции навигации отображаемое в интерфейсе.
"name"	System name	Строка латиницей, без пробелов	Задает в том числе первую часть имени файла меты секции навигации, служебное имя.
"mode"	Режим отображения	Меню: 0	Задает режим отображения меню.
		Содержание: 1	
		Ниспадающий список: 2	
		Иерархия: 3	
"tags"	Теги	Массив строк либо null.	Теги. Могут определять дополнительные свойства секции, влиять на отображение. Не реализовано.

Структура в mongoDB (registry)

```
{
    "_id" : ObjectId("578f07aa0ce0024ce143e720"),
    "caption" : "Простые типы",
    "name" : "simpleTypes",
    "mode" : 0,
    "tags" : null,
    "itemType" : "section",
    "namespace" : ""
}
```

Заголовок страницы - "title"

Поле “title” необходимо, чтобы задать отличный от поля "caption" заголовок страницы в меню навигации.

Description

- Для формирования заголовка страницы в первую очередь используется значение поля "title" соответствующего узла навигации. Если поле "title" не задано - пустая строка, то для формирования заголовка страницы используется поле "caption" меты узла навигации.
- Для формирования заголовка страницы на страницах списка выбора (при открытии списка класса справочника из форм) используется "caption" общей части меты класса.

JSON

```
{
  "code": "administrations.municipals",
  "orderNumber": 0,
  "type": 1,
  "title": "Администрации муниципальных районов", \\ отличается от наименования узла навигации поля `
↪"caption"`
  "caption": "Муниципальные районы",
  "classname": "Organisation",
  "container": null,
  "collection": null,
  "url": null,
  "hint": "Список администраций муниципальных районов",
  "conditions": [],
  "sorting": [],
}
```

3.4.4 Мета отчёта

Пример простого полного отчета

```
name: support
caption: Отчет по заявкам в техническую поддержку
sources:
  - name: task
    caption: Заявка
    load:
      - className: task
        results:
          - field: id
            expr: $id
          - field: date
            expr: $dateCreate
          - field: typeComunication
            expr:
              if:
                - eq:
                  - $typeComunication
                  - call
                - 'Звонок'
                - if:
                  - eq:
                    - $typeComunication
                    - metting
                  - 'Встреча'
                  - if:
                    - eq:
                      - $typeComunication
                      - letter
                    - 'Письмо'
                    - if:
                      - eq:
```

(continues on next page)

(continued from previous page)

```
            - $typeComunication
            - mail
          - 'E-mail'
          - ' '
    - field: typeTask
      expr:
        if:
          - eq:
            - $typeTask
            - question
          - 'Консультация'
          - if:
            - eq:
              - $typeTask
              - problem
            - 'Инцидент'
            - if:
              - eq:
                - $typeTask
                - offer
              - 'Предложение'
              - ' '
    - field: predmetSupport
      expr: $support.name
    - field: temaTask
      expr:
        if:
          - nempty:
            - $supportScenario0
          - $supportScenario0.name
          - ' '
    - field: nameClassification
      expr:
        if:
          - nempty:
            - $supportScenario1
          - $supportScenario1.name
          - if:
            - nempty:
              - $supportScenario2
            - $supportScenario2.name
            - if:
              - nempty:
                - $supportScenario3
              - $supportScenario3.name
              - if:
                - nempty:
                  - $supportScenario4
                - $supportScenario4.name
                - if:
                  - nempty:
                    - $supportScenario5
                  - $supportScenario5.name
                  - if:
                    - nempty:
                      - $supportScenario6
```

(continues on next page)

(continued from previous page)

```
                    - $supportScenario6.name
                    - ' '
        - field: coment
          expr: $comment
    index:
      - id
  - name: support
    caption: Заявки в техническую поддержку
    load:
      - source: task
        joins:
          - table: date
            alias: da
            left: id
            right: id
          - table: typeComunication
            alias: comun
            left: id
            right: id
          - table: typeTask
            alias: ta
            left: id
            right: id
          - table: predmetSupport
            alias: sup
            left: id
            right: id
          - table: coment
            alias: com
            left: id
            right: id
        results:
          - field: id
            expr: $id
          - field: date
            expr: $date
          - field: typeComunication
            expr: $typeComunication
          - field: typeTask
            expr: $typeTask
          - field: predmetSupport
            expr: $predmetSupport
          - field: temaTask
            expr: $temaTask
          - field: nameClassification
            expr: $nameClassification
          - field: coment
            expr: $coment
reports:
  - name: technicalSupport
    caption: Заявки ТП
    sheets:
      - name: technicalSupport
        caption: Заявки в техническую поддержку
        type: aggregation
        source: support
```

(continues on next page)

(continued from previous page)

```
    fetch:
      date: $date
      typeComunication: $typeComunication
      typeTask: $typeTask
      predmetSupport: $predmetSupport
      temaTask: $temaTask
      nameClassification: $nameClassification
      coment: $coment
    rangeFilters:
      date:
        caption: За период с|по
        format: date
        inclusive: both
    columns:
      - field: date
        caption: Дата создания
      - field: typeComunication
        caption: Тип коммуникации
      - field: typeTask
        caption: Тип заявки
      - field: predmetSupport
        caption: Предмет поддержки
      - field: temaTask
        caption: Тема заявки
      - field: nameClassification
        caption: Наименование классификации
      - field: coment
        caption: Комментарий
```

Description

Мета отчёта - предназначена для построения шахты данных, содержащей аналитическую информацию по данным из меты системы, организованную в виде таблиц. В мете модуля отчетов указываются источники данных, на основе которых формируется информация для построения отчета, и в дальнейшем формирование колонок таблицы отчета, с указанием ресурса на данные из метаклассов системы.

Мета модуля отчетов находится в папке bi проекта в формате YML.

NB: Определение “Шахта данных”

```
Шахта данных - (смежный термин от англ. Data Mining - глубинный анализ данных)
это некое хранилище, которое содержит глубинную аналитическую информацию
обо всех источниках данных и информацию для построения отчетов,
организованную в виде таблиц.
```

Пример YML

```
name: reportTest
caption: Тестовые данные
sources:
  - name: dataSource
```

(continues on next page)

(continued from previous page)

```
    caption: Источник данных
    load:
      - className: sourceClass
        results:
          - field: id
            expr: $id
          - field: date
            expr: $dateCreate
          - field: name
            expr: $nameObject
    index:
      - id
  - name: test
    caption: Отчет тестовый
    load:
      - source: dataTest
        joins:
          - table: date
            alias: da
            left: id
            right: id
        results:
          - field: id
            expr: $id
          - field: date
            expr: $date
          - field: name
            expr: $name
reports:
  - name: reportTest
    caption: Отчет тестовый
    sheets:
      - name: reportTest
        caption: Отчет тестовый
        type: aggregation
        source: test
        fetch:
          date: $date
        rangeFilters:
          date:
            caption: За период с|по
            format: date
            inclusive: both
        columns:
          - field: date
            caption: Дата создания
          - field: name
            caption: Наименование
```

Описание примера

Отчет тестовый содержит в себе данные из класса “sourceClass”. Источник данных “dataSource” извлекает данные из меты соответствующего класса, которые указаны в свойстве results: . Далее подраздел “test” на основе данных, полученных из источника, указанного в свойстве source: формирует и преобразовывает данные для корректного отображения в таблицах отчета. Свойство joins: задает атрибут,

который является идентификатором для построения отчета (в данном случае id объекта).

Далее система формирует таблицы отчета, на основе преобразованных данных из источника, в разделе reports:. Свойство rangeFilters: содержит информацию о фильтрах, настраиваемых для отчета (в данном случае необходимо указать диапазон дат, в соответствии с данными из класса). В модуле фильтр по диапазону задается через параметры запроса: ?rangeFld[]=0&rangeFld[]=5, где rangeFld - это поле по которому ищем. Если идет поиск по датам - дату передавать в формате локали, которая передается в http-заголовке `'accept-language'`, либо в формате ISO8601. Свойство columns: позволяет формировать колонки таблицы (порядковые номера фактические).

Результат: таблица из двух колонок (Дата и Наименование), в которой будут выводится объекты класса из источника _"dataSource"_ , в соответствии с фильтром по датам, настроенном в rangeFilters:, а количество объектов в таблице будет равно количеству значений идентификатора, настроенном в свойстве joins:.

Пример простого полного отчета можно посмотреть здесь.

Настройка строгости сравнения

Настройка строгости сравнения на границах интервала rangeFilters в отчете:

```
"rangeFilters": {
        "regDate": {
          "caption": "За период с|по",
          "format": "date",
          "inclusive": "both" | "left" | "right"
        }
      }
```

both - обе границы могут быть равны искомым значениям

left - левая граница (меньшая) может быть равна искомым значениям

right - правая граница (большая) может быть равна искомым значениям

Если inclusive не указан - сравнение строгое на обоих границах.

Иерархическая сборка

Настройка иерархической сборки необходима для обработки исходных данных при сборке шахты:

- Чтобы сделать в одном источнике данных выгрузку данных по всей иерархии в базе
- Чтобы вывести данные по первому столбцу с отступами в зависимости от глубины вложенности

Настройка иерархической сборки в шахте данных:

В конфигурации источника настройка "hierarchyBy" представляет собой объект с набором свойств: id, parent, level, order.

```
hierarchyBy:
      id: guidProj
      parent: basicobj1.guidObj
      level: objLevel
      order: objOrder
```

где id - атрибут в данных, идентифицирующий элемент иерархии

parent - атрибут в данных, содержащий идентификатор родительского элемента

level - атрибут в результирующем источнике, куда будет записан уровень вложенности элемента

order - атрибут в результирующем источнике, куда будет записано значение для упорядочивания иерархии при отображении.

Поля objLevel и objOrder это поля для записи значения (их не надо считать, агрегировать и т.д.)

Пример YML

```
reports:
  - name: roadmap
    caption: Дорожная карта
    sheets:
      - name: roadmap
        caption: >-
          Дорожная карта
        type: aggregation
        needFilterSet: true
        needFilterMessage: Выберите проект
        styles:
          objLevel:
            1: text-indent-1
            2: text-indent-2
            3: text-indent-3
          nameObjIndex:
            "3": level2
            "2": level1
            "1": level0
            "0": level0
        source: roadmapSource
        fetch:
          objLevel: $objLevel
          guidObj: $guidObj
          numLevelObj: $numLevelObj
...
```

NB: Иерархическая сборка возможна только на основе источника и невозможна на основе класса.

Алгоритм сборки:

1. Создаем результирующий источник.
2. Делаем выборку корневых элементов, у которых пустое поле parent.
3. Перебираем и записываем элементы в результирующий источник (при этом в спецатрибут element_id - идентификатор (id) обьекта, в level - значение 0, в order - приведенный к строке порядковый номер элемента в выборке, дополненный до длины 6 символов лидирующими нолями).
4. Итеративно делаем выборки следующих уровней вложенности (начиная с 0), до тех пор пока на очередной итерации не будет извлечено 0 объектов. Выборки делаются путем объединения исходного источника с результирующим по связи parent = element_id и ограничению level=текущий уровень вложенности.

5. На каждой итерации перебираем и записываем элементы в результирующий сорс, при этом:
   - в спецатрибут element_id пишем идентфикатор (id) обьекта,
   - в level пишем текущий уровень вложенности,
   - в order пишем конкатенацию order родительского элемента и приведенного к строке порядкового номера элемента в выборке, дополненного до длины 6 символов лидирующими нолями.

Настройка скрытия объектов

Настройка скрытия всех объектов, если табличные фильтры не заданы. Чтобы при открытии отчета все объекты скрывались, пока не будет выбрано значение из списка в фильтре необходимо для него применить настройку "needFilterSet: true".

Отображение в заголовке отчета параметров выборки посредством паттернов

Пример YML

```
...
        byPeriod:
          sum:
            - if:
                - and:
                    - gte:
                        - $date
                        - ':since' # берем из params->since
                    - lte:
                        - $date
                        - ':till' # берем из params->till
                - $amount
                - 0
        byMonth:
          sum:
            - if:
                - and:
                    - eq:
                        - month:
                            - dateAdd:
                                - $date
                                - 10
                                - h
                        - ':month' # берем из params->month
                    - eq:
                        - year:
                            - dateAdd:
                                - $date
                                - 10
                                - h
                        - ':year' # берем из params->year
                - $amount
                - 0
        byYear:
          sum:
            - if:
                - eq:
```

(continues on next page)

(continued from previous page)

```
                - year:
                  - dateAdd:
                    - $date
                    - 10
                    - h
                - ':year' # берем из params->year
              - $amount
              - 0
...
    params:
      year:
        caption: Год
        format: int
      month:
        caption: Месяц
        format: int
        select: # выпадающий список
          '1': январь
          '2': февраль
          '3': март
          '4': апрель
          '5': май
          '6': июнь
          '7': июль
          '8': август
          '9': сентябрь
          '10': октябрь
          '11': ноябрь
          '12': декабрь
      since:
        caption: с
        format: date
      till:
        caption: по
        format: date
...
    columns:
      - field: title
        caption: Показатель
      - field: dimension
        align: center # наименование заголовка в шапке по центру ячейки
        caption: Единица измерения
      - caption: '{$year}' # наименование заголовка в шапке из параметра year
        align: center
        columns: # колонка в шапке - группа вложенных колонок
          - field: byPeriod
            # наименование заголовка в шапке из параметров since и till
            caption: 'с {$since} по {$till}'
            align: center
            format: number
          - field: byMonth
            # наименование заголовка в шапке из параметра month
            caption: 'За {$month}'
            align: center
            format: number
          - field: byYear
```

(continues on next page)

(continued from previous page)

```
        caption: За год
        align: center
        format: number
```

Стилизация строк отчета на основании данных

Пример YML

```
...
    fetch:
      category: $category
      title:
        case:
          - eq:
              - $category
              - AA4
          - 'Выдано заключений, всего в т.ч.:'
          - eq:
              - $category
              - AB5
          - '1. Государственная экспертиза, всего в т.ч.:'
          - eq:
              - $category
              - AC6
          - '- положительных'
          - eq:
              - $category
              - AD7
          - '- отрицательных'
...
      dimension:
        case:
          - eq:
              - $category
              - AA4
          - штук
          - eq:
              - $category
              - AB5
          - штук
...
    styles:
      category:
        AA4: level0
        AB5: level1
        AC6: level2
        AD7: level2
```

Возможность использования комбобоксов в параметрах и фильтрах

Пример YML

```
...
        params:
          year:
            caption: Год
            format: int
          month:
            caption: Месяц
            format: int
            select: # выпадающий список
              '1': январь
              '2': февраль
              '3': март
              '4': апрель
              '5': май
              '6': июнь
              '7': июль
              '8': август
              '9': сентябрь
              '10': октябрь
              '11': ноябрь
              '12': декабрь
          since:
            caption: с
            format: date
          till:
            caption: по
            format: date
...
```

Настройка обработки параметров в фильтре на странице отчета

Пример YML

```
reports:
  ...
  filter:
          eq:
            - $yearStart
            - year:
              - ':dateSelect'
 ...
```

Значение года в атрибуте $yearStart равно значению года из даты в атрибуте :dateSelect.

Настройка пагинатора "pageSize"

NB: Применяется для отчетов с типом type: list.

Для случаев, когда отчет содержит в себе много объектов и на страницах нужно выводить строки постранично, чтобы не нагружать браузер тяжелой обработкой данных.

Пример YML

```
reports:
 - name: test
    caption: Тестовый отчет
    sheets:
      - name: main
        caption: Тестовый отчет
        type: list
        pageSize: 100
```

Настройка вывода построчно

Настройка вывода вложенных данных в отчете построчно настраивается следующим образом:

Пример YML

```
...
reports:
  - name: testReport
  ...
      columns:
        - caption: Группирующее поле
          columns: // поля для группировки
            - field: columns1
              caption: Поле1
              format: string
            - field: columns2
              caption: Поле2
              format: string
    ...
```

Настройка инкрементальной загрузки

Для настройки инкрементальной загрузки данных в источник при сборке шахты необходимо выставить параметр:

```
append: true
```

Он необходим для подгрузки статистики за день в шахту, чтоб не пересчитывать весь объем исходных данных и иметь историю по периодам.

Особенности сортировки объектов

Учитывая функционал агрегации MongoDB - сортировка возможна только по результирующим полям. Это значит, что для обратной совместимости поля результата, по которым сортируем, необходимо называть так же, как и поля в источнике данных.

Пример сортировки (свойство sort):

```
reports:
  - name: sors
    caption: Источник
    sheets:
      ...
        rangeFilters:
          ...
        sort:
          regDateOrder: asc
        columns:
          ...
```

3.4.5 Мета безопасности

Description

Мета безопасности - регулирует настройку прав безопасности на объекты системы. Можно разделить на статическую и динамическую безопасность:

Статическая безопасность - регулирует права доступа на объекты системы для определенной роли.

Динамическая безопасность - регулирует права доступа на объекты системы для конкретной персоны, в соответствии с какими-либо условиями, в то время как групповая динамическая безопасность - это права для группы безопасности.

Настройка динамической безопасности производится в файле deploy.json, а также в файле acl/resources-and-roles.yml. Статическая безопасность задается только в файле acl/resources-and-roles.yml.

Правила формирования идентификаторов ресурсов

- узел навигации - n:::namespace@code
- класс - c:::classname@namespace
- объект - i:::classname@namespace@id
- атрибут - a:::classname@namespace.propertyname
- геомета:
  - узел навигации: geonav:::код узла@namespace
  - слой: geolayer:::код слоя@namespace
  - данные: geodata:::код слоя@namespace@индекс запроса
- пути (модулей):
  - модуль portal: sys:::url:portal/*
  - модуль geomap: sys:::url:geomap/*

Типы прав

Чтение read

read - это право на просмотр информации по объектам класса. Задает разрешение просматривать объекты класса только для чтения и запрещает их создание/редактирование.

```
- id: Users
    name: Обычные пользователи
    permissions:
      n:::ns@navigationName:
        - read
...
```

Запись write

write - это право на создание объектов класса. Задает разрешение на создание новых объектов класса, но запрещает редактирование существующих.

```
- id: Users
    name: Обычные пользователи
    permissions:
      n:::ns@navigationName:
        - write
...
```

Использование use

use - это право на создание объектов класса. Задает разрешение на создание объектов класса, и использование объектов класса в ссылках и коллекциях.

Без use - ссылки тоже отображаются в коллекции. Если есть read, но нет use, то нельзя выбрать объект и поместить его в коллекцию.

```
- id: Users
    name: Обычные пользователи
    permissions:
      n:::ns@navigationName:
        - use
...
```

Удаление delete

delete - это право на удаление объектов класса.

Полный доступ full

full - это право на полный доступ к объектам класса.

Динамическая безопасность

```
"PROJECT_BENEFITIAR": {
"resource":

{ "id": "pm::project" }
,
"attribute": "stakeholders.id"
}
```

Если у проекта в stakeholders.id есть значение ассоциированное с текущим пользователем (настроено подтягивание организации как глобальной роли пользователя), то стоит учитывать текущего пользователя PROJECT_BENEFITIAR и проверить права на ресурс pm:project - эти права и будут правами на проект.

pm:project - это некий виртуальный ресурс безопасности. Необходимо абстрагировать настройки доступа от проверяемого объекта для разных ролей, можно разные ресурсы указать на один класс и наоборот.

Если ресурс не указать, то будут проверяться права на класс объекта. Тогда эту роль можно использовать как статическую, то есть выдавать статические права динамически.

Групповая динамическая безопасность

```
"roleMap": {
  "organization@project-management": {
    "ORGANIZATION_STAFF": {
      "caption": "Сотрудник организации",
        "resource": {
          "id": "pm::organization",
          "caption": "Организация"
        },
        "sids": [ // применять роль, если:
          "$employee", // в атрибуте employee связанное с user значение (user это сотрудник)
          // ИЛИ
          "admin", // user это admin (здесь роль, учетная запись или идентификаторы связанные с user)
          // ИЛИ
          [
             "$boss", // в атрибуте $boss связанное с user значение (user это руководитель)
             // И
             "supervisor" // user это supervisor (роль или учетная запись)
          ]
        ],
        "conditions": {"eq": ["$state", "active"]}, // применять роль только для активных организаций
        "attribute": "employee.id", // добавляется к sids
      }
    }
     }
```

При указании sids каждый уровень вложенности массивов значений меняет вид операции И/ИЛИ. На первом уровне применяется ИЛИ.

Определение ролей пользователя

1. Регистрируем пользователя с полным административным доступом - admin.

2. Под admin в registry в разделе Безопасность.Подразделения заводим иерархию подразделений (код подразделения = идентификатор безопасности).
3. Регистрируем пользователя без прав - user.
4. Под админ в registry в разделе Безопасность.Сотрудники заводим Сотрудника, указываем у него в атрибуте Пользователь пользователя без прав. Привязываем сотрудника к самому нижестоящему подразделению.
5. Заходим под user - прав нет ни на что.
6. Заходим под admin и даем роли (соответствующей самому вышестоящему подразделению) права на произвольные классы и узлы навигации.
7. Заходим под user - есть доступ к объектам, к которым есть доступ у подразделения.
8. Аналогично проверяем применение разрешений по всей иерархии подразделений.

Пример настройки в deploy.json

```
"actualAclProvider":{
      "module": "core/impl/access/aclmongo",
      "initMethod": "init",
      "initLevel": 1,
      "options":{
        "dataSource": "ion://Db"
      }
    },
    "roleAccessManager": {
      "module": "core/impl/access/amAccessManager",
      "initMethod": "init",
      "initLevel": 1,
      "options": {
        "dataSource": "ion://Db"
      }
    },
    "aclProvider": {
      "module": "core/impl/access/aclMetaMap",
      "options":{
        "dataRepo": "ion://dataRepo",
        "acl": "ion://actualAclProvider",
        "accessManager": "ion://roleAccessManager",
        "map": {
          "employee@develop-and-test": {
            "isEntry": true,
            "sidAttribute": "uid",
            "jumps": ["department"]
          },
          "department@develop-and-test": {
            "sidAttribute": "code",
            "jumps": ["superior"]
          }
        }
      }
    }
```

Модель отображения атрибутов и объектов в соответствии с заданными правами

Есть класс [Проекты], в нем содержится атрибут с типом “Коллекция” - [Мероприятия]. Если для класса [Мероприятия] нет прав на чтение, то атрибут этого класса не показывается на форме класса [Проекты].

Если для класса есть динамическая безопасность - то независимо есть или нет права на чтение класса [Мероприятия] - атрибут на форме класса [Проекты] будет показан, но сами объекты мероприятий будут показаны только если на них есть права.

NB: Для отображения атрибута и объектов необходимо задавать как статическую, так и динамическую безопасность на класс по ссылке атрибута.

Если есть статическое право на чтение на класс, то пользователь увидит все объекты этого класса вне зависимости от динамических прав, и дополнительно будет делаться выборка объектов, на которые настроена динамическая безопасность и они будут отображаться для пользователя в соответствии с настройками в ней.

3.4.6 Meta view - general part

Search in object lists “allowSearch”

The "allowSearch" field in the main part of the list view meta enables or disables the display of the search field in the form.

Principle of work

The architecture of the platform and the registry impose the following restrictions.

In order for the search to work in the “LIST” view, one of the following conditions must be met:

- The key field of the class should include one of the following types: String, Date-Time, Integer, Real, Decimal
- There should be the non-key attributes of the same types, but specified as “Indexed”

If none of the conditions is met, the search in the “LIST” view is impossible, so the search field is not displayed. If one or both conditions are met, the search is available and is performed by matching each indexed attribute with the search phrase. If at least one of the attributes matches the search phrase, the object is considered to meet the search condition and is added to the selection.

Depending on the field, the matching is performed according to the following rules:

- String: the search of the string value of the attribute using the regular expression.
- Date-time: the search phrase is converted to date-time, and, if possible, then the equivalence of the attribute value to the received date is checked. The comparison is strict (up to seconds), i.e. if the time is not specified in the search, dates with the time 00:00 will be searched.
- Integer, Real, Decimal: the search phrase is converted to a number and the attribute value is checked for equality.

Search by objects of computable attributes

For objects of calculated attributes, the search will only work if there is a caching setting: doc:more detailes </3_development/metadata_structure/meta_class/meta_class_attribute/attr_cached_true>. Accordingly , there is no need to set indexed for calculated attributes if they are not cached.

Commands

Commands are available operations that can be executed on a class object.

JSON

```
{
    "id": "SAVE",
    "caption": "Сохранить",
    "visibilityCondition": null,
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
}
```

Field description

Field	Name	Acceptable values	Description
"id"	Code	“CREATE” - crate object/object of reference field	Internal code for the object operations.
		“EDIT” - edit object/reference object of the reference field	
		“DELETE” - delete object	
		“CREATE-INLINE” - create object (not in the create mode)	Create object without opening the create form (to speed up the objects creation).
		“SAVEANDCLOSE” - save changes and close	
		“SAVE” - save changes	
		“REMOVE” - delete the reference to the reference object	
		“ADD” - add the reference to the reference object	
"caption"	Name	String	Visible name - the signature on the action button (if available).
"visibilityCondition"	Visibility conditions	Строка или null	The condition under which the action button is available.
"enableCondition"	Activity condition	Строка или null	The condition under which the action button is available.
"needSelectedItem"	Visibility condition - presence of the selected item	Boolean	The field is set to true for commands that require the selected item to activate it.
"signBefore"	Electronic signature of incoming data	Boolean	
"signAfter"	Electronic signature of outgoing data	Boolean	
"isBulk"	Group	Boolean	Sign of a batch operation, for the commands of reference fields. It is set to true for commands that are executed with all reference objects at the same time.

Logic of actions on reference objects

The "Select" operation is setting the connection between objects, regardless of the type of connection, it should be possible to set (and break) it at the level of business logic.

1. If this is a one-to-many reference relationship, (i.e. a reference to a key), then taking the “one” side for A, the “many” side for B, we implement:
    - on the side of the B object (reference to A): the " SELECT " operation, find the object (A), set the value of the reference attribute of the B object equal to the key of the A object (reset the attribute to break the connection);
    - on the side of the A object (collection or back reference): the " ADD " operation, find the object (B), set the value of the reference attribute of this object (B) equal to the key of the object A, the " REMOVE " operation - select the object (B) in the collection, nullify the reference to A.
2. If this is a many-to-many reference connection, (i.e. a link to a non-key attribute), then for both ends of the collection implement connections:
    - The “ADD” operation: find an object, set the value of its reference to the corresponding attribute of the container. The object will be in the collections of all containers with the corresponding attribute value, this is the specificity of this type of connections)
    - The "REMOVE" operation: select an object in the collection, reset the link, and the object is also removed from the collections of all containers with the corresponding attribute value.
3. If this is a many-to-many connection without a reference (communication through a system intermediate entity, this includes both “direct” and “back” collections, as different ends of connection), then for both collections implement:
    - The “ADD” operation: find an object and create the entity-connection with a container, the object is displayed only in the collection of this container
    - The “REMOVE” operation: select an object in the collection, delete the connection with a container, the object disappears only from the collection of this container

From the viewpoint of UI there are no differences between the types of connections at all, everywhere we operate with collections, and the ADD and REMOVE operations are available everywhere. And it is possible to customize the certain buttons at the collection field at the level of the view mode. In business logic, standard handlers for the ADD and REMOVE buttons should be implemented in accordance with the logic described above.

Logic of actions on class objects

The field "commands" , specified in the general part of the meta of the class views, specifies the list of actions allowed for objects of this class.

In the general part of the meta of class views, the commands of the following "id" codes can be specified:

1. "CREATE" - create object
2. "EDIT" - edit object
3. "DELETE" - delete object
4. "SAVEANDCLOSE" - save changes and close
5. "SAVE" - save changes

Use the following commands for the attribute with "type":2:

1. "SELECT" - add
2. "EDIT" - edit
3. "REMOVE" - delete

Структура в mongoDB (registry)

```
{
    "id" : "SAVE",
    "caption" : "Сохранить",
    "visibilityCondition" : null,
    "enableCondition" : null,
    "needSelectedItem" : false,
    "signBefore" : false,
    "signAfter" : false,
    "isBulk" : false
}
```

Override Mode

The Override Mode field:"overrideMode" allows to set two types of the Override Mode "Overlap" and "Override".

Types of the Override Mode in the meta view:

Type 1 - "Override" - overrides the standard display mode of the corresponding attributes specified in the meta view of the class. Attributes that are not specified in the meta view of the class are displayed in a standard mode on the form according to the specified default order.

Type 2 - "Overlap" - only attributes specified in the meta view are displayed.

Example

```
"actions": null,
"siblingFixBy": null,
"siblingNavigateBy": null,
"historyDisplayMode": 0,
"collectionFilters": null,
"version": null,
"overrideMode": 1,
"commands": [
```

Highlighting lines in the list

Conditions of highlighting the lines in the list are specified with the formula.

The list of supported functions can be viewed: doc here </3_development/metadata_structure/meta_class/meta_class_attr
The setting is set in the general part of the list view meta and looks like this:

```
"styles": {
    "css-class": "формула"
  }
```

where the "css-class" is a class of available design themes and the "формула" is a formula with a condition for highlighting the strings in the list.

Available themes of the "css-class" design

- attention-1 - red
- attention-2 - yellow
- attention-3 - orange
- attention-4 - gray
- attention-5 - blue
- attention-6 - green

Purpose of use

When opening the view list of objects, the strings of the table are highlighted according to the conditions of the formula and the theme class.

Example

```
"styles": {
   "attention-1": "lt($real, 0)"
 },
```

$real - any integer. If $real is less than 0, then the column is highlighted in red.

Tabs

Tabs are set in the create and edit view to divide class attributes into separate tabs on the form.

Structure:

```
{
  "tabs": [
    {
       "caption": "1",
         "fullFields": [],
         "shortFields": []
    },
    {
       "caption": "2",
         "fullFields": [],
         "shortFields": []
    }
  ]
}
```

where, caption is the tab name and fullFields are attributes on the tab

Example


```
{
  "tabs": [
    {
      "caption": "Первая вкладка",
      "fullFields": [
        {
          "caption": "Первый атрибут на первой вкладке",
          "type": 1,
          "property": "tab_1_1",
          "size": 2,
          "maskName": null,
          "mask": null,
          "mode": null,
          "fields": [],
          "hierarchyAttributes": null,
          "columns": [],
          "actions": null,
          "commands": [],
          "orderNumber": 10,
          "required": false,
          "visibility": null,
          "enablement": null,
          "obligation": null,
          "readonly": false,
          "selectionPaginated": true,
          "validators": null,
          "hint": null,
          "historyDisplayMode": 0,
          "tags": ["css:background-color:#AFFFAF"]
        },
        {
          "caption": "Второй атрибут на первой вкладке",
          "type": 1,
          "property": "tab_1_2",
          "size": 2,
          "maskName": null,
          "mask": null,
          "mode": null,
          "fields": [],
          "hierarchyAttributes": null,
          "columns": [],
          "actions": null,
          "commands": [],
          "orderNumber": 20,
          "required": false,
          "visibility": null,
          "enablement": null,
          "obligation": null,
          "readonly": false,
          "selectionPaginated": true,
          "validators": null,
          "hint": null,
          "historyDisplayMode": 0,
          "tags": null
        }
```


(continues on next page)

(continued from previous page)

```
    ]
```

Представление Комментарий для атрибутов типа “Коллекция”

> warning Для корректной работы функционала обязательно в зависимостях проекта должен быть указан репозиторий viewlib.

Подключение:

1. В файле package.json проекта:

```
...
"ionMetaDependencies": {
   "viewlib": "0.7.1"
 }
...
```

2. В директорию application рядом с текущим проектом добавить репозиторий проекта viewlib

Инструкция по подключению функционала

Представление реализуется посредством шаблона вида templates/registry/item_footer.ejs. Обратите внимание на пояснение строк после знака // :

```
ejs
<%
let status = item.get('status'); // атрибут со статусом БП
let readOnly = state === 'conf' || status === 'approv'; //статус в котором отображаем коллекцию на
↪форме
if ((item.getMetaClass().checkAncestor('classColl@ns')) //класс, в котором содержится атрибут Комментарий
  && item.getItemId() && (status === 'onapprov' || readOnly)) { // статус, на котором атрибут
↪отображается только для чтения
  let comments = resolveTpl('comments', null, true);
  if (comments) {
    let prop = item.property('atrClassColl'); //системное имя атрибута с представлением Комментарий
    let commId = `${form.ids.attr}_${prop.getName()}_com`;
%>

<div class="line-tabs tabs">

  <div id="item-footer-order-toggle" class="order-toggle asc" data-direction="asc"
      title="Изменить порядок в списке">
    <span class="glyphicon"></span>
  </div>

  <ul class="nav nav-tabs">
    <li class="active">
      <a href='#footer-tab-1' data-toggle="tab">
        <%- prop.getCaption() %>
      </a>
    </li>
  </ul>
```

(continues on next page)

(continued from previous page)

```
  <div class="tab-content">
    <div id="footer-tab-1" class="tab-pane active">
      <div class="comments">
        <%-partial(comments, {
          item,
          id: commId,
          property: prop,
          comment: { // атрибуты для класса по ссылке из атрибута Коллекции
            text: 'descript',
            user: 'owner',
            parent: 'answlink',
            photo: 'owner_ref.foto.link'
          },
          count: 100,
          orderToggleId: '#item-footer-order-toggle',
          readOnly
        })%>
      </div>
    </div>
  </div>
</div>

<script>
  $(function () {
    $('#<%= commId %>').on('comment-added comment-deleted', function () {
      loadWorkflowState();
    });
  });
  $('#item-footer-order-toggle').click(function () {
    if ($(this).hasClass('asc')) {
      $(this).removeClass('asc').addClass('desc').data('direction', 'desc');
    } else {
      $(this).removeClass('desc').addClass('asc').data('direction', 'asc')
    }
  });
  $(document.body).on('mouseenter', '.item-comment', function () {
    $(this).addClass('mouse-enter');
  });
  $(document.body).on('mouseleave', '.item-comment', function () {
    $(this).removeClass('mouse-enter');
  });
</script>
<% }} %>
```

Настройка "options"

Далее подключаем функционал "options" для представление Комментарий на форме представления изменения для атрибута типа “Коллекция”:

```
{
        "caption": "Коментарий",
        "type": 3,
        "property": "coment",
        "size": 2,
```

(continues on next page)

(continued from previous page)


```
"maskName": null,
"mask": null,
"mode": 3,
"fields": [],
"columns": [
 {
  "sorted": true,
  "caption": "Дата",
  "type": 120,
  "property": "date",
  "size": 2,
  "maskName": null,
  "mask": null,
  "mode": null,
  "fields": [],
  "columns": [],
  "actions": null,
  "commands": null,
  "orderNumber": 2,
  "required": false,
  "visibility": null,
  "enablement": null,
  "obligation": null,
  "readonly": false,
  "selectionPaginated": true,
  "validators": null,
  "hint": "",
  "historyDisplayMode": 0,
  "tags": null,
  "selConditions": null,
  "selSorting": null
 },
 {
  "sorted": true,
  "caption": "Подтверждение (Обоснование)",
  "type": 7,
  "property": "descript",
  "size": 2,
  "maskName": null,
  "mask": null,
  "mode": null,
  "fields": [],
  "hierarchyAttributes": null,
  "columns": [],
  "actions": null,
  "commands": null,
  "orderNumber": 1,
  "required": true,
  "visibility": null,
  "enablement": null,
  "obligation": null,
  "readonly": false,
  "selectionPaginated": true,
  "validators": null,
  "hint": null,
  "historyDisplayMode": 0,
```


(continues on next page)

(continued from previous page)


```
      "tags": null,
      "selConditions": null,
      "selSorting": null
    },
    {
      "caption": "Ведущий",
      "type": 2,
      "property": "owner",
      "size": 2,
      "maskName": null,
      "mask": null,
      "mode": 1,
      "fields": [],
      "hierarchyAttributes": null,
      "columns": [],
      "actions": null,
      "commands": null,
      "orderNumber": 6,
      "required": false,
      "visibility": null,
      "enablement": null,
      "obligation": null,
      "readonly": false,
      "selectionPaginated": true,
      "validators": null,
      "hint": null,
      "historyDisplayMode": 0,
      "tags": null
    }
  ],
  "actions": null,
  "commands": [
    {
      "id": "CREATE",
      "caption": "Создать",
      "visibilityCondition": null,
      "enableCondition": null,
      "needSelectedItem": false,
      "signBefore": false,
      "signAfter": false,
      "isBulk": false
    },
    {
      "id": "EDIT",
      "caption": "Править",
      "visibilityCondition": null,
      "enableCondition": null,
      "needSelectedItem": true,
      "signBefore": false,
      "signAfter": false,
      "isBulk": false
    }
  ],
  "orderNumber": 80,
  "required": false,
  "visibility": null,
```


(continues on next page)

(continued from previous page)

```
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": "",
        "historyDisplayMode": 0,
        "tags": null,
        "options": {
          "template": "comments",
          "comments": {
            "textProperty": "descript", // атрибут "Описание" из класса по ссылке
            "userProperty": "owner", // атрибут "Ответственный" из класса по ссылке (отображается имя
↪пользователя, оставившего комментарий)
            "parentProperty": "answlink", // атрибут "Ответ" из класса по ссылке (для возможности "Ответить
↪" на комментарий пользователя)
            "photoProperty": "owner_ref.foto.link", // атрибут "Фото" из класса Персона (отображается фото
↪персоны)
            "dateProperty": "date" // атрибут "Дата" из класса по ссылке
          }
        }
      }
```

Особенности

Мета класса с атрибутом типа “Коллекция” с представлением Комментарий

1. В классе создается обычный атрибут с типом “Коллекция”.
2. В представлении формы изменения создается аналогично стандартному атрибуту с типом “Коллекция”, но с добавлением настройки "options", подробнее смотрите настройка “options”.
3. В классе создается атрибутивный состав и их системные наименования обязательно должны соответствовать наименованиям в шаблоне item_footer.ejs и в свойстве "options". Дополнительно к обязательным - класс может содержать любые атрибуты.

Мета дополнительных классов

Класс Персона должен содержать атрибут, в которых будет задаваться информация об имени пользователя (в данном случае это атрибут “user”) и фотография персоны (атрибут “Фото”), а так же ФИО персоны, которые являются семантикой данного класса.

```
{
  "namespace": "develop-and-test",
  "isStruct": false,
  "key": [
    "id"
  ],
  "semantic": "surname| |name| |patronymic",
  "name": "person",
  "version": "",
  "caption": "Персона",
  "ancestor": null,
```

(continues on next page)

(continued from previous page)


```
  "container": null,
  "creationTracker": "",
  "changeTracker": "",
  "creatorTracker": "",
  "editorTracker": "",
  "history": 0,
  "journaling": true,
  "compositeIndexes": [],
  "properties": [
   {
    "orderNumber": 10,
    "name": "id",
    "caption": "Идентификатор",
    "type": 12,
    "size": 24,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": false,
    "readonly": true,
    "indexed": false,
    "unique": true,
    "autoassigned": true,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
   },
   {
    "orderNumber": 20,
    "name": "surname",
    "caption": "Фамилия",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
```


(continues on next page)

(continued from previous page)


```
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 30,
    "name": "name",
    "caption": "Имя",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
    "refClass": "",
    "itemsClass": "",
    "backRef": "",
    "backColl": "",
    "binding": "",
    "semantic": null,
    "selConditions": [],
    "selSorting": [],
    "selectionProvider": null,
    "indexSearch": false,
    "eagerLoading": false,
    "formula": null
  },
  {
    "orderNumber": 40,
    "name": "patronymic",
    "caption": "Отчество",
    "type": 0,
    "size": null,
    "decimals": 0,
    "allowedFileTypes": null,
    "maxFileCount": 0,
    "nullable": true,
    "readonly": false,
    "indexed": true,
    "unique": false,
    "autoassigned": false,
    "hint": null,
    "defaultValue": null,
```


(continues on next page)

(continued from previous page)


```
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    },
    {
      "orderNumber": 40,
      "name": "user",
      "caption": "Пользователь",
      "type": 18,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": true,
      "readonly": false,
      "indexed": true,
      "unique": false,
      "autoassigned": false,
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    },
    {
      "orderNumber": 70,
      "name": "foto",
      "caption": "Фотография",
      "type": 5,
      "size": null,
      "decimals": 0,
      "allowedFileTypes": null,
      "maxFileCount": 0,
      "nullable": true,
      "readonly": false,
      "indexed": false,
      "unique": false,
      "autoassigned": false,
```


(continues on next page)

(continued from previous page)

```
      "hint": null,
      "defaultValue": null,
      "refClass": "",
      "itemsClass": "",
      "backRef": "",
      "backColl": "",
      "binding": "",
      "semantic": null,
      "selConditions": [],
      "selSorting": [],
      "selectionProvider": null,
      "indexSearch": false,
      "eagerLoading": false,
      "formula": null
    }
  ],
  "metaVersion": "2.0.61.21119"
}
```

Ведение проектных документов

Ведение проектных документов - это настройка "fileshare-list" и "fileshare" предназначена для управления документами, как например, возможность скачать и/или получить ссылку на файл. Для настройки, в представлении атрибута типа “Файл” необходимо указать свойство:

```
"options": {
          "template": "fileshare-list"
        }
```

- fileshare-list - для типа multifile - множественные файлы
- fileshare - для типа file - один файл

В deploy.json в настройках registry подключаем кастомный файл-аплоадер (который с шарой директории и расширенными настройками):

```
"modules": {
  "registry": {
    "globals": {
      ...
      "di": {
        ...
        "fileshareController": {
          "module": "applications/viewlib/lib/controllers/api/fileshare",
          "initMethod": "init",
          "initLevel": 0,
          "options": {
            "module": "ion://module",
            "fileStorage": "ion://fileStorage"
          }
        }
```

Сохранение файлов в облаке

Путь для сохранения файла в облаке настраивается в deploy.json приложения. Для обращения к свойствам объекта используется знак $.

```
{item.названиеСвойства.свойствоСсылочногоОбъекта}
```

т.е. используем ${item.} для того чтобы обозначить что это обращение к объекту.

Example:

```
"modules": {
   "registry": {
     "globals": {
       "storage": {
         "basicObj@project-management": {
           "cloudFile": "/${class}/pm_${attr}/${dddd}/"
         },
         "project@project-management": {
           "cloudFile": "/${item.name} (${item.code})/"
         },
         "eventControl@project-management": {
           "cloudFile": "/${class}/pm_${attr}/${dddd}/"
         },
         "eventOnly@project-management": {
           "cloudFile": "/${class}/pm_${attr}/${dddd}/"
         }
       }
     }
   }
}
```

Настройка позволяет задавать любую структуру хранения файлов (линейно/иерархически, коллекцией/один файл)

Функционал шаринга на файлы

Connection

In the view it's necessary to set the attribute property of the "File"type:

```
"tags": [
          "share"
        ]
```

Usage

When you click on the "share" icon, open the control window similar to one in OwnCloud with the following buttons:

- apply - using the api of cloud storage for the file / directory transfer all the selected parameters :
- share a link - form a "share" on the file / directory - after applying, return to the field to copy the link

- allow editing
- protect with a password- the field for entering password
- set period of validity - the field for entering data
- go to storage - open in a new tab the link on the “share” where there are the files/directory
- close - close the control window of the file

NB: Sharing settings are available for each file, as well as for the entire directory. If the file/directory already has sharing, then when you open the control window, the sharing settings are displayed, and you can change the properties if necessary.

Direct link to file storage

Features

- download immediately
- go to nextCloud and edit

Conditions for link storage

Conditions for storing links created in the process of working with files: TO DO the ability to delete all links created in the process of working with a file after some time or as unnecessary.

Access configuration

Settings of users and access rights to OwnCloud storage objects. In some cases, you must specify users and their rights to the storage objects that you create.

You can specify it in the deploy.json file of the project.

Example:

```
"ownCloud": {
  "module": "core/impl/resource/OwnCloudStorage",
  "options": {
   ...
     "users": [
       {
          "name": "user",
          "permissions": {
            "share": true,
            "create": false,
            "edit": true,
            "delete": false
          }
        }
    ]
  }
}
```

list_view

TODO

Description

Meta view allows to set the desired attribute composition of the class to display on the form according to the view form (list view - list.json, create view - create.json, edit view - item.json) and to specify the overridden and (or) complemented properties for each individual attribute in the meta class of this attribute.

Forms of meta views

Meta view can be divided into two forms:

- List view
- Create and edit view form

List view

List view allows to display the class objects in the form of a list.

JSON

```
{
  "columns": [...],
  "styles": {},
  "actions": null,
  "commands": [...],
  "allowSearch": false,
  "pageSize": null,
  "useEditModels": true,
  "version": null,
  "overrideMode": null,
  "filterDepth": 3
}
```

Field description

Field	Name	Acceptable values	Description
"columns"	Columns	Array of objects	Columns or columns of class attributes, each of which is described by: doc:the attribute part of the view meta <meta_view_attribute>.
"styles"	Highlighting lines	Formula	In accordance with the terms of the formula, the columns in the table are colored in the specified color.
"actions"	Actions	Integer or Null	not used in current version
"commands"	Commands	Array of objects	The set of object operations.
"allowSearch"	Allowed search	Logical	Allows or denies displaying the search form.
"pageSize"	Number of objects per page	Positive integer	Specifies the number of objects on a single page by default.
"useEditModels"	Edit form for detalization	Logical	Allows or denies the use of the edit form for data detalization of a class object.
"version"	Version	String	Версия метаданных.
"overrideMode"	Override mode	0 - Overlap	Sets the override mode of the views.
		1 - Override	
"filterDepth"	Filter query depth in lists	Positive integer	Filter query depth in lists of objects. It equals 2, by default.

Create and edit view form

Create and edit view form allows to create and edit the class objects.

JSON

```
{
  "tabs": [
    {
      "caption": "",
      "fullFields": [...],
      "shortFields": []
    }
  ],
  "actions": null,
  "commands": [...],
  "siblingFixBy": null,
  "siblingNavigateBy": null,
  "historyDisplayMode": 0,
  "collectionFilters": null,
  "version": null,
  "overrideMode": null
}
```

Field description

Field	Name	Acceptable values	Description
"tabs"	Tabs	Объект	Allows creating of multiple pages of objects on one view form.
"caption"	Tab name	String	The name field of the "tabs" object will be displayed in the tab navigation bar.
"fullFields"	Full form of field	Array of objects	Object field "tabs", the array contains attributes that should be displayed in full view, described according to: doc: attribute part of meta views <meta_view_attribute>.
"shortFields"	Short form of field	Array of objects	Object field "tabs", the array contains attributes that should be displayed in short view, described according to: doc: attribute part of meta views <meta_view_attribute>.
"actions"	Actions	Integer or Null	not used in current version
"commands"	Commands	Array of objects	Set of commands of a class object.
"siblingFixBy"	Selection of related objects by	Array of strings	Enumeration of the collection attributes that will be used for selection of related objects.
"siblingNavigateBy"	Navigate to related objects by	Array of strings	Enumeration of the collection attributes that will be used to navigate to related objects.
"historyDisplayMode"	History display	Integer	Specify the format for displaying the history of object changes.
"collectionFilters"	Filtering collections	Array of objects	Select attributes from the collections by which the filtering will be performed.
"version"	Version	String	Версия метаданных.
"overrideMode"	Override mode	0 - Overlap	Sets the override mode of the views.
		1 - Override	

3.4.7 Meta view - attribute part

Commands

Commands are available operations that can be executed on a class object.

JSON

```
{
    "id": "SAVE",
    "caption": "Сохранить",
```

(continues on next page)

(continued from previous page)

```
    "visibilityCondition": null,
    "enableCondition": null,
    "needSelectedItem": false,
    "signBefore": false,
    "signAfter": false,
    "isBulk": false
  }
```

Field description

Field	Name	Acceptable values	Description
"id"	Code	“CREATE” - crate object/object of reference field	Internal code for the object operations.
		“EDIT” - edit object/reference object of the reference field	
		“DELETE” - delete object	
		“CREATE-INLINE” - create object (not in the create mode)	Create object without opening the create form (to speed up the objects creation).
		“SAVEANDCLOSE” - save changes and close	
		“SAVE” - save changes	
		“REMOVE” - delete the reference to the reference object	
		“ADD” - add the reference to the reference object	
"caption"	Name	String	Visible name - the signature on the action button (if available).
"visibilityCondition"	Visibility conditions	Строка или null	The condition under which the action button is available.
"enableCondition"	Activity condition	Строка или null	The condition under which the action button is available.
"needSelectedItem"	Visibility condition - presence of the selected item	Boolean	The field is set to true for commands that require the selected item to activate it.
"signBefore"	Electronic signature of incoming data	Boolean	
"signAfter"	Electronic signature of outgoing data	Boolean	
"isBulk"	Group	Boolean	Sign of a batch operation, for the commands of reference fields. It is set to true for commands that are executed with all reference objects at the same time.

Logic of actions on reference objects

The "Select" operation is setting the connection between objects, regardless of the type of connection, it should be possible to set (and break) it at the level of business logic.

1. If this is a one-to-many reference relationship, (i.e. a reference to a key), then taking the “one” side for A, the “many” side for B, we implement:
    - on the side of the B object (reference to A): the " SELECT " operation, find the object (A), set the value of the reference attribute of the B object equal to the key of the A object (reset the attribute to break the connection);
    - on the side of the A object (collection or back reference): the " ADD " operation, find the object (B), set the value of the reference attribute of this object (B) equal to the key of the object A, the " REMOVE " operation - select the object (B) in the collection, nullify the reference to A.
2. If this is a many-to-many reference connection, (i.e. a link to a non-key attribute), then for both ends of the collection implement connections:
    - The “ADD” operation: find an object, set the value of its reference to the corresponding attribute of the container. The object will be in the collections of all containers with the corresponding attribute value, this is the specificity of this type of connections)
    - The "REMOVE" operation: select an object in the collection, reset the link, and the object is also removed from the collections of all containers with the corresponding attribute value.
3. If this is a many-to-many connection without a reference (communication through a system intermediate entity, this includes both “direct” and “back” collections, as different ends of connection), then for both collections implement:
    - The “ADD” operation: find an object and create the entity-connection with a container, the object is displayed only in the collection of this container
    - The “REMOVE” operation: select an object in the collection, delete the connection with a container, the object disappears only from the collection of this container

From the viewpoint of UI there are no differences between the types of connections at all, everywhere we operate with collections, and the ADD and REMOVE operations are available everywhere. And it is possible to customize the certain buttons at the collection field at the level of the view mode. In business logic, standard handlers for the ADD and REMOVE buttons should be implemented in accordance with the logic described above.

Logic of actions on class objects

The field "commands" , specified in the general part of the meta of the class views, specifies the list of actions allowed for objects of this class.

In the general part of the meta of class views, the commands of the following "id" codes can be specified:

1. "CREATE" - create object
2. "EDIT" - edit object
3. "DELETE" - delete object
4. "SAVEANDCLOSE" - save changes and close
5. "SAVE" - save changes

Use the following commands for the attribute with "type":2:

1. "SELECT" - add
2. "EDIT" - edit
3. "REMOVE" - delete

Структура в mongoDB (registry)


```
{
    "id" : "SAVE",
    "caption" : "Сохранить",
    "visibilityCondition" : null,
    "enableCondition" : null,
    "needSelectedItem" : false,
    "signBefore" : false,
    "signAfter" : false,
    "isBulk" : false
}
```

Activity conditions

Description

Activity conditions set the availability of the field for editing in the view. The syntax for conditions is the same as in: doc: display conditions <visibility>.

Example in JSON:


```
{
  {
        "caption": "Основание для условия активности",
        "type": 1,
        "property": "enablement_condition_base",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 20,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
  },
  {
        "caption": "Поле активно, если основание заполнено",
        "type": 1,
```


(continues on next page)

(continued from previous page)


```
        "property": "enablement_condition_use",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 30,
        "required": false,
        "visibility": null,
        "enablement": ".enablement_condition_base !\u003d \u0027\u0027",
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
  },
  {
        "caption": "Поле активно, если в основании \u00271\u0027",
        "type": 1,
        "property": "enablement_condition_1",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 40,
        "required": false,
        "visibility": null,
        "enablement": ".enablement_condition_base \u003d\u003d \u00271\u0027",
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
  }
}
```


Field sizes

```
module.exports = {
  TINY: 0,
```

(continues on next page)

(continued from previous page)

```
  SHORT: 1,
  MEDIUM: 2,
  LONG: 3,
  BIG: 4
};
```

Fields

Description

Fields - contain class attributes that are grouped by some criterion (for more detailed information see : doc: Type" Group [0]") <type_group>.

Attention: this property applies only to the attribute type "Group [0]".

Example

NB: On the view form, the specified attributes are displayed on the lower level of the hierarchy, the name of the group is on the top.

```
{
  "tabs": [
    {
      "caption": "",
      "fullFields": [
        {
          "caption": "nameGroup",
          "type": 0,
          "property": "",
          "size": 2,
          "maskName": null,
          "mask": null,
          "mode": null,
          "fields": [
            {
              "caption": "atr1",
              "type": 1,
              ...
            },
            {
              "caption": "atr2",
              "type": 1,
              ...
            }
      ]
    }
      ],
      "shortFields": []
    }
  ]
}
```

Input masks

Description

Input mask provide the possibility to set a pattern or a template for an input data. It is used to facilitate the processing of values that have a fixed pattern, for example, telephone numbers. Input masks allow users to enter a predefined format in an attribute.

The "mask" field in meta view

Example

```
{
  "tabs": [
    {
      "caption": "Информационная система",
      "fullFields": [
        {
          "caption": "Уникальный идентификационный номер ОУ",
          "type": 1,
          "property": "OuId",
          "size": 2,
          "maskName": null,
          "mask": null,
```

At the moment, it is possible to set a mask as a string or an object in the "mask" field.

Default masks

Default masks can be redefined in the extensions:

- 9 — number
- a — letter
- A — uppercase letter
- – — number or letter

Mask types

Static masks

You can predefine the static masks. But they can not be modified while you type.

```
"mask": "aa-9999",
```

```
"mask": {"mask": "aa-9999"},
```

Optional masks

Optional masks allow making some part of the mask optional for input. In square brackets [] the optional part of the input is specified.

```
"mask": {"mask": "(99) 9999[9]-9999"},
```

Dynamic masks

Dynamic masks can be modified while typing. In braces {} the dynamic part of the input is set. It applies to the expression before the braces:

- {n} - n repetitions
- {n,m} - from n to m repetitions
- {+} - from 1 and more repetitions
- {*} - from 0 and more repetitions

```
"mask": {"mask": "aa-9{1,4}"},
```

Generated masks

The syntax of the Generated masks is similar to the OR expression. The mask can be one of three variants, specified in the generator. Use | to determine the generator.

```
"mask": {"mask": "(aaa)|(999)"},
```

```
"mask": {"mask": "(aaa|999|9AA)"},
```

KeepStatic

By default: - null (~false). It is used in combination with generator syntax and leaves the mask static as you type. When an array of masks is specified, keepStatic automatically becomes true, unless it was specified through parameters.

```
"mask": { "mask": ["+55-99-9999-9999", "+55-99-99999-9999" ], "keepStatic": true },
```

Result: Enter 1212345123 => as a result we have - +55-12-1234-5123 enter another 4 => as a result we have - +55-12-12345-1234

Additional examples of masks:

“99 99 999999” — series and number of passport

“[+]9 (999) 999-99-99” — mobile phone number

“(999)|(aaa)” - allows to input either three numbers or three characters

Complex masks

Set the complex masks with character validation. Use the inputmask plugin of the 4.0.0 version to determine the valid fields within the definition. Set as an array in field.mask:

```
field.mask: ["X{1,4}-AA №999999", {definitions: {"X": {validator: "[lLvVcCxX]", cardinality: 1, casing: "upper"}}
↪}]
```

Where Index 0 is the mask definition, and index 1 is the additional options.

Masks are also used with “regex”.

```
"mask": {
   "regex": "[A-Za-z]{1,50}"
},
```

Masks by ID

Use the "maskName" field of the attribute part of the meta view to set the mask from the mask preset. To be realized.

Mandatory conditions

Description

Mandatory conditions specify a mandatory condition for filling in a field in the view.

The syntax of the conditions is the same as in display conditions.

Under this condition, the attribute becomes mandatory, otherwise the attribute remains the same as it was specified in the view before the mandatory condition was applied.

Example in JSON:

```
{
    {
      "caption": "Основание для условия обязательности",
      "type": 1,
      "property": "obligation_condition_base",
      "size": 2,
      "maskName": null,
      "mask": null,
      "mode": null,
      "fields": [],
      "hierarchyAttributes": null,
      "columns": [],
      "actions": null,
      "commands": [],
      "orderNumber": 20,
      "required": false,
      "visibility": null,
```

(continues on next page)

(continued from previous page)


```
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
    },
    {
        "caption": "Поле обязательно, если основание заполнено",
        "type": 1,
        "property": "obligation_condition_use",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 30,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": ".obligation_condition_base !\u003d \u0027\u0027",
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
    },
    {
        "caption": "Поле обязательно, если в основании \u00271\u0027",
        "type": 1,
        "property": "obligation_condition_1",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 40,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": ".obligation_condition_base \u003d\u003d \u00271\u0027",
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
```


(continues on next page)

(continued from previous page)

```
        "historyDisplayMode": 0,
        "tags": null
  }
    }
```

Options

** Options** are used to set additional attribute parameters. The interpretation of the settings is performed by a specific implementation of the module.

The following settings are currently used in the registry module:

```
"options": {
   "cssClasses": ["class1", "class2"],
   "cssStyles": {
      "background-color": "#FF0000"
    },
    "template": "some-custom-template"
}
```

- cssClasses - css classes, to apply to the field (in a standard template)
- cssStyles- css styles, to apply to the field (in a standard template)
- template is the name of the template that will be used to render the field. A search will be performed for an ejs template with this name in the standard theme, and in the directories specified in the module templates setting.

In the custom template, all variables that are passed to the standard attribute templates are available:

- item - the object displayed by the form
- prop - the object property represented by the field (if any)
- field- field meta-object
- id - field identifier, generated by the standard algorithm

Sorting - "reorderable": true

It is used for the attribute of the “Collection” type. Determines whether the items in the collection can be sorted with the up and down arrows, and changes the sequence numbers between the two items in the collection.

```
"options": {
  "reorderable": true`
}
```

Displaying the library template

Custom attribute template

Specify the path to the template - located in the project folder .\templates\registry\, example for template below - ./templates/registry/attrs/project/stateView.ejs

```
"options": {
             "template": "attrs/project/stateView"
           }
```

The item with ${item.getItemId()}basic commands is passed to the template:

- receiving item values item.property('stage').getValue()
- identifiers item.getItemId()
- class name item.getClassName() The item of the ‘‘prop‘‘property
- name (code) of the prop.getName() property
- computing the attribute value (if formula) prop.evaluate()
- the attribute value prop.getValue()
- references to class prop.meta.refClass

The same way the ${module} path is transferred to display templates, for example <% stylesheet(${module}/app-static/css/styles.css) -%>

Hierarchy collection treegrid

```
"options": {
        "template": "treegrid/collection",
        "reorderable": true,
        "treegrid": {
          "width": "auto,100,100,100,100,0",
          "align": "left, center,center,center,center, left",
          "sort": "str, date, date, date, date, int",
          "enableAutoWidth": false,
          "paging": {
            "size": 20
          }
        }
      }
```

View in the list of objects for colored icons

In the same way as in the create/edit view meta, you can set overriding templates for the list view meta for each column:

```
...
"options":{
   "template": "templateDir/name"
}
...
```

Example

Digital slider fields

```
"options": {
          "template": "slider",
          "slider": {
            "skin": "dhx_skyblue"
          }
        }
```

Integer slider fields range

It is set in the view for the "options"property:

```
...
"tags": null,
"options": {
  "template": "range"
}
...
```

Connecting the functionality of creating objects for collections without using the inplace form

```
"options": {
  "inplaceInsertion": true,
  "inplaceInsertionClass": "className@namespace"
}
```

Example of "options" of the “Table” attribute

```
{
        "caption": "Таблица",
        "type": 3,
        "property": "table",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": 3,
        "fields": [],
        "columns": [],
        "actions": null,
        "commands": null,
        "orderNumber": 50,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": "",
        "historyDisplayMode": 0,
```

(continues on next page)

(continued from previous page)

```
        "tags": null,
        "options": {
          "inplaceInsertion": true
        },
        "selConditions": [],
        "selSorting": []
      }
```

Indicate the "inplaceInsertionClass" if it is necessary to choose a class (if there are heirs) when creating the object.

Setting the location of the attribute header above the value.

```
"options": {
  "cssClasses": ["top-label"]
}
```

Setting the attribute field for the entire string length (without the name).

```
"options": {
  "cssClasses": ["no-label"]
}
```

Customize the styles applied to the container that contains the input field with the name.

```
"options": {
  "cssStyles": {
   "max-width": "30%",
   "padding": "25px"
 }
}
```

Configuring table column parameters for the “Collection” type attribute

The date column is centered and is 110 pixels, by default.

Available attribute options:

```
"options": {
 "template": "treegrid/collection",
  "treegrid": {
    "width": "150,auto,200",
    "align": "center,left,center",
    "sort": "str, str, str",
    "enableAutoWidth": false,
    "paging": {
      "size": 20
 }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Configuring CSS fields via tags and options

You can configure the CSS fields using tagsor options. In Registry there are css-classes with the following behavior: nolabel, toplabel, full.

For an attribute in the meta view, css-classes are assigned as follows:

In the optionsproperty:

```
"options": {
  "cssClasses": ["toplabel", "fill"]
}
```

In the tags property (backward compatibility)

```
"tags": ["css-class:nolabel", "css-class:fill"]
```

In addition to the classes, you can set the styles directly (they will be applied only to the container).

Set the styles for attribute in a meta view:

In the optionsproperty:

```
"options": {
  "cssStyles": {
    "max-width": "30%",
    "padding": "25px"
  }
}
```

In the tagsproperty:

```
"tags": ["css:min-width:10%", "css:background-color:green"]
```

The description above is only for the standard field templates from the standard theme.

CSS fields

CSS fields set styles for attribute values and are configured by the tags attribute. The similar setting is configured in "options" with the use of templates. For more detailes see “Options”.

Syntax:

```
{
...
  tags: [
    "css-class:myCustomCssClass", // добавляем css-класс
    "css:background-color:green", // добавляем css-стиль
    "css:color:white" // добавляем css-стиль
  ]
}
```

Example:

```
{
    "caption": "Первый атрибут на первой вкладке",
    "type": 1,
    "property": "tab_1_1",
    "size": 2,
    "maskName": null,
    "mask": null,
    "mode": null,
    "fields": [],
    "hierarchyAttributes": null,
    "columns": [],
    "actions": null,
    "commands": [],
    "orderNumber": 10,
    "required": false,
    "visibility": null,
    "enablement": null,
    "obligation": null,
    "readonly": false,
    "selectionPaginated": true,
    "validators": null,
    "hint": null,
    "historyDisplayMode": 0,
    "tags": ["css:background-color:##AFFFAF"]
  }
```

Setting the starting position on the map

Example for the attribute of the “Geodata” type:

```
{
            "caption": "Координаты",
            "type": 100,
            "property": "geo",
            "size": 2,
            "maskName": null,
            "mask": null,
            "mode": 0,
            "fields": [],
```

(continues on next page)

(continued from previous page)

```
            "columns": [],
            "actions": null,
            "commands": null,
            "orderNumber": 34,
            "required": false,
            "visibility": null,
            "enablement": null,
            "obligation": null,
            "readonly": false,
            "selectionPaginated": true,
            "validators": null,
            "hint": "",
            "historyDisplayMode": 0,
            "tags": [
              "tryfind:Хабаровский край",
              "tryfind:$address"
            ],
            "selConditions": [],
            "selSorting": []
          }
```

Result: when you open the coordinate creation form, the coordinates are automatically determined according to the value of the "tags" property. Where $address is the value of the address attribute from the current class.

"Collection" and "Reference" display mode

The display modes of the "Collection" and "Reference" view types are the following constant in the platform:

```
module.exports = {
  TEXT_SIMPLE: 0,
  TEXT_AUTOCOMPLETE: 1,

  COLLECTION_LIST: 0,
  COLLECTION_LINK: 1,
  COLLECTION_LINKS: 2,
  COLLECTION_TABLE: 3,
  COLLECTION_HASHTAGS: 4,

  REF_STRING: 0,
  REF_LINK: 1,
  REF_INFO: 2,
  REF_HIERARCHY: 3,
  REF_SPECIFY: 4,

  GEO_MAP: 0,
  GEO_LOCATOR: 1,
  GEO_CANVAS: 2,

  GROUP_VERTICAL: 0,
  GROUP_HORIZONTAL: 1
};
```

For an attribute with the “Collection” type, the "mode" display modes are implemented on the view form:

- "mode": 0 - List
- "mode": 1 - Reference
- "mode": 2 - List of references
- "mode": 3 - Table
- "mode": 4 - Tag cloud

For an attribute with the “Reference” type, the "mode" display modes are implemented on the view form:

- "mode": 0 - String
- "mode": 1 - Reference
- "mode": 2 - Form
- "mode": 3 - Hierarchical reference
- "mode": 4 - Refining Search

“Hierarchical reference” in more details

In the Hierarchical fieldmode, the filter parameters for the hierarchy levels are displayed based on the specified nested fields. When the field is initially initialized, an ajax request is made to the controller to get the first selection list (the filter is not set). When a response is received from the server, the first filter field is displayed with a selection list. Next, when you select a value in each of the filter fields, the values of the following fields are reset and a new selection list is requested for the next field. Fields that do not have a selection list are hidden. If one option is received in the selection list, it is automatically assigned to the filter and the selection list is determined for the next level of the hierarchy. When a special “transit” value is obtained, the current value is assigned to the next filter instead of the selection list, and the procedure for obtaining the selection list for the next hierarchy level is performed, and the field corresponding to the filter is hidden. When the values of all the filter fields are set, the controller returns a list of object selection by reference, which is automatically displayed in a separate field located after the filter fields.

“Refining search” in more detail

The refinement search fields are used to simplify the search for objects in the reference. The designer of the meta, based on the subject area, can define part of the attributive composition of the sought object as “refining” and, thus, make the task easier for both the database and the user. That means, that instead of selecting from the entire set of products, we first select the value of the “manufacturer” field, then the value of the “product type” field, the options of which are already limited by the previous filter, and so on, thus significantly reducing the selection only to those products that correspond to the refinement attributes. For such a field, it becomes possible to specify fields that will refer to the attributes of the class by reference and become “refining”.

Type Group [0]

Description

Group [0] is the structure of the create and edit view allows to group attributes from other classes in the create/edit view in a horizontal and/or vertical form within one class.

Display modes of the Group [0] type

- GROUP_VERTICAL "mode": 0 - group fields are located under each other
- GROUP_HORIZONTAL "mode": 1 - group fields are arranged horizontally in a row (i.e., in columns, if there is enough space)

Setting format in the meta view for the “Group” type

```
{
  "type": 0, // группа полей
  "mode": 1, // отображается горизонтально
  "fields": [
     // поля
   ]
}
```

Configuring column size parameters:

```
{
  "type": 0, // группа верхнего уровня
  "mode": 1, // колонки
  "fields": [
    {
      "type": 0, // группа-колонка 1
      "mode": 0,
      "size": 0, // очень узкая
      "fields": [
        {
          "property": "attr1",
          "type": 1,
          "caption": "Текстовое поле 1"
        },
        {
          "property": "attr2",
          "type": 1,
          "caption": "Текстовое поле 2"
        }
      ]
    },
    {
      "type": 0, // группа-колонка 2
      "mode": 0,
      "size": 0,  // очень узкая
      "fields": [
```

(continues on next page)

(continued from previous page)

```
        {
          "property": "attr3",
          "type": 1,
          "caption": "Текстовое поле 3"
        },
        {
          "property": "attr4",
          "type": 1,
          "caption": "Текстовое поле 4"
        }
      ]
    },
    {
      "type": 0, // группа-колонка 3
      "mode": 0,
      "size": 3,  // широкая
      "fields": [
        {
          "property": "attr5",
          "type": 1,
          "caption": "Текстовое поле 5"
        },
        {
          "property": "attr6",
          "type": 1,
          "caption": "Текстовое поле 6"
        }
      ]
    }
  ]
}
```

View types

View types are the following constants in the platform:

```
module.exports = {
  GROUP: 0,
  TEXT: 1,
  REFERENCE: 2,
  COLLECTION: 3,
  CHECKBOX: 4,
  COMBO: 5,
  DATE_PICKER: 120,
  DATETIME_PICKER: 6,
  MULTILINE: 7,
  WYSIWYG: 8,
  RADIO: 9,
  MULTISELECT: 10,
  FILE: 11,
  PASSWORD: 12,
  IMAGE: 13,
  NUMBER_PICKER: 14,
```

(continues on next page)

(continued from previous page)

```
  DECIMAL_EDITOR: 15,
  URL: 17,
  PERIOD_PICKER: 60,
  GEO: 100,
  ATTACHMENTS: 110,
  SCHEDULE: 210,
  CALENDAR: 220
};
```

NB: for more details see correspondance table.

Code	Name	Description
0	Group	Special structure of creation and change views.
1	String	View for text data or conversion to text form. Trim is configured (i.e. discarding spaces from the beginning and end of the line)
2	Reference	For reference fields, the relationship is 1kN. Allows you to specify possible operations on objects of the class to which we refer.
3	Collection	For collections, the relationship is Nk1. Allows you to specify possible operations on objects of the class to which we refer.
4	Flag	Boolean type checkbox.
5	Drop-down list	For those with the specified field selectionProvider.
7	Multiline text	A view for the text. Trim is configured (i.e. discarding spaces from the beginning and end of the line)
8	Formatted text	Rich text editor.
9	Alternative choice	Attribute of the “Set [15]” type can have one element of the set. Not implemented
10	Multiple choice	Attribute of the “Set [15]” type can have several elements of the set. Not implemented
11	File selection	View for selecting and uploading a file.
12	Password	In theory it should provide hiding of the entered data, but not implemented.
13	Image selection	View for selecting and loading an image, checks that it is the image that is loaded, and displays a preview.
14	Integer editor	Editor for integers, checks the correctness of the input.
15	Real number editor	Editor for real numbers, checks the correctness of the input, requires the use of . for separating the fractional part.
17	URL	Not implemented
60	Period selection	View that allows you to enter two dates (period boundaries).
100	Geodata	Specifies the view for the “Geodata [100]” type.
110	File set	View for selecting and uploading multiple files. Controls that the files belong to one of the types specified in the meta attribute, the total size of the files and the number.
210	Schedule	View for the attribute type “Schedule [210]”, allows you to set a schedule, displayed in a tabular form.
220	Calendar	View for the attribute type “Schedule [210]”, allows you to specify a calendar, displayed as a calendar.

Visibility conditions

Description

Visibility conditions set the conditions for fields in a class view. It makes the field visible or invisible.

You should set not only the logical fields, as in the example, but also the string, string enumeration (the value is compared by the code of the enumerable) fields.

Syntax

- The \u003d symbol means the = operation.
- The \u0027 symbol means the ' operation.

```
"visibility": ".visibility_condition_base !\u003d \u0027\u0027"
```

These symbols are used to correctly display the conditions in the .json format.

Condition types

- Complex conditions:

```
".state == 'work' || .state == 'result' || .state == 'fin'"
```

where there is a check for three conditions with the combination OR (for the condition AND, the symbol &is used).

The setup syntax itself is similar to conditions in js.

- Logical:

```
".archive == true"
```

where there is a check on the value of the logical attribute.

- Simple condition:

```
".state == 'work'"
```

where there is a check on the attribute value with a selection list.

- Numeric condition:

```
".magistral == 1"
```

where there is a check for the numeric value of the attribute.

- Empty:

```
".meeting == ''"
```

- Not empty:

```
"!! .meeting"
```

where there is a check on the value in the specified attribute.

Example in JSON:


```
{
  {
        "caption": "Основание для условия отображения",
        "type": 1,
        "property": "visibility_condition_base",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 20,
        "required": false,
        "visibility": null,
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
  },
  {
        "caption": "Поле отобразится, если основание заполнено",
        "type": 1,
        "property": "visiility_condition_use",
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 30,
        "required": false,
        "visibility": ".visibility_condition_base !\u003d \u0027\u0027",
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
  },
  {
        "caption": "Поле отобразится, если в основании \u00271\u0027",
        "type": 1,
        "property": "visiility_condition_1",
```


(continues on next page)

(continued from previous page)


```
        "size": 2,
        "maskName": null,
        "mask": null,
        "mode": null,
        "fields": [],
        "hierarchyAttributes": null,
        "columns": [],
        "actions": null,
        "commands": [],
        "orderNumber": 40,
        "required": false,
        "visibility": ".visibility_condition_base \u003d\u003d \u00271\u0027",
        "enablement": null,
        "obligation": null,
        "readonly": false,
        "selectionPaginated": true,
        "validators": null,
        "hint": null,
        "historyDisplayMode": 0,
        "tags": null
    }
  }
```


Description

Meta view - attribute part describes the attribute view fields of the class on the form. The attribute views represent an array in the corresponding fields in the general part of the meta view. The view of each attribute is an object of the following structure:

JSON


```
{
  "caption": "Редактор целых чисел [14]",
  "type": 14,
  "property": "integer_integer",
  "size": 2,
  "maskName": null,
  "mask": null,
  "mode": null,
  "fields": [],
  "hierarchyAttributes": null,
  "columns": [],
  "actions": null,
  "commands": [],
  "orderNumber": 20,
  "required": false,
  "visibility": null,
  "enablement": null,
  "obligation": null,
  "readonly": false,
  "selectionPaginated": true,
  "validators": null,
```


(continues on next page)

(continued from previous page)


```
  "hint": "",
  "historyDisplayMode": 0,
  "tags": null
}
```

Field description

Field	Name	Acceptable values	Description
"sorted"	Sorting allowed	Logical	A field for list views. It is not used in create and modify views. Enables or disables sorting by this column.
"caption"	Title	String	The title of the input field / attribute column displayed in the views.
"type"	Type	Integer - identifier (code) of the type	Attribute view type.
"property"	Attribute	String only the latin characters with no spaces	Indicate the attribute name for system to work with. Therefore it can not be blank. (Except for the Groupe [0] view type).
"size"	Size	Integer - size code of the input field, depends on type	Allows you to specify the field size code for different attribute/view types. Constants in the platform: doc: FieldSizes <meta_view_attribute/field_sizes>.
"maskName"	Mask name	String	If there are any preset masks in the platform, you can set a mask by internal name, specifying it in this field.
"mask"	Mask	String	Allows you to limit the valid attribute values.
"mode"	Режим отображения	Integer - code of the view mode	In some cases, it is necessary to display the attribute data in different ways. In this field you can specify how to display it. : doc: Example of usage </3_development/metadata_structure/meta_class/property_types/type_geodata10
"fields"	Fields	Array of objects	Allows to form create/edit view in a particular way.
"hierarchyAttributes"	Hierarchy attributes	Object or Null according to	not used in current version
"columns"	Columns	Array of objects	Used for attributes of the “Collection [3]” type. Allows you to select attributes to be displayed on the form of presentation in the form of table columns (attributes are taken from a class by reference)
"actions"	Actions	Integer or Null	not used in current version
"commands"	Commands	Array of objects or Null	Describes the acceptable ways for an object of a reference field. Null for the default set of actions.
"orderNumber"	Serial number	Non-negative integer	The order number of the attribute sets the position of the attribute relative to other attributes of the same class in the user interface.
"required"	Mandatory	Logical	Determines whether the filling of this attribute is mandatory when creating/editing an object.
"visibility"	View conditions	String	Sets the condition of displaying the field in the view.
"enablement"	Activity conditions	String	Sets the edit condition (accessibility for editing) of the field in the view with the format similar to the View conditions.
"obligation"	Mandatory conditions	String	Sets the condition which oblige you to fil in the fields in the view. The format is similar to the View conditions.
"readonly"	Read only	Logical	Allows or denies changing the attribute value in this view.
"selectionPaginated"	Page-oriented selection	Logical	Allows or denies page selection list.
"validators"	Validators	String	The name of the validator that checks the values entered in the attribute field. Not implemented.
"hint"	Help text	String	Sets (or redefines the "hint" value of the attribute) message, which will be displayed in the user interface next to the attribute name.
"historyDisplayMode"	History display	Integer	Specifies the format for displaying the history of object changes.

In addition:

- Comment view for attributes of type “Collection” - : doc: more details </3_development/metadata_structure/meta_view/comments>
- Setting up "fileshare-list" and "fileshare" for document management -: doc: learn more </3_development/metadata_structure/meta_view/fileshare>

Structure in mongoDB (registry) for list views


```
{
    "sorted" : true,
    "caption" : "Редактор целых чисел [14]",
    "type" : 14,
    "property" : "integer_integer",
    "size" : 2,
    "maskName" : null,
    "mask" : null,
    "mode" : null,
    "fields" : [],
    "hierarchyAttributes" : null,
    "columns" : [],
    "actions" : null,
    "commands" : [],
    "orderNumber" : 20,
    "required" : false,
    "visibility" : null,
    "enablement" : null,
    "obligation" : null,
    "readonly" : false,
    "selectionPaginated" : true,
    "validators" : null,
    "hint" : "",
    "historyDisplayMode" : 0,
    "tags" : null
}
```


The attribute structure for create and edit views is different only by the "sorted" field, which is absent in the create and edit mode.

3.4.8 Meta workflow

Security of the workflow

Description

Security in a workflow is used to control the rights for a specific object by one user. It is set in the meta class, statuses and transitions of the workflow.

Implementation

In the meta, define in advance :doc: string attribute </3_development/metadata_structure/meta_class/property_types>, which will store the user ID.

To control the rights during the transitions on the workflow, it is necessary to add in the transition an assignment of the current user to the attribute for which rights will be issued for the next WF status.

Then, in the WF status, set the access level in the itemPermissionsproperty:

```
"itemPermissions": [
      {
        "role": ...
        "permissions": ...
      }
    ]
```

- role - indicates the attribute that stores the user ID
- permissions - the number is set by the bit mask, which is related to the role level of access to the object
  - 1 - read
  - 2 - write
  - 4 - delete
  - 8 - use
  - 31 - full access

You can use rights in any order. For example:

- read + write = 3
- read + write + delete = 7
- read + write + delete + use = 15

Attention! In the workflow, the dynamic rights can only provide more access. It is impossible to reduce access rights.

Examples

Assigning of the current user, working with the object, to the person attribute.

```
"assignments": [
   {
      "key": "person",
      "value": "$$uid"
   }
]
```

Adding the itemPermissions to the WF status

```
"states": [
   {
     "itemPermissions": [
      {
        "role": "person",
        "permissions": 15
      }
    ]
   }
 ]
```

Workflow statuses

JSON

```
"states": [
   {
     "name": "new",
     "caption": "Новое",
     "maxPeriod": null,
     "conditions": [],
     "itemPermissions": [],
     "propertyPermissions": [],
     "selectionProviders": []
   }
 ]
```

Field description

Field	Description
"name"	Status system name
"caption"	Status logical name
"maxPeriod"	no data
"conditions"	Conditions for workflow status. Set in the same way as the “Conditions for the selection of valid values”.
"itemPermissions"	Permissions for an object
"propertyPermissions"	Permissions for properties
"selectionProviders"	Selection list of valid values

Workflow transitions

JSON

```
"transitions": [
   {
     "name": "basic",
     "caption": "На согласование",
     "startState": "create",
     "finishState": "inAgreed",
     "signBefore": false,
     "signAfter": false,
     "roles": [],
     "assignments": [],
     "conditions": []
   }
 ]
```

Field description

Field	Description
"name"	Status system name.
"caption"	Status logical name.
"startState"	The initial status of the workflow transition.
"finishState"	The final status of the workflow transition.
"signBefore"	Logical value “Sign before the workflow transition begins”.
"signAfter"	Logical value “Sign at the end of the workflow transition”.
"roles"	List of roles with rights to make the transition.
"assignments"	Assigning values to attributes after the end of the workflow transition.
"conditions"	Conditions for the workflow transition. Set in the same way as the “Conditions for the selection of valid values”.

Assigning values to attributes by reference

Use the "assignments" property in the workflow transition to set the values.

Example

```
...
    "assignments": [
      {
        "key": "resolution.stateRemPet",
        "value": "end"
      }
    ]
...
```

When performing this workflow transition, assign the value “end” for the attribute [stateRemPet] by the reference of the “Reference” attribute [resolution].

Description

Workflow is a clear sequence of actions to obtain a prescribed result. Generally, the workflow is repeated many times. The workflow allows you to display the stages of the process and set the conditions for its execution.

JSON

```
{
  "name": "basic",
  "caption": "Поручения",
  "wfClass": "basic",
  "startState": "new",
  "states": [],
  "transitions": [],
  "metaVersion": "2.0.61"
}
```

Field description

Field	Name	Description
"name"	System name	Workflow system name
"caption"	Logical name	Workflow logical name
"wfClass"	Workflow class	Class to apply the workflow
"startState"	Status	Status assigned to the beginning of the workflow
"states"	List of statuses	List of workflow statuses
"transitions"	Transitions	Transitions between work flow statuses.
"metaVersion"	Versioning	Версия метаданных.

"Contains" condition

Attention! To use the "contains" field, you should configure the eager loading in the collection. Otherwise, the collection will be empty, and the result will always be false. Conditions apply to the object retrieved from the database, without additional requests.

Example

```
{
   "property": "charge",
   "operation": 10,
   "value": null,
   "nestedConditions": [
     {
       "property": "state",
       "operation": 1,
       "value": [
         "close"
       ],
       "nestedConditions": []
     }
   ]
 }
```

Configuration of hints

The "hints" feature represents the instructions in a separate modal window with buttons "continue" or "cancel". When you hover over the button, a pop-up hint appears, for more convenient use of workflows.

Example

```
"transitions": [
   {
     ...
     "confirm": true,
     "confirmMessage": null
   }
```

- "confirm" - confirmation of the action on the workflow transition (+ standard text “you really want to perform the “name” action”).
- "confirmMessage" - unique text to display in confirmation instead of standard text.

Utility to form an array of objects

The utility allows you to create an object in the collection when the main object moves in the specified status. The fields of the created object are automatically filled in accordance with the settings specified for the "values" property.

In the di, in the options property write the following option to attach the indicator value creation utility to the WF status.

state - имя этапа БП

When the object moves in this status, the objects in the collection should be created.

It is possible to use the utility as an “action”. When remaking, just remove the command from the meta view. Configure the utility in the deploy.json of the project. Syntax settings:

```
"map": {
   "workflow@namespace.stage": {
     "className@namespace": { // для объекта какого класса создается объект в коллекцию
        "collection": { // наименование атрибута коллекции, в которой создается объект
           "elementClass": "className2@namespace", // класс, объекты которого создаются утилитой
           "patterns": [
             {
                "values": {
                   "attr1": "string", // строка
                   "attr2": 123, // число
                   "attr3": true,
                   "attr4": "$containerProperty1", // свойство контейнера
                   "attr5": {"add": ["$containerProperty2", 300]} // формула
                },
                "push": [
                  "workflow2@namespace.stage1", // присвоение БП созданных объектов статуса
                ]
             },
             ...
           ]
        },
        ...
     },
     ...
   },
   ....
}
```

Security of the workflow

Security in the workflow is necessary for the control of rights for a specific object by a single user and is set through the class meta, statuses, and transitions of the workflow. More details about WF security.

3.4.9 Correspondence table: attribute types to view types

Code of attribute type	Name of attribute type	Code of the main view type	Name of the main view type	Preferred display type for the main view type	Acceptable view types (selectively)
0	String	1	Text	•	7 - Multiline text, 8 - Formatted text
1	Text	7	Multiline text	•	1 - Text, 8 - Formatted text
2	HTML	8	Formatted text	•	1 - Text, 7 - Multiline text
3	URL	17	URL	•	•
4	Image	13	Image selection	•	•
5	File	11	File selection	•	•
6	Integer	14	Integer editor	•	•
7	Real	15	Real number editor	•	•
8	Decimal	15	Real number editor	•	•
9	Date/Time	120	Date selecion	•	6 - Date-time selection
10	Logical	4	Flag	•	•
11	Password	12	Password	•	•
12	Global identifier	1	Text	•	•
13	Reference	2	Reference	1 - Reference	•
14	Collection	3	Collection	3 - Table	•
15	Multiplicity	9	Alternative choice	•	10 - Multiple choice
16	Structure	0	Group	•	•
17	User type	•	not defined, depends on the main type	•	•
18	User	1	Text	•	•
60	Period	60	Period selection	•	•
100	Geodata	100	Geo object	0 - Map	•
110	File collection	110	File set		

3.4.10 Переменные

Значение атрибутов

$ - применяется вместе с системным наименованием атрибута и возвращает значение указанного атрибута, т.е. если указать $name при формировании какого-либо условия для действий над значениями атрибутов, то значение атрибута name и будет источником для заданного условия.

Пример применения для вычислямого атрибута

```
{
  "formula": {
       "count": [
          "$projects"
       ]
     }
   ...
}
```

В текущем классе есть атрибут типа “Коллекция” projects и, согласно формуле, необходимо посчитать колличество значений данного атрибута.

Результат: количество объектов атрибута projects.

Текущая дата/время, дата

$$now - возвращает текущую дату и время

Пример применения для фильтра допустимых значений в навигации:

```
{
     "property": "dateEnd",
     "operation": 5,
     "value": [
       "$$now"
     ],
     "nestedConditions": []
 }
```

"dateEnd" меньше текущей даты/времени

Пример применения для вывода значения по умолчанию в мете класса:

```
...
"defaultValue": "$$now",
...
```

$$today - возвращает начало суток текущей даты принцип тот же, только дата без времени

Синтаксис форматирования дат в формате momentjs

```
'DD.MM.YYYY'
```

NB: В драйвере к монгодб поддерживаются только основные возможности формата momentjs

Текущий пользователь

$$uid - возвращает текущего пользователя

Пример применения для фильтра допустимых значений в коллекции:

```
{
  "property": "collectionAttr",
  "operation": 10,
  "nestedConditions": [
    {
      "property": "user",
      "operation": 0,
      "value": ["$$uid"]
    }
  ]
}
```

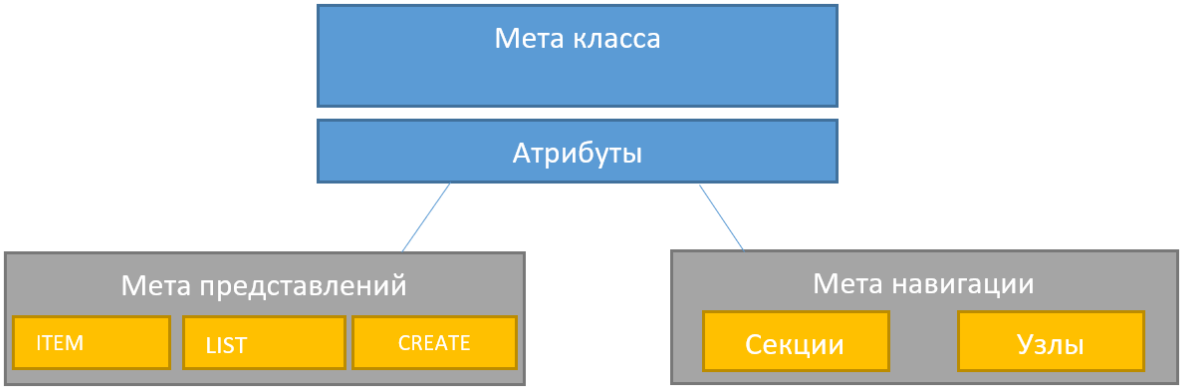
По ссылке атрибута типа “коллекция” отображаются только те объекты, у которых значение атрибута “user” совпадает со значением текущего пользователя.

Metadata (Meta) is a complex of JSON files that describe the set of structures, which the app operates, ways of displaying the data structures in the user interface and navigation on them, as well as displaying the application configuration files.

3.4.11 Types of meta files

1. Meta class
2. Meta view
3. Meta navigation: meta navigation sections, meta navigation nodes.
4. Meta report
5. Meta workflows
6. Meta security
7. Meta admin
8. Geometa

3.4.12 Structure of the meta main types



The structure of the main types of meta can be described as follows:

Meta class is the main source of data generation in the application. A meta class is composed of the attributes (attribute part) and of the class parameters (general part). Attributes are the objects of the "properties" field of the general part, which contains fields relevant to the structure of the meta class and to the data processing in this structure.

Meta class is the basis for meta views, meta navigation, meta reports, meta business processes, etc.

Meta view allows to specify the desired set of attributes for the class to be displayed on the form according to the type of the view form (the form of the list view list.json, of create view create.json, of change view item.json). Also it allows to indicate redefined and (or) complemented properties specified in the meta class of this attribute.

Мета представления + Атрибуты класса = Отображение атрибутов на форме

Meta navigation adjusts the position of elements in the navigation block. The meta navigation is divided into the meta navigation node and the meta navigation section.

3.4.13 Name of meta files:

Meta class	Meta view	Мета навигации
Located in meta directory and consists of the name of the general part of the meta class + .class.json.. Example: adress.class.json.	The name of the meta view directory shows its meta class. The meta view is located in the viewsdirectory. It contains directories whose names match the first part of the name of the meta class file. Example: address@project_name, where address belongs to the address class.	The meta navigation consists of the "name" + field .section.json and is located in navigation directory. Example: Example: workflow.section.json.

3.5 Platform configuration

3.5.1 Authorization and Security Settings

Application configuration parameters, file deploy.json

Application configuration options are designed to identify key features of the system when the application is running at the design stage and changing the default settings.

Setting authorization parameters when working with a password

The parameters and requirements for working with the password are set in di in the configuration of the auth component of the module. But mostly the settings are set globally.

```
{
  "globals": {
    "parametrised": true,
    "plugins":{
      "auth": {
        "module": "lib/auth",
        "initMethod": "init",
        "initLevel": 2,
        "options": {
          "app": "ion://application",
          "logger": "ion://sysLog",
          "dataSource": "ion://Db",
          "acl": "ion://aclProvider",
          "passwordLifetime": "[[auth.passwordLifeTime]]", // Максимальный срок действия пароля
```

(continues on next page)

(continued from previous page)

```
        "passwordMinPeriod": "[[auth.passwordMinPeriod]]", // Минимальный срок действия пароля
        "passwordMinLength": "[[auth.passwordMinLength]]", // Минимальная длина пароля
        "passwordComplexity": { // Требования к сложности пароля
          "upperLower": true, // Обязательно верхний и нижний регистр
          "number": true, // Обязрательно использование хотя бы одного числа
          "special": true // Обязательно использование хотя бы одного специального символа
        },
        "passwordJournalSize": "[[auth.passwordJournalSize]]", // Вести журнал паролей размере паролей
        "tempBlockInterval": "[[auth.tempBlockInterval]]", // Время до сброса счетчика блокировки
        "attemptLimit": "[[auth.attemptLimit]]", // Пороговое значение количества попыток для блокировки
        "tempBlockPeriod": "[[auth.tempBlockPeriod]]" // Продолжительность блокировки учетной записи
      }
    }
```

The values indicated as [[auth.passwordLifeTime]] can be reconfigured in the application settings file - /config/setup.ini. But for this, it is necessary to verify that the “parametrised”: true setting is set to global.

The lifetime is set in the format [duration][unit], while units:

- y - year
- d - day
- h - hour
- m - minute
- s - second

By default, the key parameter values are:

- passwordLifetime = 100y
- passwordMinPeriod = 0d
- passwordMinLength = 8

All created passwords in the system, including the imported ones, are automatically set as required to be changed. To avoid having to change passwords during import, the parameter needPwdReset: false must be specified in the user properties in the imported acl file.

Setting the minimum password length

You can specify the minimum password length to log in, using the "passwordMinLength" property.

```
"plugins":{
   "accounts": {
      "options": {
        "passwordMinLength": 8
      }
   }
}
```

Setting the access rights “aclProvider”

"plugins":{

```
"aclProvider": {
    "module": "core/impl/access/aclMetaMap",
    "initMethod": "init",
    "initLevel": 1,
    "options":{
      "dataRepo": "lazy://dataRepo",
      "acl": "lazy://actualAclProvider",
      "accessManager": "lazy://roleAccessManager"
    }
}
```

Framework and application settings parameters in the config/setup.ini file

The settings are intended to refine and change the application parameters and are initialized at startup. The settings have a higher priority than the configuration settings.

Application settings can also be set in the environment variables; however, the environment variables have a higher priority over the settings.

Overriding password configuration settings

The password parameters set in the deploy.json of the project, if parameterization is enabled and the parameter code is specified, you can override them via the platform settings or through environment variables.

Example of a settings file /config/setup. ini that overrides the values specified in the example file deploy.json.

```
# Максимальный срок действия пароля
auth.passwordLifeTime=90d
# Минимальный срок действия пароля
auth.passwordMinPeriod=75d
# Минимальная длина пароля
auth.passwordMinLength=8
# Вести журнал паролей размере паролей
auth.passwordJournalSize=5
# Время до сброса счетчика блокировки
auth.tempBlockInterval=30m
# Пороговое значение блокировки
auth.attemptLimit=6
# Продолжительность блокировки учетной записи
auth.tempBlockPeriod=30m
# Время жизни авторизованной сессии, при отсутствии активности
auth.sessionLifeTime=4h
```

Setting the session length in the system

Set the session length in the в config/config.json in sessionHandler, using placeholders for the cookie. maxAgeparameter:

```
"sessionHandler": {
  "module": "lib/session",
  "initMethod": "init",
  "initLevel": 1,
  "options": {
```

(continues on next page)

(continued from previous page)

```
    "app": "ion://application",
    "dataSource": "ion://Db",
    "session": {
      "secret": "ion:demo:secret",
      "resave": false,
      "saveUninitialized": true,
      "cookie": {
        "httpOnly": true,
        "secure": false,
        "maxAge": "[[auth.sessionLifeTime]]"
      }
    }
  }
}
```

Add this setting in the deploy.ini-file of the project. The format is the same as for the period setting in the auth:

```
auth.tempBlockPeriod=2s
auth.tempBlockInterval=15m
auth.blockPeriod=1d
auth.sessionLifeTime=2h
```

You can also set it in numbers, and then it will be in milliseconds.

To store the session not in the database, but in the redis caching server, add the setting and caching parameters to the deploy.ini file of the project

```
session.type=redis
cache.redis.host=127.0.0.1
cache.redis.port=6379
```

Setting to disable the authorization form to go to the module page

In the core setting the “auth” field has the excludesetting:

```
"auth": {
  "module": "lib/auth",
  "initMethod": "init",
  "initLevel": 2,
  "options": {
    "app": "ion://application",
    "logger": "ion://sysLog",
    "dataSource": "ion://Db",
    "denyTopLevel": "[[auth.denyTop]]",
    "authCallbacks": ["[[auth.callback]]"],
    "publicRegistration": "[[auth.registration]]",
    "exclude": ["[[auth.exclude1]]", "[[auth.exclude2]]", "[[auth.exclude3]]"]
  }
}
```

So, write the following in the ini-file of the project:

```
auth.exclude[] = /registry/ # исключаем только запросы к корню модуля
auth.exclude[] = /registry/** # исключаем запросы ко всем страницам модуля
```

(continues on next page)

(continued from previous page)

```
auth.exclude[] = \/registry\/khv-svyaz-info@naselenniePunkty\/\w+ # исключаем запросы ко всем страницам модуля
внутри ноды khv-svyaz-info@naselenniePunkty
auth.exclude[] = /registry/api/naselenniyPunkt@khv-svyaz-info/** # исключаем запросы к api класса
```

When you go to the page specified in the module settings, the data is displayed without the authorization.

Deactivation of the authorization for static paths on the example of the develop-and-test project:

```
; Исключение статичных путей ядра из проверки доступа безопасности
auth.exclude[]=/
auth.exclude[]=/vendor/**
auth.exclude[]=/css/**
auth.exclude[]=/fonts/**
auth.exclude[]=/favicon.ico

; Исключение статичных путей модулей из проверки доступа безопасности
auth.exclude[]=/registry/vendor/**
auth.exclude[]=/registry/css/**
auth.exclude[]=/registry/js/**
auth.exclude[]=/registry/app-vendor/**
auth.exclude[]=/registry/app-static/**
auth.exclude[]=/registry/common-static/**
auth.exclude[]=/registry/img/**
auth.exclude[]=/registry/fonts/**
auth.exclude[]=/dashboard/vendor/**
auth.exclude[]=/dashboard/develop-and-test/** ; для проекта develop-and-test
auth.exclude[]=/dashboard/js/**
auth.exclude[]=/registry/viewlib-ext-static/** ; для проекта viewlib-extra
auth.exclude[]=/registry/viewlib-static/js/** ; для проекта viewlib
auth.exclude[]=/gantt-chart/vendor/**
auth.exclude[]=/gantt-chart/gantt/**
auth.exclude[]=/gantt-chart/css/**
auth.exclude[]=/gantt-chart/js/**
auth.exclude[]=/gantt-chart/common-static/**
auth.exclude[]=/gantt-chart/fonts/**
auth.exclude[]=/geomap/vendor/**
auth.exclude[]=/geomap/css/**
auth.exclude[]=/geomap/js/**
auth.exclude[]=/geomap/common-static/**
auth.exclude[]=/geomap/img/**
auth.exclude[]=/geomap/fonts/**
auth.exclude[]=/report/vendor/**
auth.exclude[]=/report/css/**
auth.exclude[]=/report/js/**
auth.exclude[]=/report/common-static/**
auth.exclude[]=/report/img/**
auth.exclude[]=/report/fonts/**

; Исключение всего модуля из проверки доступа безопасности
auth.exclude[]=/portal/**
```

3.5.2 Ways to configure parameters:

- using ini-files
- using environment variable

NB: Priority is given to settings that are set via environment variables, not via ini files. At the same time, the /config/setup.ini settings do not affect the settings of the deploy application - they are only used for the core and modules. Deploy.json is most often parameterized via an ini in the application directory, such as the deploy.ini file.

Example of configuring the parameters of OwnCloud with a user account

First of all, set the parametric settings of the storage in the deploy.json file:

```
"ownCloud":
  "ownCloud": {
          "module": "core/impl/resource/OwnCloudStorage",
          "options": {
            "url": "[[ownCloud.url]]",
            "login": "[[ownCloud.login]]",
            "password": "[[ownCloud.pwd]]"
          }
        }
```

Configure the parameters in deploy by use of the ini-files:

In the deploy.ini ini-file near the deploy.json file set the following parameters:

```
ownCloud.url=https://owncloud.com/
ownCloud.login=api
ownCloud.pwd=api
```

Configure the parameters in deploy by use of the environment variable:

In the environment variables for the NODE set the following parameters:

```
ownCloud.url=https://owncloud.com/
ownCloud.login=api
ownCloud.pwd=api
```

Setting up the limitation

Configure the limitation of switching between system menu items for an anonymous user. The system menu is formed taking into account the access control to the module pages, i.e. if there are no rights to the module page, then the menu item for switching to this module is not displayed in the system menu. In the application's ini file, set auth. checkUrlAccess=true to set the limitation setting.

Changing all references to relative ones

To change all references to relative ones, set the following parameter in the ini-file:

```
app.baseUrl= '/нужный_путь/'
```

If the path is not specified, then it is considered as '/' by default.

Setting the control unit to run jobs on a schedule in admin module

To display the control unit, add the following setting in the ini-file of the project:

```
jobs.enabled=true
```

It will enable the scheduler in the web server process, which will allow you to manage jobs from the admin module.

Scheduler manages job start timers.

Job is a specific task that starts on a timer.

Configuring data caching at the core level

Setting of cached data at the core level allows to correctly recover from the cache eager loaded reference attributes and collections, as well as files and calculated attributes. Lists are cached correctly. Caching is implemented in the geomodule. This setting once and for all solves the problem of circular references when serializing objects.

Write the following in the ini-file of the project:

```
cache.module=memcached
```

Setting the time limit

connectTimeOut - the maximum connection time.

operTimeOut - the maximum time to complete an operation.

```
db.connectTimeOut=
db.operTimeOut=
```

Setting the minimum password length

```
auth.passwordMinLength=8
```

The setting for an individual application can be overridden in the file deploy.json

---

3.5.3 Configuration file deploy.json

Deploy build from separate files

Splitting the deploy.json configuration file

For the convenience of organizing and increasing the readability of the application configuration, deploy.json has the ability to split this file into several separate configuration files.

There can be any separation options: you can take out only the configuration of application modules from it or describe each deploy object in a separate file.

For configuration, in the deploy folder of the application, you need to place arbitrary files with the extensions .json or .yml.

To work correctly in them, it is necessary to preserve the original structure of deploy.json: objects must be nested in the same order as in the source file. With the exception of modules in the application.

For example, if the globals.jobs.ticketClose.di object was defined in deploy.json, then to transfer the di object to a separate file, the structure of these nested objects must be reproduced in this file:

```
---
globals:
  jobs:
    ticketClose:
      di:
        ticketCloser:
          executable: applications/khv-ticket-discount/lib/overnightTicketClose
          options:
            dataRepo: ion://dataRepo
            log: ion://sysLog
            workflows: ion://workflows
```

Module configuration files

In the case of module configs, they can also be

1. Placed in the directories deploy/modules/<module name>/
2. Put the entire config for the module in the deploy/modules/<module name>.yaml (.json) file

The description begins with the root of the module (not the application):

```
---
globals:
  navigation:
    namespaces:
      khv-ticket-discount: Льготные билеты
  eagerLoading:
    "*":
      applicant@khv-ticket-discount:
        item:
          - documents.vidDocument
      flight@khv-ticket-discount:
        list:
          - route.pointDeparture
          - route.pointArrival
  listSearchOptions:
    ticketYear@khv-ticket-discount:
      "*":
        searchBy:
          - flight
```

(continues on next page)

(continued from previous page)

```
        - person
        - numberTicket
        - area
      refDepth: 3
    flight@khv-ticket-discount:
     "*":
      searchBy:
        - route
        - number
      refDepth: 3
 di:
   pmListToDocx:
     module: modules/registry/export/listToDocx
      initMethod: init
     initLevel: 0
     options:
       tplDir: applications/khv-ticket-discount/export/list
       log: ion://sysLog
       injectors:
         - ion://monthTicketStatsInjector
...
```

Global settings in deploy.json

Global application configuration settings include the following sections:

- Application namespace "namespace"
- Parameterization "parametrised"
- Path to templates "theme"
- Caching time and other parameters for static files https://expressjs.com/en/4x/api.html#express.static "staticOptions" (works with NODE_ENV=production)
- Browser tab name"pageTitle"
- Application modules "moduleTitles"- details
- Configure the display of a common system menu for all project modules "explicitTopMenu" - details
- Overriding session storage settings "SessionHandler"
- "actualAclProvider"
- "aclProvider"
- Configure HTML attributes for displaying and saving images in an attribute‘“”fileStorage”‘ - details
- Settings for displaying the user name and avatar in all modules of the project "customProfile" - details
- Setting the eager loading depth‘“”dataRepo”‘ - details
- "accounts"
- "securedDataRepo"
- "ldap"
- "auth"

- "sendmail"
- "nodemailer"
- "emailNotifier"
- "notifier"
- "eventNotifier"
- Configure integration with calendar "icsMailer" - details
- "recache"
- "fact-creator"
- "report-builder"
- "projectReportCreator"

The structure of which is constructed as follows:

```
{
  "namespace": "...",
  "parametrised": true,
  "globals": {
    "theme": "...",
    "staticOptions": {...},
    "pageTitle": "...",
    "moduleTitles": {...},
    "explicitTopMenu": [...],
    "plugins": {
      "sessionHandler": {...},
      "actualAclProvider": {...},
      "aclProvider": {...},
      "fileStorage": {...},
      "customProfile": {...},
      "dataRepo": {...},
      "accounts": {...},
      "securedDataRepo": {...},
      "ldap": {...},
      "auth": {...},
      "sendmail": {...},
      "nodemailer": {...},
      "emailNotifier": {...},
      "notifier": {...},
      "eventNotifier": {...},
      "icsMailer": {...}
    },
    "jobs": {
      "recache": {...},
      "fact-creator": {...},
      "report-builder": {...},
      "projectReportCreator": {...}
    }
  }
}
```

Modules of application

In the “moduleTitles” field specify the modules that will be used in the application. Also, the same modules will be displayed in the system menu.

```
{
  "namespace": "crm",
  "globals": {
    "moduleTitles": {
      "registry": "Тех. поддержка",
      "report": "Отчеты"
    }
  }
}
```

Settings of hiding module in system menu

Set the nullvalue in the module (in the deploy.jsonfile) that you would like to hide in the system menu of the project, for example "ionadmin": null.

```
{
  "namespace": "project-management",
  "parametrised": true,
  "globals": {
    "moduleTitles": {
      "registry": {
        "description": "Проектное управление",
        "order": 10,
        "skipModules": true
      },
      "ionadmin": null
    }
  }
}
```

Configuring the display of a common system menu for all project modules

In order for the system menu to display the same set of items, regardless of which module page you are on, set "explicitTopMenu" in the deploy.json project file at the global level, while retaining the ability to override "explicitTopMenu" in the registry.

Example

```
"globals": {
    "explicitTopMenu": [
      {
        "id":"mytasks",
        "url": "/registry/project-management@indicatorValue.all",
        "caption":"Мои задачи"
      },
      {
```

(continues on next page)

(continued from previous page)

```
      "id":"projectmanagement",
      "url": "/registry/project-management@project",
      "caption":"Проектное управление"
    },
    {
      "type": "system",
      "name": "gantt-chart"
    },
    {
      "type": "system",
      "name": "portal"
    },
    {
      "type": "system",
      "name": "geomap"
    },
    {
      "type": "system",
      "name": "report"
    },
    {
      "id":"distionary",
      "url": "/registry/project-management@classification.okogu",
      "caption":"Справочники"
    },
    {
      "id":"mark",
      "url": "/registry/project-management@person",
      "caption":"Прогресс-индикатор"
    }
  ],
```

Field description

- "id" - identifier of the navigation section
- "url" - url of the navigation section
- "caption" - name of the navigation section
- "name" - system name of the module

Field “plugins”

This field contains settings that allow you to expand the capabilities of the application.

Setting the HTML attributes to display and save images in attributes

"plugins":{

```
"fileStorage": {
  "module": "core/impl/resource/OwnCloudStorage",
  "options": {
```

(continues on next page)

(continued from previous page)

```
    "url": "https://owncloud.iondv.ru/",
    "login": "api",
    "password": "apiapi"
  }
}
```

```
"htmlFiles": {
  "module": "core/impl/resource/FsStorage",
  "initMethod":"init",
  "initLevel": 3,
  "options": {
    "storageBase": "./htmlFiles",
    "urlBase": "/htmlFiles",
    "dataSource": "ion://Db",
    "log": "ion://sysLog",
    "app": "ion://application",
    "auth": "ion://auth"
  },
  "htmlImages": {
     "module": "core/impl/resource/ImageStorage",
     "initMethod": "init",
     "initLevel": 3,
     "options": {
       "fileStorage": "ion://htmlFiles",
       "app": "ion://application",
       "auth": "ion://auth",
       "log": "ion://sysLog",
       "urlBase": "/htmlFiles",
       "thumbnails": {
         "small": {
           "width": 100,
           "height": 100
         }
       }
     }
  }
}
```

"modules": { "registry": { "globals":

```
{
  "refShortViewDelay": 1000, // количество миллисекунд до появления окна с инфо. Если не указан или 0,␣
↪или нет shortView представления, то окно не выводится
  "defaultImageDir": "images",
  "contentImageStorage": "htmlImages"
}
```

Setting to display username and user icon (avatar) in all modules of the project

Set the connection with the icon in the “avatar” field to set the user icon. The system will choose the user avatar from the corresponding class attribute whose object is bound to the current system user.

Example

```
"customProfile": {
  "module": "lib/plugins/customProfile",
  "initMethod": "inject",
  "options": {
    "auth": "ion://auth",
    "metaRepo": "ion://metaRepo",
    "dataRepo": "ion://dataRepo",
    "propertyMap": {
      "person@project-management": {
        "filter": "user",
        "properties": {
          "avatar": "foto"
        }
      }
    }
  }
}
```

Setting the depth of the eager loading

```
"dataRepo": {
  "options": {
    "maxEagerDepth": 4
  }
}
```

Setting the integration with calendar

Integration is performed as follows: the module sends an email with an attached ics file, which specifies the iCalendar event. Outlook treats such a letter as an invitation to a meeting. * Yandex * also adds the meeting to the calendar.

Module configuration:

```
"icsMailer": {
  "module": "applications/extensions/lib/icsMailer",
  "initMethod": "init",
  "initLevel": 2,
  "options": {
    "dataRepo": "ion://dataRepo",
    "transport": {...}, //Настройки smtp-сервера
    "defaults": {...}, //Общий настройки всех писем
    "listeners": [
      {
        "component": //Ссылка на слушаемый компонент
        "events": {
          "...": {// Идентификатор события
            "calendar": {...}, //Настройки календаря, несущего событие и передаваемого в ics-вложении
            "event": {...}, //Настройки VEVENT, передаваемого в ics-вложении
            "filename": "...", //Имя вложенного ics-файла
            "letter": {...} //Настройки письма, отправляемого по событию.
```

(continues on next page)

(continued from previous page)

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

- Configuration details transport and defaults.
- Configuration details letter
- Configuration details calendar
- Configuration details event

For the letter, event, filename and calendar settings, you can use data from the event object by specifying property names with the point refAttr.stringAttr, or wrapping this construction in ${refAttr.stringAttr} when you need to use a template.

Full file example deploy.json

Module settings in deploy.json

- Module “registry”
- Module “geomap”
- Module “gantt-chart”
- Module “report”
- Module “rest”
- Module “portal”
- Module “ionadmin”
- Module “dashboard”
- Module “diagram”

The “registry” module

Registry module is a key module designed directly for operating with data based on metadata structures, including managing projects, programs, events, etc. More details on registry module.

Setting up configurable file saving

In order to set the path for saving the file to the storage, specify:

```
"modules": {
   "registry": {
     "globals": {
...
       "storage": {
```

(continues on next page)

(continued from previous page)

```
        "className@ns":{
            "file_attr":"/${class}/example_${attr}/${dddd}/"
        }
      },
...
```

In the object, the key is the name of the class, then “attribute name”: “relative path”.

Aliases are written to ${alias}. Available aliases:

- class - class name
- attr - attribute name
- also date denominations from ‘‘ moment.js‘‘ are available

Setting to specify the number of characters for search query

For the entire application - ‘‘”listSearchMinLength” ‘‘.

```
"modules": {
  "registry": {
    "globals": {
      "listSearchMinLength": 1
    }
  }
}
```

For the one specific class - "minLength".

```
"modules": {
  "registry": {
    "globals": {
      "listSearchOptions": {
        "className@ns": {
          "*": {
            "searchBy": [
              "atr1"
            ],
            "splitBy": "\\s+",
            "mode": [
              "starts"
            ],
            "joinBy": "and",
            "minLength": 3
          }
        }
      }
    }
  }
}
```

Setting the container assignment when creating the nested object

For cases when it is necessary to assign a value for an attribute by reference, when creating an object, specify the setting for the class that contains the assigned value in the deploy.json file of the application:

```
"registry": {
  "globals": {
    "forceMaster": {
      "name_class@ns": true
    }
  }
}
```

An example of the sequence generators: now for each object its code is the code of its direct container plus the next value of the sequence counter associated with the container object.

Setting the eager loading for printed forms "skipEnvOptions"

More details on printed forms.

Use the skipEnvOptions flag to enable/disable the eager loading.

Example

```
...
"modules": {
   "registry": {
     "globals": {
...
       "di": {
...
          "export": {
            "options": {
             "configs": {
               "class@ns": {
                 "expertItemToDocx": {
                   "type": "item",
                   "caption": "Наименование",
                   "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
                   "extension": "docx",
                   "skipEnvOptions": true,
                   "preprocessor": "ion://expertItemToDocx"
                 }
               }
             }
            }
          }
...
       }
    }
  }
}
...
```

Due to the eager loading the system creates a file very quickly, but it may not always be acceptable.

Configuring notifications about another user editing an object

When setting the notification about another user editing an object, specify the time before blocking in milliseconds:

```
"modules": {
   "registry": {
     "globals": {
       "concurencyCheck": 10000
     }
   }
}
```

ConcurencyChecker component:

The ConcurencyChecker component stores the lock status for objects in the data source. It stores the following parameters:

- full id of an object (class@id),
- date/time of block (blockDate),
- user who blocked.

The component creates the block states, starting a timer that will delete the block record when the timeout expires. If at the moment the timer is triggered, the record is still up to date (blockDate was updated), then the record is not deleted, and the timer is updated.

Logic of the view controller::

Read the registry.concurencyCheck setting (blocking timeout in seconds).

If it is bigger than 0, read the ConcurencyCheker - check the block status.

If not found (expired - blockDate < now() - registry.concurencyCheck), then through the checker write a new block on behalf of the current user. If you found a running block, transfer the information about the block to the template. Display this information on the form in the "read only" mode (globalReadOnly).

An additional controller concurencyState, which takes the id of the object and checks its block status. If the object is not blocked (there is no block, or it is expired), then it blocks the object on behalf of the current user. If the object is locked by the current user, then it updates blockDate to new Date() and returns the block state.

Object form behavior:

If the information about the block is transferred to the template, then a script is added that accesses the concurencyState controller periodically (with a period of registry.concurencyCheck/2).

If you receive the information about blocking by another user, it is displayed (update the message). If the blocking lock intercepted by the current user, the form reloads (it is displayed in the edit mode).

Connecting design resources in the project

This is related, for example, to groups in a particular style, in order not to connect resources through changes to module templates, you need to connect them in the application.

```
"statics": {
  "geoicons": "applications/khv-svyaz-info/icons"
}
```

All that is inside the icons directory is available by the registry/geoicons link.

Configuring the form for specifying export parameters (for printed forms)

Example of params:

```
...
      "di": {
        "pmListToDocx": {
          "module": "modules/registry/export/listToDocx",
          "initMethod": "init",
          "initLevel": 0,
          "options": {
            "tplDir": "applications/project-management/export/item2",
            "log": "ion://sysLog"
          }
        }
...
        "export": {
          "options": {
            "configs": {
              "evaluationPerform@project-management": {
                "rating": {
                  "caption": "Оценка деятельности исполнителя и соисполнителей проекта",
                  "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
                  "extension": "docx",
                  "type": "list",
                  "query": {
                    "filter": {
                      "and": [
                        {
                          "eq": [
                            "$basicObjPerform",
                            ":project"
                          ]
                        },
                        {
                          "gte": [
                            "$date",
                            ":since"
                          ]
                        },
                        {
                          "lte": [
                            "$date",
                            ":till"
                          ]
                        }
                      ]
                    }
                  },
                  "params": {
                    "project": {
                      "caption": "Проект",
                      "type": "reference",
                      "className": "project@project-management"
                    },
```

(continues on next page)

(continued from previous page)

```
                    "since": {
                      "caption": "Период с",
                      "type": "date",
                      "default": "$monthStart"
                    },
                    "till": {
                      "caption": "Период по",
                      "type": "date",
                      "default": "$monthEnd"
                    }
                  },
                  "eagerLoading": [
                    "ownOrg",
                    "basicObjs"
                  ],
                  "preprocessor": "ion://pmListToDocx"
                }
              }
...
```

Configuring search parameters in the list of objects

The functionality allows you to determine at the class level whether we are looking for class objects from the list view by the first instance of a word or by full words, by individual attributes or by the specified attributes in the list with search parameters separated by spaces.

Format and available operations:

```
"listSearchOptions": {
    "person@khv-childzem": {...} // для класса
      "khv-childzem@person": {...} // только в узле навигации person
     "*": {...} // везде по умолчанию
}
```

Substitute ... with attributes and set operations for search, for example:

```
 "searchBy": [ // атрибуты по которым ищем, по умолчанию то, что выводится в колонках
  "surname",
  "name",
  "patronymic"
],
"splitBy": "\\s+", // разбивать поисковую фразу регуляркой, части сопоставить с атрибутами
"mode": ["starts", "starts", "starts"], // режимы сопоставления - в данном случае "начинается с" (доступны␣
↪like, contains, starts, ends)
"joinBy": "and" // режим объединения условий на атрибуты (по умолчанию or)
```

Configuring the hierarchical view for collections

Hierarchical view of collectionsdisplays collections, in which elements are connected in the form of a hierarchical reference book. The viewlib library has a custom controller that returns the next level of hierarchy in the TreeGrid format.

Example

```
"treegridController": {
          "module": "applications/viewlib/lib/controllers/api/treegrid",
          "initMethod": "init",
          "initLevel": 0,
          "options": {
            "module": "ion://module",
            "logger": "ion://sysLog",
            "securedDataRepo": "ion://securedDataRepo",
            "metaRepo": "ion://metaRepo",
            "auth": "ion://auth",
            "config": {
              "*": {
                "project@project-management":{
                  "roots":[{
                    "property": "name",
                    "operation": 1,
                    "value": [null],
                    "nestedConditions": []
                  }],
                  "childs":["stakeholders", "basicObjs"]
                },
                "governmentPower@project-management": {
                  "roots":[],
                  "childs":null,
                  "override": {
                    "descript": "url"
                  }
                },
                "object@project-management": {
                  "roots":[],
                  "childs":null
                },
                "event@project-management": {
                  "roots":[],
                  "childs":null
                }
              }
            }
          }
        }
...
```

The config field contains all settings:

- first key is the navigation node (the “*” sign means “applies to all nodes”),
- classes have roots, that indicate what objects of this class to pull out as root (“conditions” are used),
- ‘‘childs‘‘are class attributes to pull the hierarchy.

Configuring the text search in depth by reference attributes

searchByRefs is an array of settings to indicate the class hierarchy. You can compare it with several classes.

Example

```
"family@khv-childzem": {
        "*": {
          "searchByRefs":[
           {
             "class": "person@khv-childzem",
             "idProperties": ["famChilds", "famParentMale", "famParentFemale"],
             "searchBy": [
               "surname",
               "name",
               "patronymic"
             ],
             "splitBy": "\\s+",
             "mode": [
               "starts",
               "starts",
               "starts"
             ],
             "joinBy": "and"
           }
          ]
        }
       }
```

The “geomap” module

Geomap module is designed to visualize information and data that has geo-referenced. The data is divided into layers, and for each data type, you can provide brief and detailed information.

Setting the application icons

The logo for the module is described through the standard mechanism of static routes:

```
{
  "modules": {
    "geomap": {
      "statics":[{"path":"applications/khv-svyaz-info/icons", "name":"icons"}],
      "logo": "icons/logo.png"
    }
  }
}
```

Setting to hide header and sidebar

Example:

```
"geomap": {
  "globals": {
    "hidePageHead": true, //отобразить шапку
    "hidePageSidebar": false, //скрыть боковое меню
```

(continues on next page)

(continued from previous page)

```
    ...
  }
}
```

The gantt-chart module

Gantt-chart module is a module designed to display specific types of hierarchical data with dates. More details on gantt-chart module.

Setting the time scale

The time scale is configured by setting the “Step” in the Gaant module. In the preconfiguration “Step” is set through the stepparameter:

```
{
  "unit": "year",
  "step": 5
}
```

Example

```
...
  "gantt-chart": {
    "globals": {
      "config": {
...
        "preConfigurations": {
...
          "config3": {
            "caption": "Третья конфигурация",
            "showPlan": true,
            "units": "year",
            "step": 5,
            "days_mode": "full",
            "hours_mode": "full",
            "columnDisplay": {
              "text": true,
              "owner": true,
              "priority": true
            },
            "filters": {
              "priority": "Обычный"
            }
          }
        }
...
      }
    }
  }
```

The “report” module

Report module is a module designed to generate analytical reports and reference information based on special metadata. Расчеты могут выполняться по расписанию или быть инициированы оператором. Подробнее о модуле отчетов.

```
"report": {
    "globals": {
      "namespaces": {
        "project-management": "Проектное управление"
      },
      "defaultNav": {
        "namespace": "project-management",
        "mine": "projects",
        "report": "roadmap"
      },
      "mineBuilders": {
        "project-management": {
          "test": {
            "projects": "mineBuilder"
          },
          "projects": {
            "indicatorAll": "mineBuilder"
          }
        }
      },
      "di": {},
      "statics": {
        "common-static": "applications/project-management/templates/static"
      },
      "logo": "common-static/logo.png"
    },
    "import": {
      "src": "applications/project-management/bi",
      "namespace": "project-management"
    }
  }
```

The “rest” module

REST-services module is a module designed to implement integration in the REST format. Used to provide data for the portal module. More details on REST.

```
"rest": {
    "globals": {
      "di": {}
    }
  },
```

The “portal” module

Portal module is a module designed to display arbitrary data templates. Подробнее о модуле портала

```
"portal": {
    "import": {
      "src": "applications/project-management/portal",
      "namespace": "project-management"
    },
    "globals": {
      "portalName": "pm",
      "needAuth": true,
      "default": "index",
      "theme": "project-management/portal",
      "templates": [
        "applications/project-management/themes/portal/templates"
      ],
      "statics": {
        "pm": "applications/project-management/themes/portal/static"
      },
      "pageTemplates": {
        "navigation": {
          "index": "pages/index"
        }
      }
    }
  }
```

The “ionadmin” module

Administration module (ionadmin) is used for assigning permissions, managing scheduled jobs, and other administrative tasks. More details on administration module.

Hiding fields in the admin from assigning to a user

For roles that you want to hide in the admin from being assigned to the user, in the deploy of the application prescribe the filters based on the regular expressions, by which such roles will be determined.

```
"ionadmin": {
    "globals": {
      "securityParams": {
        "hiddenRoles": [
          "^somePrefix_"
        ]
      }
    }
  }
```

Settings of hiding module in system menu

Set the nullvalue in the module (in the deploy.jsonfile) that you would like to hide in the system menu of the project, for example "ionadmin": null.

```
{
  "namespace": "project-management",
  "parametrised": true, //
```

(continues on next page)

(continued from previous page)

```
  "globals": {
    "moduleTitles": {
      "registry": {
        "description": "Проектное управление",
        "order": 10,
        "skipModules": true
      }
      "ionadmin": null
    }
```

The “dashboard” module

**Dashboard module ** is a module designed to display brief information in the form of blocks. More details on dashboard module.

Set the following function in the "modules"section in the deploy.json file of the application, to load the data from the meta into the “dashboard” module:

```
"dashboard": {
  "globals": {
    "namespaces": {
      "project-management": "Проектное управление"
    },
    "root": {
      "project-management": "applications/project-management/dashboard"
    }
  }
}
```

The “diagram” module

```
"diagram": {
    "globals": {
      "config": {
        "org1": {
          "caption": "Организационная структура",
          "edit": true,
          "showSections": false,
          "relations": {
            "className": "organization@project-management",
            "title": "name",
            "text": "address",
            "img": "",
            "filter": [
              {
                "property": "headOrg",
                "operation": 0,
                "value": [
                  null
                ],
                "nestedConditions": []
              }
            ],
```

(continues on next page)

(continued from previous page)

```
            "children": [
              {
                "className": "branchOrg@project-management",
                "property": "branch",
                "title": "name",
                "text": "address",
                "children": [
                  {
                    "className": "branchOrg@project-management",
                    "property": "branch",
                    "children": []
                  }
                ]
              }
            ]
          }
        }
      }
    }
  }
```

Full file example deploy.json

The deploy.json file on the example of the “Project management system” application

```
{
  "namespace": "project-management",
  "parametrised": true, //
  "globals": {
    "moduleTitles": {
      "registry": {
        "description": "Проектное управление",
        "order": 10,
        "skipModules": true
      }
    },
    "explicitTopMenu": [
      {
        "id": "mytasks",
        "url": "/registry/project-management@indicatorValue.all",
        "caption": "Мои задачи"
      },
      {
        "type": "system",
        "name": "report"
      }
    ],
    "plugins": {
      "sessionHandler": {
        "options": {
          "storage": {
            "type": "[[session.type]]",
            "options": {
```

(continues on next page)

(continued from previous page)

```
          "host": "[[cache.redis.host]]",
          "port": "[[cache.redis.port]]"
        }
      }
    }
  },
  "wfEvents": {
    "module": "applications/project-management/lib/wfEvents",
    "initMethod": "init",
    "initLevel": 1,
    "options": {
      "workflows": "ion://workflows",
      "metaRepo": "ion://metaRepo",
      "dataRepo": "ion://dataRepo",
      "log": "ion://sysLog"
    }
  },
  "actualAclProvider": {
    "module": "core/impl/access/aclmongo",
    "initMethod": "init",
    "initLevel": 1,
    "options": {
      "dataSource": "ion://Db"
    }
  },
  "aclProvider": {
    "module": "core/impl/access/aclMetaMap",
    "options": {
      "dataRepo": "ion://dataRepo",
      "acl": "ion://actualAclProvider",
      "accessManager": "ion://roleAccessManager",
      "map": {
        "person@project-management": {
          "isEntry": true,
          "sidAttribute": "user",
          "jumps": [
            "employee"
          ]
        }
      }
    }
  },
  "fileStorage": {
    "module": "core/impl/resource/OwnCloudStorage",
    "options": {
      "url": "[[ownCloud.url]]",
      "login": "[[ownCloud.login]]",
      "password": "[[ownCloud.pwd]]"
    }
  },
  "dataRepo": {
    "options": {
      "maxEagerDepth": 4
    }
  },
  "customProfile": {
```

(continues on next page)

(continued from previous page)

```
    "module": "lib/plugins/customProfile",
    "initMethod": "inject",
    "options": {
      "auth": "ion://auth",
      "metaRepo": "ion://metaRepo",
      "dataRepo": "ion://dataRepo",
      "fields": {
        "piAct": {
          "caption": "Участник прогресс-индикатора",
          "required": false,
          "readonly": true,
          "type": 4
        }
      },
      "propertyMap": {
        "person@project-management": {
          "filter": "user",
          "properties": {
            "person": "id",
            "piAct": "piAct",
            "surname": "surname"
          }
        }
      }
    }
  },
  "securedDataRepo": {
    "options": {
      "accessManager": "ion://roleAccessManager",
      "roleMap": {
        "eventBasic@project-management": {
          "PROJECT_ADMIN": {
            "caption": "Администратор проекта",
            "resource": {
              "id": "pm::project-events"
            },
            "attribute": "project.administrator"
          },
          "PROJECT_RESPONSIBLE": {
            "caption": "Ответственный по проекту",
            "resource": {
              "id": "pm::project-events"
            },
            "sids": [
              "$project.owner"
            ]
          }
        }
      }
    }
  },
  "indicatorWfHandler": {
    "module": "applications/project-management/lib/util/indicatorWfHandler",
    "initMethod": "init",
    "initLevel": 2,
    "options": {
```

(continues on next page)

(continued from previous page)

```
      "workflows": "ion://workflows",
      "data": "ion://securedDataRepo",
      "log": "ion://sysLog"
    }
  },
  "auth": {
    "options": {
      "checkUrlAccess": [
        "/registry/project-management@project",
        "/portal"
      ]
    }
  }
},
"jobs": {
  "fact-creator": {
    "description": "Служба генератора фактический показателей",
    "launch": {
      "day": 1
    },
    "worker": "factCreator",
    "di": {
      "factCreator": {
        "executable": "applications/project-management/lib/fact-creator",
        "options": {
          "log": "ion://sysLog",
          "data": "ion://dataRepo",
          "workflows": "ion://workflows"
        }
      }
    }
  },
  "report-builder": {
    "description": "Служба сборки шахт данных модуля отчетов",
    "launch": {
      "hour": 24
    },
    "worker": "rebuilder",
    "di": {
      "reportMeta": {
        "module": "modules/report/lib/impl/DsReportMetaRepository",
        "initMethod": "init",
        "initLevel": 1,
        "options": {
          "dataSource": "ion://Db",
          "calc": "ion://calculator"
        }
      },
      "stdBuilder": {
        "module": "modules/report/lib/impl/StdMineBuilder",
        "options": {
          "dataSource": "ion://Db",
          "metaRepo": "ion://metaRepo",
          "dataRepo": "ion://dataRepo"
        }
      },
```

(continues on next page)

(continued from previous page)

```
        "rebuilder": {
          "executable": "modules/report/lib/rebuilder",
          "options": {
            "log": "ion://sysLog",
            "meta": "ion://reportMeta",
            "mineBuilders": {
              "project-management": {
                "projects": {
                  "indicatorAll": "ion://stdBuilder"
                }
              }
            }
          }
        }
      }
    }
  }
},
"deployer": "built-in",
"modules": {
  "registry": {
    "globals": {
      "signedClasses": [
        "indicatorBasic@project-management"
      ],
      "staticOptions": {
        "maxAge": 3600000
      },
      "explicitTopMenu": [
        "mytasks",
        {
          "type": "system",
          "name": "report"
        }
      ],
      "eagerLoading": {
        "*": {
          "briefcase@project-management": {
            "item": [
              "projects.typeProject.name"
            ],
            "list": [
              "projects.typeProject.name"
            ],
            "exportItem": [
              "direction.name"
            ],
            "exportList": [
              "result"
            ]
          }
        }
      },
      "listSearchMinLength": 3,
      "listSearchOptions": {
        "indicatorBasic@project-management": {
```

(continues on next page)

(continued from previous page)

```
      "*": {
        "searchBy": [
          "name",
          "objectBasic"
        ],
        "mode": [
          "starts",
          "starts"
        ],
        "joinBy": "and"
      }
    }
  },
  "storage": {
    "basicObj@project-management": {
      "cloudFile": "/${item.code} (${item.name})/",
      "resultCloudFile": "/${item.code} (${item.name})/"
    }
  },
  "defaultPath": "dashboard",
  "inlineForm": true,
  "navigation": {
    "namespaces": {
      "project-management": "Проектное управление"
    },
    "menus": {
      "top": [
        "project-management@mark"
      ]
    }
  },
  "templates": [
    "applications/project-management/templates/registry"
  ],
  "customTemplates": [
    {
      "node": "project-management@eventBasic",
      "classes": [
        {
          "name": "*",
          "types": {
            "create": "task/view",
            "item": "task/view",
            "selectClass": "task/selectClass"
          }
        }
      ]
    },
    {
      "node": "*",
      "classes": [
        {
          "name": "project@project-management",
          "types": {
            "item": "to-gantt-view",
            "selectClass": "task/selectClass"
```

(continues on next page)

(continued from previous page)

```
          }
        }
      ]
    }
  ],
  "statics": {
    "app-static": "applications/project-management/templates/registry/static",
    "app-vendor": "applications/project-management/themes/registry/static/vendor",
    "common-static": "applications/project-management/templates/static"
  },
  "logo": "common-static/logo.png",
  "di": {
    "pmItemToDocx": {
      "module": "modules/registry/export/itemToDocx",
      "initMethod": "init",
      "initLevel": 0,
      "options": {
        "tplDir": "applications/project-management/export/item",
        "injectors": []
      }
    },
    "pmListToDocx": {
      "module": "modules/registry/export/listToDocx",
      "initMethod": "init",
      "initLevel": 0,
      "options": {
        "tplDir": "applications/project-management/export/item2",
        "log": "ion://sysLog"
      }
    },
    "export": {
      "options": {
        "configs": {
          "project@project-management": {
            "passport": {
              "caption": "Паспорт проекта",
              "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
              "extension": "docx",
              "type": "item",
              "preprocessor": "ion://pmItemToDocx",
              "isBackground": true
            },
            "markResult": {
              "caption": "Оценка проектов",
              "mimeType": "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
              "extension": "docx",
              "type": "list",
              "query": {
                "filter": {
                  "and": [
                    {
                      "eq": [
                        "$guid",
                        ":project"
                      ]
                    }
```

(continues on next page)

(continued from previous page)

```
            ]
          }
        },
        "params": {
          "project": {
            "caption": "Проект",
            "type": "reference",
            "className": "project@project-management"
          }
        },
        "preprocessor": "ion://pmFromListToDocx",
        "isBackground": true
      }
    }
  }
},
"createIndicatorValueHandler": {
  "module": "applications/project-management/lib/actions/createIndicatorValueHandler",
  "initMethod": "init",
  "initLevel": 2,
  "options": {
    "data": "ion://securedDataRepo",
    "workflows": "ion://workflows",
    "log": "ion://sysLog",
    "changelogFactory": "ion://changelogFactory",
    "state": "onapp"
  }
},
"actions": {
  "options": {
    "actions": [
      {
        "code": "CREATE_INDICATOR_VALUE",
        "handler": "ion://createIndicatorValueHandler"
      }
    ]
  }
},
"digestData": {
  "module": "applications/project-management/lib/digest/digestData",
  "options": {
    "log": "ion://sysLog"
  }
},
"signManager": {
  "options": {
    "Preprocessor": "ion://digestData",
    "signaturePreprocessor": "ion://signSaver"
  }
},
"treegridController": {
  "module": "applications/viewlib-extra/lib/controllers/api/treegrid",
  "initMethod": "init",
  "initLevel": 0,
  "options": {
```

(continues on next page)

(continued from previous page)

```
            "module": "ion://module",
            "logger": "ion://sysLog",
            "dataRepo": "ion://securedDataRepo",
            "metaRepo": "ion://metaRepo",
            "auth": "ion://auth",
            "config": {
              "*": {
                "eventBasic@project-management": {
                  "roots": [
                    {
                      "property": "name",
                      "operation": 1,
                      "value": [
                        null
                      ],
                      "nestedConditions": []
                    }
                  ],
                  "childs": [
                    "basicObjs"
                  ]
                }
              }
            }
          }
        },
        "fileshareController": {
          "module": "applications/viewlib/lib/controllers/api/fileshare",
          "initMethod": "init",
          "initLevel": 0,
          "options": {
            "module": "ion://module",
            "fileStorage": "ion://fileStorage"
          }
        }
      },
      "dashboard": {
        "project-management": {
          "modules": {
            "dashboard": {}
          }
        }
      }
    }
  },
  "geomap": {
    "globals": {
      "ymapControls": {
        "loader": {
          "position": {
            "left": 15,
            "top": 90
          }
        },
        "rulerControl": null,
        "typeSelector": {
```

(continues on next page)

(continued from previous page)


```
        "float": "right"
      },
      "zoomControl": {
        "position": {
          "right": 10,
          "top": 10
        }
      }
    },
    "panels": {
      "rightInfo": {
        "type": "rightInfo"
      },
      "navFloat": {
        "type": "float",
        "cssClass": "map-nav-float nav-tree",
        "cssStyle": "left:10px; top:46px; width: 310px; max-height:calc(100% - 163px);"
      },
      "filterFloat": {
        "type": "float",
        "title": "Фильтры",
        "cssClass": "map-filter-float collapsible",
        "cssStyle": "left:10px; bottom:10px; width: 310px; max-height:calc(100% - 163px);"
      }
    },
    "hidePageHead": false,
    "hidePageSidebar": true,
    "stroke": {
      "panel": {
        "name": "filterFloat"
      },
      "path": {
        "strokeColor": "#00ff00",
        "strokeWidth": 6,
        "opacity": 0.8
      },
      "polygon": {
        "fillColor": "#00ff00",
        "fillOpacity": 0.1,
        "strokeColor": "#00ff00",
        "strokeOpacity": 0.9,
        "strokeWidth": 3
      }
    },
    "namespaces": {
      "project-management": "Геоданные проекта"
    },
    "templates": [
      "applications/project-management/templates"
    ],
    "statics": {
      "geoicons": "applications/project-management/icons"
    },
    "start": [
      135.07,
      48.48
```


(continues on next page)

(continued from previous page)


```
    ],
    "zoom": 10,
    "regions": {
      "enabled": true,
      "osmIds": [
        "151223"
      ],
      "panel": {
        "name": "filterFloat"
      },
      "button": {
        "caption": "Районы",
        "hint": "Фильтр по районам",
        "resetHint": "Сбросить фильтр"
      },
      "levels": {
        "4": {
          "strokeWidth": 3,
          "strokeColor": "#7e8dab",
          "strokeStyle": "solid",
          "strokeOpacity": 1,
          "fillColor": "#fffff",
          "fillOpacity": 0
        }
      }
    },
    "defaultNav": {
      "namespace": "project-management",
      "node": "objectBasic"
    },
    "search": {
      "panel": {
        "name": "filterFloat",
        "orderNumber": 10
      },
      "enabled": true,
      "timeout": 2000
    },
    "formFilter": {
      "panel": {
        "name": "filterFloat"
      }
    },
    "di": {
      "dataRepo": {
        "module": "core/impl/datarepository/ionDataRepository",
        "options": {
          "dataSource": "ion://Db",
          "metaRepository": "ion://metaRepo",
          "fileStorage": "ion://fileStorage",
          "imageStorage": "ion://imageStorage",
          "log": "ion://sysLog",
          "keyProvider": {
            "name": "keyProvider",
            "module": "core/impl/meta/keyProvider",
            "options": {
```


(continues on next page)

(continued from previous page)

```
            "metaRepo": "ion://metaRepo"
          }
        },
        "maxEagerDepth": 3
      }
    }
  }
 },
 "import": {
  "src": "applications/project-management/geo",
  "namespace": "project-management"
 }
},
"gantt-chart": {
 "globals": {
  "staticOptions": {
   "maxAge": 3600000
  },
  "config": {
   "columns": [
    {
     "name": "owner",
     "caption": "Владелец",
     "align": "center",
     "filter": true,
     "editor": {
      "type": "select2",
      "from": "employee@project-management"
     }
    }
   ],
   "preConfigurations": {
    "config2": {
     "caption": "Расширенная",
     "showPlan": false,
     "units": "year",
     "days_mode": "full",
     "hours_mode": "work",
     "columnDisplay": {
      "text": true,
      "owner": true,
      "priority": true,
      "start": true,
      "progress": true
     }
    }
   },
   "roots": [
    "briefcase@project-management",
    "project@project-management"
   ],
   "initialDepth": 1,
   "createUrl": {
    "project@project-management": "registry/project-management@myprojectevent.all/new/{{parentClass}}.
↪{{parentId}}/basicObjs/event@project-management"
   },
```

(continues on next page)

(continued from previous page)


```
        "searchCount": 25,
        "inplaceCreation": {
          "rootLevel": true,
          "skip": [
            "briefcase@project-management"
          ],
          "ambigiousDefault": "event@project-management",
          "force": {
            "@root": "briefcase@project-management",
            "eventObject@project-management": "eventOnly@project-management"
          }
        },
        "map": {
          "employee@project-management": {
            "eager": [
              "person",
              "organization"
            ]
          },
          "project@project-management": {
            "type": "project",
            "open": true,
            "color": "#e3fcef",
            "textColor": "#000",
            "text": "name",
            "override": {
              "owner": "head"
            },
            "parents": [
              "briefcase"
            ],
            "filter": {
              "ne": [
                "$archive",
                true
              ]
            },
            "url": "registry/project-management@myprojectevent.all/view/:class/:id"
          }
        }
      },
      "statics": {
        "common-static": "applications/project-management/templates/static"
      },
      "logo": "common-static/logo.png",
      "rootParamNeeded": true
    }
  },
  "report": {
    "globals": {
      "namespaces": {
        "project-management": "Проектное управление"
      },
      "defaultNav": {
        "namespace": "project-management",
        "mine": "projects",
```


(continues on next page)

(continued from previous page)

```
      "report": "roadmap"
    },
    "mineBuilders": {
      "project-management": {
        "test": {
          "projects": "mineBuilder"
        },
        "projects": {
          "indicatorAll": "mineBuilder"
        }
      }
    },
    "di": {},
    "statics": {
      "common-static": "applications/project-management/templates/static"
    },
    "logo": "common-static/logo.png"
  },
  "import": {
    "src": "applications/project-management/bi",
    "namespace": "project-management"
  }
},
"rest": {
  "globals": {
    "di": {}
  }
},
"portal": {
  "import": {
    "src": "applications/project-management/portal",
    "namespace": "project-management"
  },
  "globals": {
    "portalName": "pm",
    "needAuth": true,
    "default": "index",
    "theme": "project-management/portal",
    "templates": [
      "applications/project-management/themes/portal/templates"
    ],
    "statics": {
      "pm": "applications/project-management/themes/portal/static"
    },
    "pageTemplates": {
      "navigation": {
        "index": "pages/index"
      }
    }
  }
},
"ionadmin": {
  "globals": {
    "defaultPath": "ionadmin",
    "securityParams": {
      "resourceTypes": {
```

(continues on next page)

(continued from previous page)


```
        "*": {
          "title": "Общие"
        }
      },
      "hiddenRoles": [
        "^PROJ_DEPART_EMPLOYEE"
      ]
    }
  },
  "statics": {
    "common-static": "applications/project-management/templates/static"
  },
  "logo": "common-static/logo.png"
},
"dashboard": {
  "globals": {
    "namespaces": {
      "project-management": "Проектное управление"
    },
    "root": {
      "project-management": "applications/project-management/dashboard"
    }
  }
},
"diagram": {
  "globals": {
    "config": {
      "org1": {
        "caption": "Организационная структура",
        "edit": true,
        "showSections": false,
        "relations": {
          "className": "organization@project-management",
          "title": "name",
          "text": "address",
          "img": "",
          "filter": [
            {
              "property": "headOrg",
              "operation": 0,
              "value": [
                null
              ],
              "nestedConditions": []
            }
          ],
          "children": [
            {
              "className": "branchOrg@project-management",
              "property": "branch",
              "title": "name",
              "text": "address",
              "children": [
                {
                  "className": "branchOrg@project-management",
                  "property": "branch",
```


(continues on next page)

(continued from previous page)

```
                "children": []
              }
            ]
          }
        ]
      }
    }
  }
 }
}
```

Configuration file deploy.json is a file that describes the structure of the parameters of the software system and its specific configuration.

There is also a method to set the configuration through many separate files: :doc: deploy build from separate files <deploy/deploy_pulling>

Structure of the deploy.json file:

Field	Name	Description
"namespace"	Project name	The project namespace.
"parametrisedParameterization": true,	Parameterization	The parameterization settings. Set the "true" value to specify the parameters to which the system transfers the variables defined in ini-files or variables of the project environment when building the application.
"globals": {	Global settings	Global configuration settings.
"deployer": "built-in"	Builds	The built configuration parameter. Currently, it is the only one.
"modules"	Module settings	Module configuration settings.

File example deploy.json

3.5.4 Dependencies in package.json

The package.json file defines the structure of dependencies and the detailed composition of the system modules.

```
"ionMetaDependencies": {
  "viewlib": "0.0.1"
}
```

Connection using a script

- if there is no slash in the object name - / => “project-management”, substitute the name in the default path the ION-APP group, i.e the path is //git.iondv.ru/ION-APP/project-management.
- if there is a slash in the object name, it means it’s already set up with the group and just pick up the path to the git with the group and meta, for example for “ION-METADATA/viewlib” the path is //git.iondv.ru/ION-METADATA/viewlib.
- if the version value begins with git+http:// или git+https://, then this is the full path to the external repository. Drop git+ and pull the git.
- if the version starts with http:// or https://, then this is the full path to the archive. Pull and unpack.

  **Not implemented **, since dapp does not support working with archives.

Example of the package.json file

```
{
  "name": "develop-and-test",
  "description": "Метапроект для тестирования и разработки",
  "version": "1.9.2",
  "homepage": "http://docker.local:8080",

  "engines": {
    "ion": "1.24.1"
  },
  "scripts": {
    "test": "mocha ./test/e2e/**/*.js"
  },
  "ionModulesDependencies": {
    "registry": "1.27.1",
    "geomap": "1.5.0",
    "portal": "1.3.0",
    "report": "1.9.2",
    "ionadmin": "1.4.0",
    "dashboard": "1.1.0",
    "rest": "1.1.2",
    "gantt-chart": "0.8.0"
  },
  "ionMetaDependencies": {
    "viewlib": "0.9.1"
  }
}
```

Field description

Field	Name	Description
"name"	Name	Project name.
"description"	Description	Project description.
"version"	Version	Number of a current version.
"homepage"	Home page	Link to the built project on the docker.
"bugs"	Bugs	The link to the application project in GitLab, where issues about bugs are collected, is specified.
"repository"	Repository	Consists of "type" and "url" fields. Indicates the type of repository and a link to it.
"engines"	Core	Number of a core version.
"scripts"	Scripts	Script to build meta from different groups and different url.
"ionModuleDependencies"	Dependencies of ion modules	Specifies the modules and their versions used in the application. The project includes the following modules: • "ionadmin" – administration module • "registry" – register module • "report" – reports module • "rest" - rest services module • "dashboard" - dashboards module • "geomap" – geomodule • "gantt - chart" -charts module gantt • "portal" – portal module
"ionMetaDependencies"	Dependencies of ion metadata	Additional applications to operate the system.
"dependencies"	Dependencies	Other project dependencies from the npm repository.

3.5.5 Operations with MongoDB accounts via the CLI

Authorization

For authorization when interacting with the DBMS via the console interface, the parameters are passed to mongo

1. --authenticationDatabase <the database used for authorization>
2. -u <user name>
3. -p <password>

example:

```
mongo --authenticationDatabase admin -u admin -p 123
```

Create a user

To create a user, you must have the dbOwner role, or the UserAdmin role in the database that is used for authorization (for example, admin), and in all the databases where roles are added, or the createUser command privilege in this database and the grantRole command privileges in databases where roles are added.

A user is created through the console interface by the command

```
mongo --eval "(new Mongo()).getDB(<имя бд, куда писать данные авторизации>').createUser( \
{ \
  user: '<пользователь>', \
  pwd: '<пароль>', \
  roles: [ \
    { role: 'readWrite', db: '<бд куда доступ на чтение-запись>' }, \
    { role: 'read', db: '<бд где только чтение>' }, \
    { role: 'write', db: '<бд где только запись>' } \
  ]})"
```

example:

```
mongo --eval "(new Mongo()).getDB('admin').createUser( \
{ \
  user: 'admin', \
  pwd: '123', \
  roles: [ \
    { role: 'readWrite', db: 'admin' }, \
    { role: 'readWrite', db: 'config' }, \
    { role: 'readWrite', db: 'local' } \
  ]})"
```

or in one line

```
mongo --eval "(new Mongo()).getDB('admin').createUser({user: 'demo',pwd: '123',roles: [{ role: 'readWrite',␣
↪db: 'admin' },{ role: 'readWrite', db: 'config' },{ role: 'readWrite', db: 'local' }]})"
```

Delete a user

To delete a user, you must have the dbOwner role, or have the UserAdmin role or the dropUser command privilege in the database that is used for authorization, for example, admin.

Delete a user through the CLI mongodb with the command

```
mongo --eval "(new Mongo()).getDB('<бд с данными авторизации>').dropUser('<имя>')"
```

example:

```
mongo --eval "(new Mongo()).getDB('admin').dropUser('demo')"
```

ГЛАВА 4

4. Modules

4.1 Registry Module

4.1.1 The logics of forming ID elements, associated with objects by type

Implemented features

INPUT, SELECT, TEXTAREA

1. for editing the attribute of the object “ a_ {namespace, class, attribute}“

```
<input type="text" class="form-control" id="a_develop-and-test_class_integer_integer_integer" name=
↪"integer_integer" pattern="[0-9]+([\.|,][0-9]+)?" value="5120">
```

BUTTON

1. list action la_{namespace, class, command name}

```
<button id="la_develop-and-test_class_integer_edit" class="edit btn btn-info command" title="Редактировать
↪" data-id="EDIT" style="display: inline-block;">Править</button>
```

2. form action fa_{namespace, class, command name}

```
<button id="fa_develop-and-test_class_integer_saveandclose" data-id="SAVEANDCLOSE" type="button"␣
↪class="btn command object-control SAVEANDCLOSE" style="">
      Сохранить и Закрыть
    </button>
```

3. reference field action ra_{namespace, class, attribute, command name}

```
<button id="ra_develop-and-test_ref_use_ref_use_create" type="button" class="create-btn btn btn-success"
↪data-ref-property="ref_use" title="Создать">
                <span class="glyphicon glyphicon-plus-sign"></span>
            </button>
```

4. collection field action ca_{namespace, class, attribute, command name} no opportunity to check collections in progress

5. floating form buttons ffa_{namespace, class, attribute, command name}

```
<button id="ffa_develop-and-test_class_integer_close" type="button" class="btn btn-default CLOSE" title=
↪"Закрыть" data-cmd="CLOSE">
    <span class="glyphicon glyphicon-remove"></span>
  </button>
```

FORM, DIV, TR, LI (for objects view)

1. navigation sections and nodes n_{navigation section name}

```
<a id="n_simple_types" href="#" title="Простые типы">
  <i class="fa fa-menu-arrow pull-right toggler"></i>
  <i class="main-icon fa fa-institution" title="Простые типы"></i>
  <span>Простые типы</span>
 </a>
```

Navigation node n_{navigation node code}.

```
<a id="n_class_integer" class="menu-link" href="/registry/develop-and-test@class_integer" title="Класс &
↪quot;Целое [6]">Класс "Целое [6]</a>
```

Open class navigation node.

4.1.2 DI settings

Connect register module in the global settings

Example in deploy.json

```
"modules": {
  "registry": {
    "globals": {
   "di": {
```

treegridController

Description

Is for creating hierarchical lists of objects in the class collection attribute or in the class navigation.

Operates using the component dhtmlxSuite_v51_pro (https://dhtmlx.com/docs/products/dhtmlxTreeGrid/).

Connection in DI

```
"treegridController": {
        "module": "applications/viewlib/lib/controllers/api/treegrid",
        "initMethod": "init",
        "initLevel": 0,
        "options": {
          "module": "ion://module",
          "logger": "ion://sysLog",
          "securedDataRepo": "ion://securedDataRepo",
          "metaRepo": "ion://metaRepo",
          "auth": "ion://auth",
          "config": { // основной конфиг
            "*": { // выборка объектов возможна в каждой навигации
              "eventBasic@project-management":{ // выборка объектов по указанному классу
                "roots":[{ // поиск корней
                  "property": "name",
                  "operation": 1,
                  "value": [null],
                  "nestedConditions": []
                }],
                "childs":["basicObjs"] // поиск дочерних элементов
              },
```

Template types

1) "template": "treegrid/collection"

For a collection-attribute. Connected in the object form view:

```
"options": {
        "template": "treegrid/collection",
        "reorderable": true,
        "treegrid": {
          "width": "auto,100,100,100,100,0",
          "align": "left, center,center,center,center, left",
          "sort": "str, date, date, date, date, int",
          "enableAutoWidth": false,
          "paging": {
            "size": 20
          }
        }
      }
```

2) "template": "treegrid/list"

For class navigation. Connection:

```
"options": {
   "template": "treegrid/list"
 }
```

3) Setting up skin

https://docs.dhtmlx.com/grid__skins.html

```
"options" : {
...
  "treegrid" : {
    "skin": "material" // по умолчанию
    // "skin": "skyblue"
    // "skin": "terrace"
    // "skin": "web"
  }
}
```

Additional sources of information on treegridController

- Hierarchical view for collections.

DHTMLX (dhtmlxSuite_v51_pro)

- https://docs.dhtmlx.com/
- https://dhtmlx.com/docs/products/dhtmlxTreeGrid/

Registry module is a key module for data operation based on the metadata structures (for example for project management, events, programms etc.).

4.1.3 Configuration

- DI (treegridController)

Deploy

Setting requests frequency in deploy.json

Setting the server requests frequency to ensure that the object is not blocked in deploy.json:

```
"registry": {
  "globals": {
    "notificationCheckInterval": 60000 // раз в минуту
  }
}
```

Setting “createByCopy”

Configuring the display of the “Create more” button in deploy.json:

```
"createByCopy": [
        "person@khv-childzem" // класс
      ],
```

Filters

More details on filters here.

Setting help on filters

“Help” is placed in the template file - modules/registry/view/default/templates/view/list-filter-helper.ejs

4.1.4 Code requirements

Front-end components style here.

4.2 Report Module

4.2.1 Comments on the mine design

“NOT CONTAINS” operations

When using MongoDB ‘‘NOT CONTAINS‘‘operations will not work correctly, since NOT is applied not to CONTAINS, but to the entry conditions.

At the moment, it is not possible to change this.

The data source builder is not configured

The builder for the data source "summaryArea.serviceGrid" is not configured. In addition to the meta itself in ‘‘ deploy.json‘‘, the following must be added to modules.report.globals:

```
"mineBuilders": {
   "khv-svyaz-info": {
     "summaryArea": {
       "internet": "mineBuilder",
       "population": "mineBuilder",
       "internetGrid": "mineBuilder",
       "station": "mineBuilder",
       "serviceGrid": "mineBuilder"
     }
   }
 }
```

This is the binding of builders to data sources in order to be able to aggregate from different databases (and sources in general). For now, the standard mineBuilder is used, which uses the local Db datasource.

Cannot compare two fields

The compariing two fields using search expressions will not work. Mongo DB can’t compare two fields. For example, the following expressions will not work:

```
{"attr1": {"$regex": "$attr2", "$options": "i"}}
```

Filters

For sources based on classes, filters are specified by conditions.

```
"filter": [
          {
            "property": "typePet",
            "operation": 0,
            "value": "statement",
            "nestedConditions": []
          }
        ]
```

Report module is a module for generating analytical reports and reference information in the form of graphs based on special metadata. Calculations can be scheduled or initiated by the operator.

4.2.2 Library to build the Pivot type reports

The library for building Pivot reports is PivotTable.js (Examples and description: https://pivottable.js.org) The functionality is rich, but at the same time complex for building excel or word level reports. At the moment, when designing a report, it is better to consult with the developers and put a task on the implementation of a Pivot report, if the expected functionality of the report is relatively complex.

4.2.3 Metadata

Report module metadata (mine, navigation)

Notes on mine design

4.2.4 Automatic data mine building

Add the jobs.enabled = true setting to the config.ini file to set up automatic data mine building.

Example

To run a job at the start of the application and every 6 hours after that, you need to configure the job as follows:

```
"jobs": {
    "report-builder": {
      "description": "Служба сборки шахт данных модуля отчетов",
      "launch": {
        "hour": 21600000
      }
    }
```

4.3 Gantt-chart Module

Gantt-chart module is a module for output of specific types of hierarchical data that have dates.

4.3.1 Configuration in deploy.json

Specifying the length for the search output

```
"searchCount": 25
```

Adding filters to the columns

```
"gantt-chart": {
    "globals": {
     "config": {
       "columns": [
         {
           "name": "text",
           "caption": "Название"
         },
         {
           "name": "owner",
           "caption": "Владелец",
           "align": "center",
           "filter": true
         },
         {
           "name": "priority",
           "caption": "Приоритет",
           "align": "center",
           "filter": true
         },
```

Specifying of diffrent “createUrl” for different classes

```
"createUrl": {
        "project@project-management": "registry/project-management@eventBasic/new/eventBasic",
        "event@project-management": "registry/project-management@eventBasic/new/eventBasic",
        "eventObject@project-management": "registry/project-management@eventOnly/new/eventOnly",
        "eventOnly@project-management": "registry/project-management@eventOnly/new/eventOnly",
        "projectKNA704@project-management": "registry/project-management@eventKNA704/new/
↪eventKNA704",
        "eventKNA704@project-management": "registry/project-management@eventOnlyKNA704/new/
↪eventOnlyKNA704",
        "eventObjectKNA704@project-management": "registry/project-management@eventOnlyKNA704/new/
↪eventOnlyKNA704",
        "eventOnlyKNA704@project-management": "registry/project-management@eventOnlyKNA704/new/
↪eventOnlyKNA704"
      }
```

4.3.2 Configuration of view types

```
"preConfigurations": {
        "config1": {
          "caption": "Основная",
```

(continues on next page)

(continued from previous page)

```
        "showPlan": true,
        "units": "month",
        "step": 3,
        "days_mode": "full",
        "hours_mode": "full",
        "default": true
      },
      "config2": {
        "caption": "Расширенная",
        "showPlan": false,
        "units": "year",
        "days_mode": "full",
        "hours_mode": "work",
        "columnDisplay": {
          "text": true,
          "owner": true
        },
        "filters": {
          "priority": "Высокий"
        }
      },
      "config3": {
        "caption": "Обзорная",
        "showPlan": true,
        "units": "year",
        "step": 5,
        "days_mode": "full",
        "hours_mode": "full",
        "columnDisplay": {
          "text": true,
          "owner": true,
          "priority": true
        },
        "filters": {
          "priority": "Обычный"
        }
      }
    }
```

Set the property and values for the filter in the filters field.

Adjustable filter when selecting subnodes

In formulas in the general syntax of expressions, you can now access the context data. Currently it's only implemented for lists in registry and gant. As we move to a common syntax, we'll implement the support everywhere in the core.

The adjustable filter is not applied to the root object, which is explicitly specified via the URL parameter or selected in the drop-down list. The filter is applied only when SELECTING the SUBNODES.

Sorting the output

When displaying the project, the events are sorted by the numEvent attribute at all levels of the hierarchy.

```
"sortBy": "numEvent"

// либо
"sortBy": {"numEvent": -1, "anyOtherAttr": 1}
```

Configuring an object selection list for information output

Applied in case when the filter is set for the column, and objects are not displayed all at once, but can be chosen from a list. If value ‘‘”rootParamNeeded:true”‘‘is specified, an empty screen will be displayed and a window for choosing the project.

```
"gantt-chart": {
    "globals": {
      "rootParamNeeded": true
    }
  }
```

4.4 Portal Module

Portal Module is a module for displaying arbitrary data templates. The Portal module displays the design of different information using Markdown and HTML markup languages.

4.4.1 Structure

View layer - is responsible for displaying data:

1. Adapter Provider - provides connection between the data and their display on the portal (configuration in the ‘deploy.json’ file).
2. File Adapter - returns the resources of “File” type (remains in the memory of the application).
3. Class Adapter - designed to display data from the database through the data repository (updates each time).

4.4.2 The logic of displaying data on the portal

Portal styles

CSS (cascading style sheets) are used to describe/design the portal pages. An example of the portal page design in Dnt - /develop-and-test/portal/view/static/css/style.css

Possible options for applying styles are set in this folder - portal/view/static/style.css.

```
.navbar {
  border-radius: 0;
  margin-bottom: 0;
}

.navbar .navbar-brand {
```

(continues on next page)

(continued from previous page)

```
  padding: 5px;
}
...
```

Portal pages

Portal page templates are configured in the portal/view/templates folder.

The location of objects on the page is described in HTML markup language:

```
<html>
<head>
  <title><%= portal %></title>
  <link href="/<%= module %>/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
  <link href="/<%= module %>/dnt/css/style.css" rel="stylesheet">
</head>
<body>
  <%- partial('../parts/menu', { menu }) %>
  <%- body -%>
  <script src="/<%= module %>/vendor/jquery/jquery.min.js"></script>
  <script src="/<%= module %>/vendor/bootstrap/js/bootstrap.min.js"></script>
  <script src="/<%= module %>/js/scripts.js"></script>
</body>
</html>
```

- The <head> tag contains styles for the appearance of the portal.
- The <body> tag contains information about the objects displayed on the portal page.
- The <script> tag contains information in the form of a link to a program or its text in a specific language. Scripts can be placed in an external file and linked to any HTML document, which allows you to use the same functions on several web pages, thereby speeding up their loading. In this case, the <script> tag is used with the srcattribute, which points to the address of the script from an external file to import into the current document.

Portal navigation

A portal navigation meta is a set of navigation nodes, and each has a resource type specified.

Example of the navigation section creating:

```
{
  "code": "main",
  "caption": "Главное меню",
  "itemType": "section",
  "subNodes":["classes", "texts"]
}
```

- code - system object name
- caption - logical object name
- itemType - object display type
- subNodes - an array of navigation nodes of this section

An example of creating a navigation node:

```
{
  "code": "texts",
  "caption": "Публикация текстов",
  "resources": "texts",
  "PageSize": 5,
  "itemType": "node"
}
```

- code - system object name
- caption - logical object name
- resources - turning data into a portal content
- PageSize - page size
- itemType - object display type

Data styling

1. The format of paging information

```
<% layout('./layout/content') %>
<%
if (Array.isArray(resources) && resources.length) {
 resources.forEach(function(resource){
%>
<div>
<h3 id="<%= node.code %>_<%= resource.getId() %>">
  <a href="/<%= module %>/<%= node.code %>/<%= resource.getId() %>">
    <%= resource.getTitle() %>
  </a>
  <%
    var formatedDate = null;
    var date = resource.getDate();
    if (date) {
      formatedDate = date.toLocaleString('ru',{year: 'numeric', month: 'numeric', day: 'numeric'});
    }
    %>
  <% if (formatedDate) { %><small><%= formatedDate %></small><% } %>
</h3>
<p><%- resource.getContent() %></p>
</div>
<%
 })
}
%>
<%- partial('./parts/pagination', { resources }) %>
```

2. The format of the correct displaying of the text of errors

```
<% layout('./layout/layout') %>
<div class="container">
  <h1>404</h1>
  <h2>Страница не найдена</h2>
</div>
```

3. The format of converting data to portal content

```
<% layout('./layout/layout') %>

<div class="container">

  <div class="row">
    <div class="col-md-12">
      <div class="page-header">
        <h2><%= resource.getTitle() %></h2>
      </div>
      <div>
        <%
        var formatedDate = null;
        var date = resource.getDate();
        if (date) {
          formatedDate = date.toLocaleString('ru',{year: 'numeric', month: 'numeric', day: 'numeric'});
        }
        %>
        <% if (formatedDate) { %><h1><small><%= formatedDate %></small></h1><% } %>
      </div>
      <div>
        <%- resource.getContent() %>
      </div>
    </div>
  </div>

</div>
```

4. The format of the text display

```
<% layout('./layout/layout') %>

<div class="container">

  <div class="row">
    <div class="col-md-12">
      <div>
        <%
        var formatedDate = null;
        var date = resource.getDate();
        if (date) {
          formatedDate = date.toLocaleString('ru',{year: 'numeric', month: 'numeric', day: 'numeric'});
        }
        %>
        <% if (formatedDate) { %><h1><small><%= formatedDate %></small></h1><% } %>
      </div>
      <div>
        <%- resource.getContent() %>
      </div>
    </div>
  </div>

</div>
```

4.5 Dashboard Module

Dashboard module is a module for displaying brief information as blocks. Based on the widget model.

4.5.1 Module structure

The Dashboard consists of the three basic components: manager, layout и widget.

Manager

Manager is a main component of the module for creating and initializing widgets and layouts and connecting the dashboard to other modules.

```
let manager = require('modules/dashboard/manager');
```

Layout

Layout is a EJS template, which defines the layout of widgets, parameters of widget templates, a plugin for managing layout grid on the client (for example gridster), and connects common resources.

Basic module layouts are located in the folder /dashboard/layouts. The published ones from the metadata are located in the folder /applications/${meta-namespace}/layouts.

Each layout has a unique ID. When publishing from meta, a prefix is added to the ID.

```
let dashboard = require('modules/dashboard');
dashboard.getLayout('demo');
dashboard.getLayout('develop-and-test-demo');
```

When rendering a layout, you must pass an object to manager.

```
res.render(dashboard.getLayout('demo'), { dashboard });
```

Widget

Widget is an object which is located on the layout and interacts with the server through the ajax-requests.

Basic widgets are located in the folder /dashboard/widgets. The published ones from the metadata are located in the folder /applications/${meta-namespace}/widgets.

A widget contains the the index.js file class and the view.ejs view template. The class must be inherited from the base class /dashboard/base-widget or its descendants.

- The init() method is responsible for the widget initialization when the server starts.
- The refresh() method is called when receiving an ajax request from a client.
- The job() method gets the data for the widget.

Each widget has a unique ID. When publishing a widget from meta, a prefix is added to the ID.

```
dashboard.getWidget('demo');
dashboard.getWidget('develop-and-test-demo');
```

When rendering a widget view, you must pass an object to the widget.

```
<% var widget = dashboard.getWidget('develop-and-test-demo') %>
<%- partial(widget.view, {widget}) %>
```

4.5.2 Publishing from the meta

Example of the structure in applications/develop-and-test:

```
dashboard
    layouts
      demo-layout
    widgets
      demo-widget
        index.js
        view.ejs
    static
        layouts
        widgets
          demo-widget
```

If you want that the data from meta loaded to the dashboard modules, add the following snippet to the "modules"section of the ‘‘deploy.json‘‘configuration file of the application:

```
"dashboard": {
  "globals": {
    "namespaces": {
      "develop-and-test": "Мета для тестирования и разработки"
    },
    "root": {
      "develop-and-test": "applications/develop-and-test/dashboard"
    }
  }
}
```

4.6 Ionadmin module

4.6.1 Security settings

Initialization

When configuring security for the first time, do the following:

1) Synchronize the rights so that the rights set via the acl utility appear in the admin area. Press the "Synchronization of access rights" button on the /ionadmin/security/sync page to synchronize the rights.

Upon completion you'll see the message "Access synchronization successfully conducted!".

2) Make import of resources. If on the /ionadmin/security/resource page there are no objects or really few of them, then you need to import the resources.

Press the "Import resources" button on the /ionadmin/security/sync page.

Upon completion you'll see the message "Import successfully completed!".

Role management

You can create, edit or delete roles on the /ionadmin/security/role page:

1) Create role. Press the “Create” button to create a role on the /ionadmin/security/role page.

You will be redirected to a new page, where you need to specify the role ID. Click the “Save” button to confirm the creation of the specified role.

2) Edit role. Choose the desired role and press the “Edit” button on the /ionadmin/security/role page to edit the role.

You will be redirected to a new page, where the following fields are displayed on the form:

‘‘Identifier‘‘is an expression for the unique name of the role. When changing the identifier, the rights of users who tied this role to the previous identifier may be lost. In this case it’s necessary to reconnect the role to each user.

Name is a role’s name that can contain the exressions in Russian.

Access rights are tabs for rights distribution:

- Common - for distribution of a role of access to all the resources (* - all resources)
- Navigation - for distribution of a role to access the menu of the registry module. First, the system name of the project is displayed, which has a plus sign to display internal and imported resources. At this point only the rights to read the menu are distributed.
- Classes - is for distribution of a role to access to metadata classes. First, the system name of the project is displayed, which has a plus sign to display internal and imported resources. You can set separate rights for these resources.

3) Delete role. To delete a role select the required role on the page /ionadmin/security/role and click the “Delete” button.

Confirm the deletion of the role.

Resource access

When managing roles in access rights, the following access is provided to each resource:

Access	Description
Full access	Includes all accesse types. You cannot select full access and additional access to read, edit, delete or use. For navigation, only the read access is granted.
Read	The ability to read resource objects
Write	The ability to edit resource objects, not used for navigation
Delete	The ability to delete resource objects, not used for navigation
Usage	The ability to create resource objects, not used for navigation

Access can be assigned to the entire resource group or separately to each resource and access in it.

User management

An the page / ionadmin / security / user you can create, edit or delete users:

1) Create user. Press the “Create” button on the /ionadmin/security/user page to create a user.

You will be redirected to a new page, where you need to specify the username, password, description in the name. Click the “Save” button to confirm the creation of the specified user.

2) Edit user. Select the required user at the page / ionadmin / security / user and press the “Edit” button to edit the user.

You will be redirected to a new page, where the following fields are displayed on the form:

Type - type of user account, only local users are available in the admin panel.

Login - user ID.

Password - user password.

Name - user’s name which may contain expressions in Russian, for example, full name.

Roles - a list of user roles. If the role is ticked, the role is tied to the user.

3) Delete user. Choose the user at the /ionadmin/security/user page and press the “Delete” button to delete the user.

Confirm the deletion of the user.

User authentication settings. Password requirements.

The password requirements are set in the ini-file of the application, after which the variables must be declared in the application configuration file *deploy.json *.

In the file setup.ini:

```
auth.passwordLifeTime=90d # Максимальный срок действия пароля
auth.passwordMinPeriod=75d # Минимальный срок действия пароля
auth.passwordMinLength=8 # Минимальная длина пароля
auth.passwordJournalSize=5 # Число уникальных новых паролей пользователя до повторного использования
↪старого пароля
auth.tempBlockInterval=30m # Время до сброса счетчика блокировки
auth.attemptLimit=6 # Количество неудачных попыток входа в систему, приводящее к блокировке учетной
↪записи пользователя
auth.tempBlockPeriod=30m # Продолжительность блокировки учетной записи
auth.sessionLifeTime=4h # Время жизни авторизованной сессии, при отсутствии активности
```

The duration is set in the format: [duration][unit]

Unit	Value
y	Year
d	Day
h	Hour
m	Minute
s	Second

In the file deploy.json:

NB: Нужно обязательно, чтобы стояла настройка "parametrised": true, на уровне "global"

```
{
  "parametrised": true,
  "globals": {
    "plugins":{
      "auth": {
        "module": "lib/auth",
        "initMethod": "init",
```

(continues on next page)

(continued from previous page)

```
      "initLevel": 2,
      "options": {
        "app": "ion://application",
        "logger": "ion://sysLog",
        "dataSource": "ion://Db",
        "acl": "ion://aclProvider",
        "passwordLifetime": "[[auth.passwordLifeTime]]", // максимальный срок действия пароля
        "passwordMinPeriod": "[[auth.passwordMinPeriod]]", // минимальный срок действия пароля
        "passwordMinLength": "[[auth.passwordMinLength]]", // минимальная длина пароля
        "passwordComplexity": { // требования к сложности пароля
          "upperLower": true, // требование использовать буквы в разном регистре
          "number": true, // требование использовать числа
          "special": true // требование использовать спецсимволы
        },
        "passwordJournalSize": "[[auth.passwordJournalSize]]", // ведение журнала паролей
        "tempBlockInterval": "[[auth.tempBlockInterval]]", // счетчик блокировки
        "attemptLimit": "[[auth.attemptLimit]]", // пороговое значение блокировки
        "tempBlockPeriod": "[[auth.tempBlockPeriod]]" // продолжительность блокировки
      }
    }
  }
}
```

Administration module (Ionadmin) – is used for assigning rights, managing scheduled jobs and other administrative tasks.

4.6.2 Configure the Ionadmin module in the config.json file

Configure slow query recording

Setting up as a modal window in the list of slow queries. Set the source in the config.json file of the ionadmin module:

```
"profiling": {
  "slowQuery": {
    "sources": [
      {
        "collection": "system.profile"
      }
    ]
  }
}
```

If the "sources" property is not set or null, the data will be taken from the table:

```
{
  "profiling": {
    "slowQuery": {
      "sources": null
    }
  }
}
```

If an empty array is set, then there are no sources.

Log sources configuration

Log sources (can be a few) are specified in the config.json file of the module:

```
"profiling": {
   "slowQuery": {
     "sources": [
       {
         "collection": "system.profile"
       },
       {
         "file": "D:/Temp/slow-query.txt"
       }
     ]
   }
 }
```

The selections made are stored in a separate table and do not depend on the current state of the log sources. They can be supplemented by editing. For example, with comments or notes reporting whether the problem is solved.

DB backup configuration

Settings in the ionmodule/config:

```
"backup": {
   "dir": "../ion-backups",
   "zlib": {
     "level": 1
   }
 }
```

- dir contains the path to the folder, where the node application is launched. By default "../ion-backups"
- zlib.level - the level of compression also affects the speed of the archive creation. By default - level 3.
- In addition, it is necessary that the utility export.js with the specified parameters worked correctly itself.

4.6.3 Security User Guide

The security user guide is located here.

4.7 Personal account

Personal account is used for displaying different "instrumental" forms, etc. in the applications when creating navigation (or page forms), the module finds them and forms the navigation and pages display logics.

4.7.1 Module configuration

1. Put the html-pages in a directory of the application.
2. Write this directory in deploy.json in the PA module settings, and also optionally a hash array of a match between the file name and the display name.
3. The personal account reads these settings and displays the PA navigation menu in the master template.
4. When clicking on the menu item in the working area of the PA window, the corresponding markup from the html file is displayed.
5. Also in deploy.json file, in the PA module settings , the tool page is specified by default. If not specified, the first in order is taken.

There is also the possibility of structuring the menu by nesting directories.

4.8 Soap Module

NB: Soap module does not support the GET requests to services.

This is partly due to the fact that a SOAP request is transmitted in the request body, while a GET request does not have a body (so to speak). For this reason, you need to send a POST request. This can be done using the SOAP-UI utility (it is also possible in a browser, but in the request body you need to write a SOAP request, which is based on WSDL and is quite cumbersome).

4.8.1 Crud service data structure settings

The ‘‘ types option contains a mapping specified in the form of a mapping between the class (full name) and the map of the published attributes of this class. In map, the key is the attribute name, and the value is either a string alias or a Boolean value indicating whether the attribute is included in the schema or not, i.e., if an alias is specified, the attribute appears in the schema under this alias, in all other cases, except for specifying ` `false, the attribute appears under its own name.

The setting is used when parsing classes when forming a service data schema, as well as when parsing incoming messages and generating responses. By replacing normalize with a function that correctly converts the data to the schema.

For collections and references

If the values of collections and links also need to be parsed somehow differently, the objects that are in these properties can also be described by a map in deploy.json according to this principle:

```
"название_свойства": {
  "name": "новое_название(если нужно, поле необязательное)",
 "types": {
 //описания свойств
 }
}
```

Example

```
"petitionExperts": {
        "module": "modules/soap/service/crud",
        "options": {
          "dataRepo": "ion://dataRepo",
          "metaRepo": "ion://metaRepo",
          "keyProvider": "ion://keyProvider",
          "namespace": "khv-gosekspertiza",
          "className": "petitionExpert",
          "types": {
            "petitionExperts@khv-gosekspertiza":{
              "property1":"new_property_name",
              "property2":true
              }
            }
          }
        }
```

4.8.2 The setting for removing system attributes from a request

```
let gosEkspRemPetNew = normalize(e.item, null, {skipSystemAttrs: true});
```

4.8.3 Oauth2 token authentication in the SOAP module

Login-password and login-token authentication is applied by default for all services. Add the WSSecurity security header to the message to authenticate soap requests. To authenticate REST services, add standard HTTP authentication headers.

Set the type of verification - by password or token (pwd/token) in deploy.json file by setting the authMode in the corresponding module:

Example

```
"soap": {
    "globals": {
      "authMode": {
        "petitionExperts": "none",
        "petitionEstimated": "none",
        "gosEkspContract": "none",
        "bankAccounts": "none",
        "resolution": "none"
      }
```

By default, all services are authenticated by passwords. The admin panel has a special form to generate a user token. Configure the authMode for the service in token, go to the admin panel, generate a token and use it instead of the password in the headers.

4.9 Image-storage Module

Image-storage is an IONDV.Framework module to preview pictures.

4.9.1 Description and purpose of the module

The module is an additional system component and is used for application attributes, which contain images.

To connect a module, specify it in the application dependencies. To do this, specify this module in the package.json file of the application, in the "dependencies" section:

```
"dependencies": {
   "image-storage": "git+https://github.com/iondv/image-storage.git"
}
```

with a link to the module's repository in github.com.

Next, you need to add the attribute, which image will be used in a preview mode, to the list form. After that, images uploaded for objects can be viewed on the list form.

The function of Image-storage Module

- Preview of images without opening the object form

4.10 REST Module

REST is a module providing IONDV application data opetations through REST API. It is used to create web services for various types of data, including visually designed in IONDV. Studio.

4.10.1 Description

IONDV. REST is a module for:

- out-of-the-box CRUD model for accessing data, managing business processes, various types of authorization - as a backend for mobile applications, for SPA applications (created on the frameworks of Angular, Redux, Vue, etc.) or applications with divided front and back office;
- rapid development of proprietary web services by registering them and writing code on a ready-made data operation API for implementation of a microservice architecture;
- providing the integration of applications created on IONDV. Framework with other systems using the REST API.

IONDV. REST module is a wrapper for working with data through standard CRUD functions or connecting your own application services, including those using the core API.

More details:

Authorization for requests to services

Getting a token

There are two ways to get a token: in the console of the ionadmin module or through the token service of the rest module.

All generated tokens are stored in the ion_user_tokens collection in the application database.

Getting a permanent token via the ionadmin module

To get a token via the admin console, go to the navigation item “Web Service Security Keys” in the ionadmin module, for example, at locahost:8888/ionadmin/token

On the “Security Token Generator” page:

- Enter the user name in the “User Name” field
- Specify “local” value in the “Account Type” field
- Click the “Generate Token” button
- The “Token” field will display a token value similar to 3a546090355317c287886c0e81dfd304fa5bda99, and it should be used as the auth-token header value.

The default token lifetime is 100 years.

Getting a temporary token via the rest/token service

The second way to get the token is to use the rest module web service - token. You can get a token through an authenticated request to the address rest/token. More details.

Proxy client for access to module functions without receiving a new token

Client connection is carried out in modules.registry.globals.di in deploy.json:

```
{
  "modules": {
    "registry": {
      "globals": {
        "di":{
          "apiGateWay": {
            "module": "modules/rest/client/GateWay",
            "options": {
              "log": "ion://sysLog",
              "base": "/registry-ajax-api",
              "clientId": "ext@system",
              "clientSecret": "ion-demo",
              "tokenPath": "/rest/token",
              "endPoint": "[[rest.endPoint]]",
              "definition": {
                "paths": {
                  "/rest/echo-token": {
                    "post": true,
                    "get": true
```

(continues on next page)

(continued from previous page)

```
            }
          }
        }
      }
    }
```

Example of a request to rest/echo-token through a proxy client:

```
curl -X POST --cookie-jar 1.txt -d username="demo@local" -d password="ion-demo" http://localhost:8888/auth
curl -X GET --cookie 1.txt https://dnt.iondv.com/registry-ajax-api/rest/echo-token
```

example of a request in dnt: test/modules/rest/gateway.spec.js

```
/Checking rest-api proxy
```

Authorization to access services can be done in the following ways:

- Without authorization
- With account
- With token
- Oauth2

Services without authentication

To implement the operation of the service without authentication, it's necessary to set its vaue to none in the authMode setting in deploy.json

```
{
  "modules": {
    "rest": {
      "globals": {
        "authMode": {
          "echo": "none"
```

The request to the service will not require authentication, example of request curl https://dnt.iondv.com/rest/echo

An example of a request to a service without authentication in dnt: test/modules/rest/echo.spec.js

```
/Checking echo-pwd service/# Requesting echo-pwd GET/check if the request can be made using the headers auth
```

Services with a standard account authorization mechanism

All services use the standard authorization mechanism by default, which implies the transfer of credentials in the header:

- by authorization via basicAuth, example

  ```
  curl -u demo@local:ion-demo https://dnt.iondv.com/rest/simple
  ```

  an example of a request with Basic Auth authorization in develop-and-test (dnt): test/modules/rest/echopwd.spec.js

```
/Checking echo-pwd service/# Requesting echo-pwd GET/check if the request can be made using the␣
↪basicAuth
```

- by passing credentials in the request headers

```
curl -H "auth-user: demo" -H "auth-pwd: ion-demo" -H "auth-user-type: local" https://dnt.iondv.com/rest/
↪simple
```

or

```
curl -H "auth-user: demo@local" -H "auth-pwd: ion-demo" https://dnt.iondv.com/rest/simple
```

an example of a request with authorization credentials in the header in dnt: test/modules/rest/echopwd.spec.js

```
/Checking echo-pwd service/# Requesting echo-pwd GET/check if the request can be made using the␣
↪headers auth
```

Services with token authentication

Token authentication is used to exclude the constant transfer of an account in requests. Tokens are limited in their lifetime.

To implement the service operation with authentication with a token, you have to set the token value in the authMode setting in the deploy.json

```
{
  "modules": {
    "rest": {
      "globals": {
        "authMode": {
          "echo-token": "token"
```

Authentication through a token is performed by sending the token value in the auth-token request header

```
curl -H "auth-token: c369a361db9742e9a9ae8e9fe55950a571493812" http://dnt.iondv.com/rest/echo-token
```

an example of a request with authorization via a token in dnt: test/modules/rest/token.spec.js

```
/Checking token service/# basicAuth authorization with admin rights/# check if the generated token is valid␣
↪(basicAuth) (using echo-token)
```

Learn more about getting a token: Getting token

Proxy client to access the module functions without getting a new token.

Services with OAuth 2 authentication

To implement the service with oauth2 authentication, you have to first enable it in deploy.json plugin of the form

```
"oauth": {
        "module": "lib/oAuthAdapter",
        "options": {
```

(continues on next page)

(continued from previous page)

```
        "auth": "ion://auth",
        "dataSource": "ion://Db"
      }
    }
```

then you can set the service to oauth in the setting auth_mode:

```
{
  "modules": {
    "rest": {
      "globals": {
        "authMode": {
          "echo-oauth": "oauth"
```

oauth2 specification is available at: https://oauth2-server.readthedocs.io/en/latest/index.html

This type of authorization is used to provide a third party with limited access to user resources without having to provide a username and password. Requests for access are made in the following order:

1. From the user's side, we get a cookie with session id:

   ```
   curl -X POST --cookie-jar 1.txt -d username="demo@local" -d password="ion-demo" http://dnt.iondv.com/
   ↪auth
   ```

2. Using an authorized session we allow ext@system client the requests on our behalf:

   ```
   curl -X POST --cookie ./1.txt "http://dnt.iondv.com/oauth2/grant?client_id=ext@system&response_
   ↪type=code&state=123"
   ```

   The response will contain the code parameter.

3. Now using code you can get a token:

   ```
   curl -X POST -d grant_type="authorization_code" -d code="<code>" -H "Authorization:Basic␣
   ↪ZXh0QHN5c3RlbTppb24tZGVtbw==" http://dnt.iondv.com/oauth2/token
   ```

   in the Authorization header, enter Basic <client_secret> client code. The response will contain access_token.

4. For requests on behalf of the user in services with oauth2 authorization, you can now log in using access_token:

   ```
   curl -X POST -H "Authorization:Bearer <access_token>" http://dnt.iondv.com/rest/echo-oauth
   ```

an example of a request to a service with oauth2 authorization in dnt: test/modules/rest/echooauth.spec.js

```
/Checking echo-oauth service
```

REST Services

Built-in service Acceptor

The " acceptor" service is intended for mass storage of objects of different classes.

Available at <server address>/rest/acceptor.

To work with the service, its registration in the application configuration file deploy.json is required. Also the repositories dataRepo and metaRepo must be specified in options for the service. For example:

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "acceptor": {
            "module": "modules/rest/lib/impl/acceptor",
            "options": {
              "dataRepo": "ion://dataRepo",
              "metaRepo": "ion://metaRepo"
            }
          }
```

Authorization is carried out through all major access types.

The service uses the POSTmethod, objects are passed as an array of objects in JSON format in the request body, with the mandatory indication of the json content Content-Type:application/json in the header. You don't need to specify auto-generated fields.

In the header in the property ion-converter the name of the converter that should be used when processing data - both the request and the response - can be passed. And the data converter itself must be registered in the options of the service. If no handler is specified, the default handler is used.

The object data must include:

- _id - the object ID in the key field
- _class - object class with namespace
- _classVer - class version

All the rest of the values must correspond to class properties, including data types correspondance. Example:

```
curl -X POST -u demo@local:ion-demo \
  -H "Content-Type:application/json" \
  -d '[{"_class": "class_string@develop-and-test", "__classVer": null,"id": "10101010-5583-11e6-aef7-
↪cf50314f026b",\
      "string_text": "Example10", "string_miltilinetext": "Example10", "string_formattext": "Example10"}]' \
  https://dnt.iondv.com/rest/acceptor
```

an example of a request for creating objects to the acceptor service in dnt: test/modules/rest/acceptor.spec.js

```
/Checking acceptor service/# basicAuth authorization with admin rights, POSTing strings/# result of creation␣
↪of objects
```

The method returns the code 200 and an array of stored objects.

```
[
  {
    "id": "10101010-5583-11e6-aef7-cf50314f026b",
    "_class": "class_string@develop-and-test",
    "_classVer": "",
    "string_formattext": "Example10",
    "string_miltilinetext": "Example10",
    "string_text": "Example10",
    "_id": "10101010-5583-11e6-aef7-cf50314f026b"
```

(continues on next page)

(continued from previous page)

```
  }
]
```

In case of an error, the response code will be 400 , and the response text will contain

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Bad Request</pre>
</body>
</html>
```

Built-in service Token

Token service is for issuing a token to an authenticated user for its further use in services that authenticate using a token.

Available at <server address>/test/token.

The service doesn't require registration in deploy.json. The service provides the issuance of a token for an authorized user, if a user has the use rights for the resource ‘‘ws:::gen-ws-token‘‘or has administrator rights.

A token of the form e444c69894d2087696e0a6c6914788f67ebcf6ee is returned in response to the request. The default token lifetime is 100 years.

An example of the request through the Basic Auth type authentication

```
curl -u demo@local:ion-demo https://dnt.iondv.com/rest/token
```

** An example of a request with authentication through parameters in the header **

```
curl -H "auth-user: demo@local" -H "auth-pwd: ion-demo" -H "auth-user-type: local" https://dnt.iondv.com/rest/
↪token
```

Examples of requestd to the token service in dnt: test/modules/rest/token.spec.js

```
/Checking token service/# basicAuth authorization with admin rights
/Checking token service/# authorization with admin rights using header parameters
```

Resource rights

Add the resource for generating tokens for the role using the command line node bin/acl.js --role restGrp --p USE --res ws:::gen-ws-token (where restGrp is a name of existing group)

Another way to add rights to a resource is to use the administrator console of the ionadmin module, for example, at locahost:8888/ionadmin/:

- Select the “Security” navigation item
- Select the “Roles” navigation sub-item
- Select the existing role and click on “Edit” or “Create new role”.

- In the role field “Access rights”, select the “Services” tab
- Expand the list of rights for the resource “Generation of security tokens through web services (ws:::gen-ws-token)”
- Select “Use” and click “Save”

Built-in service CRUD

Crud service implements the REST API based on the model of the main CRUD operations (create, read, update, delete).

Available at <server address>/rest/crud.

The service requires the application registration and the data source ‘‘ dataRepo‘‘ indication in the options of the service in deploy.json, as well as the ‘‘ auth‘‘ authorization source to access user data. It is advisable to specify a repository with full security processing as a data repository, in order to work out access to objects taking into account dynamic security. For example:

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "crud": {
            "module": "modules/rest/lib/impl/crud",
            "options": {
              "auth": "ion://auth",
              "dataRepo": "ion://securedDataRepo"
            }
          }
```

Authentication is carried out through all the main access types.

Example:

```
curl -X POST -u demo@local:ion-demo https://dnt.iondv.com/rest/crud
```

An example of a request to the crud service without parameters in dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/# check if the response for null parameters is valid
```

By default, without the correct parameters - the server responds with the 404 error code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot POST /rest/crud</pre>
</body>
</html>
```

Information on interacting with crud through basic methods:

Creating an object: POST method

Creating an object is carried out by the method POST, with the class code with the namespace specified, for example rest/crud/class_string@develop-and-test. The object itself is passed in the body of the request in json format and in the header the json type of content Content-Type:application/json must be specified. Auto-generated fields are optional.

Example:

```
curl -X POST -u demo@local:ion-demo \
  -H "Content-Type:application/json" \
  -d '{"string_text": "Example3", "string_miltilinetext": "Example3", "string_formattext": "Example3"}' \
  https://dnt.iondv.com/rest/crud/class_string@develop-and-test/
```

An example of a request to the crud service to create an object dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/POST/# creating an object (POST)
```

In response, the created object will be returned, in which all auto-created fields will be filled and the response code 200 will be indicated.


```
{
   "_creator": "admin@local",
   "_id": "10c77900-b96e-11e9-a7ce-314f02bd4197",
   "__string": "10c77900-b96e-11e9-a7ce-314f02bd4197",
   "__class": "class_string@develop-and-test",
   "__classTitle": "Class \"String [0]\"",
   "id": "10c77900-b96e-11e9-a7ce-314f02bd4197",
   "string_text": "Example3",
   "string_miltilinetext": "Example3",
   "string_formattext": "Example3"
}
```


In case of an error, the response code will be 400 , and the response text will contain

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Bad Request</pre>
</body>
</html>
```

Getting an object or a list of objects: GET method

Getting an object

Obtaining an object is performed using the GETmethod, and the code of the namespace class and the value of the object key are specified, for example“rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b“

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/66dbb3d0-
↪5583-11e6-aef7-cf50314f026b
```

An example of a request to the crud service to get an object in dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/GET/# getting an object (GET)
```

In addition, the _eager parameter that contains a list of class properties separated by the ‘ |‘ symbol for which data must be eagerly loaded (links or collections) can be set in query . For example

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/66dbb3d0-
↪5583-11e6-aef7-cf50314f026b?_eager=string_text
```

An example of a request to the crud service to get an object with eager loading of the “table” property in dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/GET/# getting an object with eager loading of the "table" property (GET)
```

If the object exists, the response code 200 and the object itself in json format are returned, if the object was not found - 404, if there are no rights - 403.


```
{
    "_id": "66dbb3d0-5583-11e6-aef7-cf50314f026b",
    "__string": "Example of the \"String [0]\" type in the \"Text [1]\" view",
    "__class": "class_string@develop-and-test",
    "__classTitle": "Class \"String [0]\"",
    "id": "66dbb3d0-5583-11e6-aef7-cf50314f026b",
    "string_text": "Example of the \"String [0]\" type in the \"Text [1]\" view",
    "string_miltilinetext": "Example of the \"String [0]\"\r\n in the Multiline text [7] view",
    "string_formattext": "Example of the \r\n \"String [0]\" type \r\n in the \r\nFormatted text [7] view"
}
```


Getting a list of objects

The list of objects is requested using the GETmethod, and the class code and namespace are specified, for example, rest/crud/class_string@develop-and-test/

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/
```

An example of a request to the crud service to get a list of objects in dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/GET/# getting a list of text objects
```

In response, the service issues a JSON Object with an offset of 0 and a count of 5 records and a status of 200 , if there is no such class, it shows the code 404.


```
[{"_creator":"admin@local",
"_id":"4567a830-b8ea-11e9-9cdf-7bd384cbb7a5",
"__string":"example1",
"__class":"class_string@develop-and-test",
"__classTitle":"Class \"String [0]\"",
"id":"4567a830-b8ea-11e9-9cdf-7bd384cbb7a5",
"string_text":"example1",
"string_miltilinetext":"example1",
"string_formattext":"<p>example1</p>"},
```


(continues on next page)

(continued from previous page)


```
{"_id":"66dbb3d0-5583-11e6-aef7-cf50314f026b",
"__string":"Example of the \"String [0]\" type in the \"Text [1]\" view",
"__class":"class_string@develop-and-test",
"__classTitle":"Class \"String [0]\"",
"id":"66dbb3d0-5583-11e6-aef7-cf50314f026b",
"string_text":"Example of the \"String [0]\" type in the \"Text [1]\" view",
"string_miltilinetext":"Example of the \"String [0]\"\r\n in the Multiline text [7] view",
"string_formattext":"Example of the \r\n \"String [0]\" type \r\n in the \r\nFormatted text [7] view"}]
```


The query can be implemented with the following query parameters:

- _offset - sampling offset, 0 by default
- _count - number of values in the sample, by default 5
- _eager - a list of class properties separated by the | symbol for which data must be eagerly loaded.
- [name of property] - all parameters are assumed to be query names, except those beginning with "_"which are considered class attribute names, and their values are set as filters.

Examples:

1. Request for a list of class objects with offset 1 and count 2

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/?_
↪offset=1&_count=2
```

2. Request for a list of objects whose string_text property is set to example1

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/?
↪string_text=example1
```

3. Request for a list of objects for which the string_text property has the value example1, with an offset of 1 and a count of 2

```
curl -X GET -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/?
↪string_text=example1&_offset=1&_count=2
```

An example of a request to the crud service to get a list of objects with different shift and filter parameters in dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/GET/# getting a list of text objects, with an offset of 1 and a count of 2
/Checking crud service/GET/# getting a list of text objects containing a specific string
```

Getting objects using the SEARCH method

The SEARCH HTTP method is accessed in the CRUD service in the same way as the GET``method. The exception is that in the body of the message it is possible to set the * filtering * \, *sorting* and *greedy loading * conditions in the .json format in the notation of the `` GetList method of the data repository:

```
{
  "filter": {
    "eq": ["attr3", "etalon"]
  },
  "forceEnrichment": [["attr1", "attr2"], ["col1"]]
  "sort": {
```

(continues on next page)

(continued from previous page)

```
    "attr4": -1
  }
}
```

```
curl -X GET -u admin@local:ION-admin http://modws-26.develop-and-test.kube.local/rest/crud/class_
↪string@develop-and-test/
```

Checking for the presence of an object: the HEAD method

Checking for the presence of the object is carried out by the method HEAD, with the class code with its namespace and the value of the object key, for example rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b

```
curl -X HEAD -u demo@local:ion-demo https://dnt.iondv.com/rest/crud/class_string@develop-and-test/66dbb3d0-
↪5583-11e6-aef7-cf50314f026b
```

An example of a request to the crud service to check the presence of an object in dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/HEAD/# checking if an object is present (HEAD)
```

If the object exists, the response code “200”is returned, if the object is not found - 404, if there are no permissions - 403.

Updating an object: PATCH and PUT methods

Updating an object is carried out using the method PATCH or PUT, with the class code with the namespace and the object key value specified, for example rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b. The object itself is passed in the body of the request in json format and in the header the json type of content Content-Type:application/json must be specified.

Example:

```
curl -X PATCH -u demo@local:demo-ion -H "Content-Type:application/json" -d '{"string_text": "NEW Example
↪", "string_miltilinetext": "NEW Example", "string_formattext": "NEW Example"}' https://dnt.iondv.com/
↪rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b
# Или эквивалентно
curl -X PUT -u demo@local:demo-ion -H "Content-Type:application/json" -d '{"string_text": "NEW Example",
↪"string_miltilinetext": "NEW Example", "string_formattext": "NEW Example"}' https://dnt.iondv.com/rest/
↪crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b
```

An example of a request to the crud service to update an object in dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/PATCH/# updating an object (PATCH)
```

If the object exists, the response code 200 and the object itself in json format are returned, if the object does not find the code is 404, if the processing fails, the code is 500 , if there are no rights - 403.

Example of an object.

```
{
  "_editor": "admin@local",
```

(continues on next page)

(continued from previous page)


```
    "_id": "66dbb3d0-5583-11e6-aef7-cf50314f026b",
    "__string": "NEW Example",
    "__class": "class_string@develop-and-test",
    "__classTitle": "Class \"String [0]\"",
    "id": "66dbb3d0-5583-11e6-aef7-cf50314f026b",
    "string_text": "NEW Example",
    "string_miltilinetext": "NEW Example",
    "string_formattext": "NEW Example"
}
```


Deleting an object: the DELETE method

Deleting an object carried out by the method DELETE, and the class code with the namespace and the object key value are specified, for example rest/crud/class_string@develop-and-test/66dbb3d0-5583-11e6-aef7-cf50314f026b.

```
curl -X DELETE -u demo@local:demo-ion https://dnt.iondv.com/rest/crud/class_string@develop-and-test/
↪66dbb3d0-5583-11e6-aef7-cf50314f026b
```

An example of a request to the crud service to delete objects in dnt: test/modules/rest/crud.spec.js

```
/Checking crud service/DELETE/# deleting an object (DELETE)
```

If successful, the service returns the response code 200, if the object is not found 404.

Sending requests with files in the CRUD service

When requesting to CRUD using POST, PATCH and PUT methods, you can transfer files in the request body.

Files are sent and received in two ways:

- the data is sent in the json format, then the file content is transferred as a string in the Base64 format in the corresponding field of the metadata class.
- data is sent as FormData (application/x-www-form-urlencoded), then files are transmitted as multipart.

Correct reception of file attributes in case of sending such requests is carried out using the POST , PUT and PATCH methods in the CRUD service.

It is also possible to transfer references and collections according to the example described for the soap module.

The examples of the POST requests with files to CRUD in dnt: test/modules/rest/crud.spec.js

```
/checking crud service/# sending a file with multipart body request (POST)
/checking crud service/# sending a file with json body request (POST)
```

Meta metadata publishing Service

Meta meta is a built-in service in the rest module, providing access to the meta repository interface in the form of web-service.

The service requires a connection in deploy.json and a mandatory indication of options.dataRepo and options.metaRepo, example:

```
"meta": {
       "module": "modules/rest/lib/impl/meta",
       "options": {
         "dataRepo": "ion://dataRepo",
         "metaRepo": "ion://metaRepo"
       }
}
```

All types of authorization are supported, by default is the authorization with credentials.

The service provides access to the following ** GET ** requests:

Getting information about the metadata class: getMeta

The request is made along the path <server URL>/rest/<service name>/getMeta/<class name>, where the class name is indicated with a namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- version
- namespace

Query example:

```
https://localhost:8888/rest/meta/getMeta/class_text@develop-and-test
```

Information on the class_text class in the namespace develop-and-test will be requested, an example response:

```
{ namespace: 'develop-and-test',
isStruct: false,
metaVersion: '2.0.7',
key: [ 'id' ],
semantic: '',
name: 'class_text',
version: '',
caption: 'Class "Text [1]"',
ancestor: null,
container: null,
creationTracker: '',
changeTracker: '',
history: 0,
journaling: false,
compositeIndexes: null,
properties:
[ { orderNumber: 10,
   name: 'id',
   caption: 'Identifier',
   type: 12,
   size: 24,
   decimals: 0,
   allowedFileTypes: null,
   maxFileCount: 0,
   nullable: false,
```

(continues on next page)

(continued from previous page)

```
  readonly: false,
  indexed: true,
  unique: true,
  autoassigned: true,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null },
{ orderNumber: 20,
  name: 'text_text',
  caption: 'Text [1]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null },
{ orderNumber: 30,
  name: 'text_multilinetext',
  caption: 'Multiline text [7]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
```

(continues on next page)

(continued from previous page)


```
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null },
{ orderNumber: 40,
  name: 'text_formattext',
  caption: 'Formatted text [8]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null } ] }
```


The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing info about metadata class: getMeta
```

Getting a list of all metadata classes: listMeta

The request is made using the path <server URL>/rest/<service name>/list Meta.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- ancestor
- version

- direct
- namespace

Query example:

```
https://localhost:8888/rest/meta/listMeta
```

information will be requested for all classes from the meta repository specified in the meta service settings in deploy.json.

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing the list of metadata classes: listMeta
```

Getting information about the ancestor class: ancestor

The request is made along the path <server URL>/rest/<service name>/ancestor/<class name>, where the class name is indicated with a namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- version
- namespace

Query example:

```
https://localhost:8888/rest/meta/getMeta/event@develop-and-test
```

Information will be requested on the ancestor class of the event class in the develop-and-test namespace - BasicObj.

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing info about class ancestor
```

Getting information about properties of objects of the class: propertyMetas

The request is made using the path <server URL>/rest/<service name>/propertyMetas/<class name>, where the class name is specified with a namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- version
- namespace

Query example:

```
https://localhost:8888/rest/meta/propertyMetas/class_text@develop-and-test
```

information about the properties of objects of the class_text class in the develop-and-test namespace will be requested, for example:

```
[ { orderNumber: 10,
  name: 'id',
  caption: 'Identifier',
  type: 12,
  size: 24,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: false,
  readonly: false,
  indexed: true,
  unique: true,
  autoassigned: true,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null,
  definitionClass: 'class_text@develop-and-test',
  mode: 0 },
{ orderNumber: 20,
  name: 'text_text',
  caption: 'Text [1]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null,
  definitionClass: 'class_text@develop-and-test',
```


(continues on next page)

(continued from previous page)

```
  mode: 0 },
{ orderNumber: 30,
  name: 'text_multilinetext',
  caption: 'Multiline text [7]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
  formula: null,
  definitionClass: 'class_text@develop-and-test',
  mode: 0 },
{ orderNumber: 40,
  name: 'text_formattext',
  caption: 'Formatted text [8]',
  type: 1,
  size: null,
  decimals: 0,
  allowedFileTypes: null,
  maxFileCount: 0,
  nullable: true,
  readonly: false,
  indexed: true,
  unique: false,
  autoassigned: false,
  hint: null,
  defaultValue: null,
  refClass: '',
  itemsClass: '',
  backRef: '',
  backColl: '',
  binding: '',
  semantic: null,
  selConditions: null,
  selSorting: [],
  selectionProvider: null,
  indexSearch: false,
  eagerLoading: false,
```

(continues on next page)

(continued from previous page)

```
  formula: null,
  definitionClass: 'class_text@develop-and-test',
  mode: 0 } ]
```

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing info about meta object properties: propertyMetas
```

Getting a list of navigation sections: getNavigationSections

The request is carried out along the path <server URL>/rest/<service name>/getNavigationSections.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- namespace

Query example:

```
https://localhost:8888/rest/meta/getNavigationSections
```

a list of all objects of the navigation sections from the meta repository specified in the settings of the meta service in deploy.json will be requested.

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing the list of navigation sections: getNavigationSections
```

Getting information about the navigation section: getNavigationSection

The request is made along the path <server URL>/rest/<service name>/getNavigationSection/<section code> where the section code is specified in the namespace @ name format.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- namespace

Query example:

```
https://localhost:8888/rest/meta/getNavigationSection/develop-and-test@simpleTypes
```

the object of the navigation section simpleTypes in the namespace develop-and-test will be requested, an example response:

```
{ caption: 'Simple types',
    code: 'simpTypes',
    name: 'simpleTypes',
    orderNumber: 20,
    mode: 0,
    tags: null,
    metaVersion: '2.0.7',
    itemType: 'section',
    namespace: 'develop-and-test',
    nodes:
    { class_boolean:
      { namespace: 'develop-and-test',
```

(continues on next page)

(continued from previous page)

```
    code: 'class_boolean',
    orderNumber: 0,
    type: 0,
    caption: 'Class "Boolean [10]"',
    classname: null,
    container: null,
    collection: null,
    url: null,
    hint: null,
    conditions: null,
    sorting: [],
    pathChains: [],
    title: '',
    metaVersion: '2.0.0',
    itemType: 'node',
    section: 'develop-and-test@simpleTypes',
    children: [Array] },
  class_custom:
   { namespace: 'develop-and-test',
    code: 'class_custom',
    orderNumber: 0,
    type: 1,
    caption: 'Class "User type [17]"',
    classname: 'class_custom@develop-and-test',
    container: null,
    collection: null,
    url: null,
    hint: null,
    conditions: null,
    sorting: [],
    pathChains: [],
    title: '',
    metaVersion: '2.0.0',
    itemType: 'node',
    section: 'develop-and-test@simpleTypes',
    children: [] },
  class_datetime:
   { namespace: 'develop-and-test',
    code: 'class_datetime',
    orderNumber: 0,
    type: 1,
    caption: 'Class "Date/time [9]"',
    classname: 'class_datetime@develop-and-test',
    container: null,
    collection: null,
    url: null,
    hint: null,
    conditions: null,
    sorting: [],
    pathChains: [],
    title: '',
    metaVersion: '2.0.0',
    itemType: 'node',
    section: 'develop-and-test@simpleTypes',
    children: [] },
  class_decimal:
```

(continues on next page)

(continued from previous page)

```
    { code: 'class_decimal',
      orderNumber: 0,
      type: 1,
      caption: 'Class "Decimal [8]"',
      classname: 'class_decimal@develop-and-test',
      container: null,
      collection: null,
      url: null,
      hint: null,
      conditions: null,
      sorting: [],
      pathChains: [],
      metaVersion: '2.0.7',
      itemType: 'node',
      section: 'develop-and-test@simpleTypes',
      namespace: 'develop-and-test',
      children: [] }
  }
}
```

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access info about a navigation section: getNavigationSection
```

Getting information about the navigation node: getNode

The request is made along the path <server URL>/rest/<service name>/getNode/<navigation code> where the navigation code is specified in the format namespace@code.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- namespace

Query example:

```
https://localhost:8888/rest/meta/getNode/develop-and-test@semantic
```

the semantic navigation node object will be requested in the develop-and-test namespace, example response:

```
{ code: 'semantic',
orderNumber: 0,
type: 0,
caption: 'Display semantics [semantic]',
classname: null,
container: null,
collection: null,
url: null,
hint: null,
conditions: null,
sorting: [],
pathChains: [],
metaVersion: '2.0.61',
itemType: 'node',
section: 'develop-and-test@classProperties',
namespace: 'develop-and-test',
```

(continues on next page)

(continued from previous page)

```
children:
[ { code: 'semantic.semErrCatalog',
   orderNumber: 0,
   type: 1,
   caption:
    'Catalog for checking semantics (string1| |string2| |string3| |date| |integer)',
   classname: 'semErrCatalog@develop-and-test',
   container: null,
   collection: null,
   url: null,
   hint: null,
   conditions: null,
   sorting: [],
   pathChains: [],
   metaVersion: '2.0.61',
   itemType: 'node',
   section: 'develop-and-test@classProperties',
   namespace: 'develop-and-test',
   children: [] },
 { code: 'semantic.semErrClass',
   orderNumber: 0,
   type: 1,
   caption: 'Attribute semantics is taken from reference class',
   classname: 'semErrClass@develop-and-test',
   container: null,
   collection: null,
   url: null,
   hint: null,
   conditions: null,
   sorting: [],
   pathChains: [],
   metaVersion: '2.0.61',
   itemType: 'node',
   section: 'develop-and-test@classProperties',
   namespace: 'develop-and-test',
   children: [] }]
}
```

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access info about a navigation node: getNode
```

Getting a list of navigation nodes in the section: getNodes

The request is made along the path <server URL>/rest/<service name>/getNodes/<section code> where the section code is specified in the namespace@name format .

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- section
- parent
- namespace

Query example:

```
https://localhost:8888/rest/meta/getNodes/develop-and-test@simpleTypes
```

a list of navigation nodes will be requested in the simpleTypes section of the develop-and-test namespace.

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access the list of navigation nodes in a section: getNodes
```

Getting information about the list form for representing class objects: getListViewModel

The request is made along the path <server URL>/rest/<service name>/getListViewModel/<class name>, where the class name is indicated with the namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- node
- namespace
- version

Query example:

```
https://localhost:8888/rest/meta/getListViewModel/class_text@develop-and-test
```

an object will be requested to represent objects of the class_text class in a list form in the develop-and-test namespace, an example response:

```
 { columns:
[ { sorted: true,
    caption: 'Identifier',
    type: 1,
    property: 'id',
    size: 2,
    maskName: null,
    mask: null,
    mode: null,
    fields: [],
    hierarchyAttributes: null,
    columns: [],
    actions: null,
    commands: [],
    orderNumber: 10,
    required: false,
    visibility: null,
    enablement: null,
    obligation: null,
    readonly: true,
    selectionPaginated: true,
    validators: null,
    hint: '',
    historyDisplayMode: 0,
    tags: null },
  { sorted: true,
    caption: 'Text [1]',
    type: 1,
    property: 'text_text',
```

(continues on next page)

(continued from previous page)

```
    size: 2,
    maskName: null,
    mask: null,
    mode: null,
    fields: [],
    hierarchyAttributes: null,
    columns: [],
    actions: null,
    commands: [],
    orderNumber: 20,
    required: false,
    visibility: null,
    enablement: null,
    obligation: null,
    readonly: false,
    selectionPaginated: true,
    validators: null,
    hint: '',
    historyDisplayMode: 0,
    tags: null },
  { sorted: true,
    caption: 'Multiline text [7]',
    type: 7,
    property: 'text_multilinetext',
    size: 2,
    maskName: null,
    mask: null,
    mode: null,
    fields: [],
    hierarchyAttributes: null,
    columns: [],
    actions: null,
    commands: [],
    orderNumber: 30,
    required: false,
    visibility: null,
    enablement: null,
    obligation: null,
    readonly: false,
    selectionPaginated: true,
    validators: null,
    hint: '',
    historyDisplayMode: 0,
    tags: null },
  { sorted: true,
    caption: 'Formatted text [8]',
    type: 8,
    property: 'text_formattext',
    size: 2,
    maskName: null,
    mask: null,
    mode: null,
    fields: [],
    hierarchyAttributes: null,
    columns: [],
    actions: null,
```

(continues on next page)

(continued from previous page)

```
    commands: [],
    orderNumber: 40,
    required: false,
    visibility: null,
    enablement: null,
    obligation: null,
    readonly: false,
    selectionPaginated: true,
    validators: null,
    hint: '',
    historyDisplayMode: 0,
    tags: null } ],
actions: null,
commands:
 [ { id: 'CREATE',
    caption: 'Create',
    visibilityCondition: null,
    enableCondition: null,
    needSelectedItem: false,
    signBefore: false,
    signAfter: false,
    isBulk: false },
   { id: 'EDIT',
    caption: 'Edit',
    visibilityCondition: null,
    enableCondition: null,
    needSelectedItem: true,
    signBefore: false,
    signAfter: false,
    isBulk: false },
   { id: 'DELETE',
    caption: 'Delete',
    visibilityCondition: null,
    enableCondition: null,
    needSelectedItem: false,
    signBefore: false,
    signAfter: false,
    isBulk: true } ],
allowSearch: false,
pageSize: null,
useEditModels: true,
version: null,
overrideMode: null,
metaVersion: '2.0.7',
type: 'list',
className: 'class_text@develop-and-test',
path: '',
caption: '' }
```

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class list view model: getListViewModel
```

Getting information about the form of representation of class objects as a collection: getCollectionViewModel

The request is made along the path <server URL>/rest/<service name>/getCollectionViewModel/<class name>, where the class name is indicated with a namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- collection
- node
- namespace
- version

Query example:

```
https://localhost:8888/rest/meta/getCollectionViewModel/class_text@develop-and-test
```

an object will be requested to represent objects of the class_text class in the form of a collection in the develop-and-test namespace.

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class collection view model: getCollectionViewModel
```

Getting information about the presentation form of class objects when creating: getCreationViewModel

The request is made along the path ‘‘<server URL>/rest/<service name>/getCreationViewModel/<class name> ‘‘, where the class name is indicated with the namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- node
- namespace
- version

Query example:

```
https://localhost:8888/rest/meta/getCreationViewModel/class_text@develop-and-test
```

an object will be requested to represent objects of the class_text class when they are created in the develop-and-test namespace, an example response:

```
  { tabs: [ { caption: '', fullFields: [Array], shortFields: [] } ],
actions: null,
commands:
 [ { id: 'SAVE',
     caption: 'Save',
     visibilityCondition: null,
     enableCondition: null,
     needSelectedItem: false,
     signBefore: false,
     signAfter: false,
     isBulk: false },
   { id: 'SAVEANDCLOSE',
     caption: 'Save and close',
```

(continues on next page)

(continued from previous page)

```
    visibilityCondition: null,
    enableCondition: null,
    needSelectedItem: false,
    signBefore: false,
    signAfter: false,
    isBulk: false } ],
siblingFixBy: null,
siblingNavigateBy: null,
historyDisplayMode: 0,
collectionFilters: null,
version: null,
overrideMode: null,
metaVersion: '2.0.7',
type: 'create',
className: 'class_text@develop-and-test',
path: '',
caption: '' }
```

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class creation view model: getCreationViewModel
```

Getting information about the presentation form of class objects when editing: getDetailViewModel

The request is made along the path <server URL>/rest/<service name>/getItemViewModel/<class name>, where the class name is indicated with the namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- node
- namespace
- version

Query example:

```
https://localhost:8888/rest/meta/getItemViewModel/class_text@develop-and-test
```

an object will be requested to represent objects of the class_text class when editing them in the develop-and-test namespace, an example response:

```
  { tabs: [ { caption: '', fullFields: [Array], shortFields: [] } ],
actions: null,
commands:
 [ { id: 'SAVE',
    caption: 'Save',
    visibilityCondition: null,
    enableCondition: null,
    needSelectedItem: false,
    signBefore: false,
    signAfter: false,
    isBulk: false },
   { id: 'SAVEANDCLOSE',
    caption: 'Save and close',
    visibilityCondition: null,
```

(continues on next page)

(continued from previous page)

```
    enableCondition: null,
    needSelectedItem: false,
    signBefore: false,
    signAfter: false,
    isBulk: false } ],
siblingFixBy: null,
siblingNavigateBy: null,
historyDisplayMode: 0,
collectionFilters: null,
version: null,
overrideMode: null,
metaVersion: '2.0.7',
type: 'item',
className: 'class_text@develop-and-test',
path: '' }
```

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class item view model: getItemViewModel
```

Getting information about the presentation form of class objects when viewing: getDetailViewModel

The request is made along the path <server URL>/rest/<service name>/getDetailViewModel/<class name>, where the class name is indicated with the namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- node
- namespace
- version

Query example:

```
https://localhost:8888/rest/meta/getDetailViewModel/class_text@develop-and-test
```

an object will be requested to represent objects of the class_text class when viewing them in the develop-and-test namespace.

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class detail view model: getDetailViewModel
```

Getting a list of possible workflows for the class: getWorkflows

The request is made using the path <server URL>/rest/<service name>/getWorkflows/<class name>, where the class name is specified with a namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- namespace
- version

Query example:

```
https://localhost:8888/rest/meta/getWorkflows/workflowBase@develop-and-test
```

A list of workflows for the class workflowBase in the namespace develop-and-test will be requested, an example response:

```
[ { name: 'simpleWorkflow',
caption: 'Simple WF',
wfClass: 'workflowBase@develop-and-test',
startState: 'canStart',
states: [ [Object], [Object], [Object], [Object], [Object] ],
transitions: [ [Object], [Object], [Object], [Object], [Object] ],
metaVersion: '2.0.61.16945',
namespace: 'develop-and-test' } ]
```

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing the list of possible workflows for meta class: getWorkflows
```

Getting information about the representation form of a class object when the workflow is in a certain state: getListViewModel

The request is made using the path <server URL>/rest/<service name>/getListView Model/<class name>/<workflow name>/<state name>, where the class name and workflow name are specified with namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- workflow
- state
- namespace
- version

Query example:

```
https://localhost:8888/rest/meta/getWorkflowView/workflowBase@develop-and-test/simpleWorkflow@develop-and-
↪test/canStart
```

an object will be requested to represent objects of the workflowBase class with the canStart state of the simpleWorkflow workflow in the ‘‘develop-and-test‘‘namespace, example of a response:

```
  { tabs: [ { caption: '', fullFields: [Array], shortFields: [] } ],
actions: null,
siblingFixBy: null,
siblingNavigateBy: null,
historyDisplayMode: 0,
collectionFilters: null,
version: null,
overrideMode: 1,
commands:
 [ { id: 'SAVE',
     caption: 'Save',
     visibilityCondition: null,
     enableCondition: null,
     needSelectedItem: false,
```

(continues on next page)

(continued from previous page)

```
    signBefore: false,
    signAfter: false,
    isBulk: false },
  { id: 'SAVEANDCLOSE',
    caption: 'Save and close',
    visibilityCondition: null,
    enableCondition: null,
    needSelectedItem: false,
    signBefore: false,
    signAfter: false,
    isBulk: false } ],
metaVersion: '2.0.61',
type: 'item',
className: 'workflowBase@develop-and-test',
path: 'workflows:simpleWorkflow@develop-and-test.canStart',
caption: '' }
```

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access meta class view model in a certain workflow state: getWorkflowView
```

Getting information about the workflow of the class: getWorkflow

The request is made using the path <server URL>/rest/<service name>/getWorkflow/<class name>/<business process name>, where the class name and business process name are specified with a namespace.

You can specify in the request: doc: additional parameters <meta_query_parameters>:

- workflow
- namespace
- version

Query example:

```
https://localhost:8888/rest/meta/getWorkflow/workflowBase@develop-and-test/simpleWorkflow@develop-and-test
```

the workflow object simpleWorkflow``` will be requested for the object class ``` workflowBase in the namespace develop-and-test. A response example:

```
  { name: 'simpleWorkflow',
caption: 'Simple WF',
wfClass: 'workflowBase@develop-and-test',
startState: 'canStart',
states:
 [ { name: 'canStart',
    caption: 'Ready to check',
    maxPeriod: null,
    conditions: [Object],
    propertyPermissions: [],
    itemPermissions: [],
    selectionProviders: [] },
  { name: 'inProcess',
    caption: 'In process',
    maxPeriod: null,
```

(continues on next page)

(continued from previous page)

```
    conditions: null,
    itemPermissions: [Array],
    propertyPermissions: [],
    selectionProviders: [] },
  { name: 'accepted',
    caption: 'Accepted',
    maxPeriod: null,
    conditions: null,
    itemPermissions: [],
    propertyPermissions: [],
    selectionProviders: [] },
  { name: 'returned',
    caption: 'Returned',
    maxPeriod: null,
    conditions: null,
    itemPermissions: [Array],
    propertyPermissions: [],
    selectionProviders: [] },
  { name: 'rejected',
    caption: 'Rejected',
    maxPeriod: null,
    conditions: null,
    itemPermissions: [],
    propertyPermissions: [],
    selectionProviders: [] } ],
transitions:
 [ { name: 'startCheck',
    caption: 'Start checking',
    startState: 'canStart',
    finishState: 'inProcess',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
    conditions: null,
    confirm: false,
    confirmMessage: null },
  { name: 'return',
    caption: 'Return',
    startState: 'inProcess',
    finishState: 'returned',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
    conditions: null,
    confirm: false,
    confirmMessage: null },
  { name: 'accept',
    caption: 'Accept',
    startState: 'inProcess',
    finishState: 'accepted',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
```

(continues on next page)

(continued from previous page)

```
    conditions: null,
    confirm: false,
    confirmMessage: null },
  { name: 'reject',
    caption: 'Reject',
    startState: 'inProcess',
    finishState: 'rejected',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
    conditions: null,
    confirm: false,
    confirmMessage: null },
  { name: 'notify',
    caption: 'To check',
    startState: 'returned',
    finishState: 'canStart',
    signBefore: false,
    signAfter: false,
    roles: [],
    assignments: [Array],
    conditions: [Object],
    confirm: false,
    confirmMessage: null } ],
metaVersion: '2.0.61.16945',
namespace: 'develop-and-test' }
```

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access information about workflow: getWorkflow
```

Getting information about the view form mask: getMask

The request is made along the path <URL сервера>/rest/<название сервиса>/getMask/<имя маски>.

For example:

```
https://localhost:8888/rest/meta/getMask/snils
```

the snils mask object will be requested.

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access information about view mask: getMask
```

Getting a list of available input validators: getValidators

The request is made using the path <server URL>/rest/<service name>/getValidators, for example:

```
https://localhost:8888/rest/meta/getValidators
```

A list of available validators will be requested.

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access information about input validators: getValidators
```

Additional metadata service request parameters

In most requests to the metadata service (meta), in addition to the request itself, you can specify additional parameters. The list of additional parameters that need to be applied to the request begins with the symbol ?. After which the name and value of the parameter are written through the sign ‘‘ = ‘‘, several parameters are separated using ‘‘ & ‘‘.

An example of a GET request to meta/getList with the additional parameter ancestor - filtering by ancestor:

```
https://localhost:8888/rest/meta/listMeta?ancestor=basicObj@develop-and-test
```

The response will return a list of metadata classes that are derived from basicObj.

The example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# accessing the list of metadata classes filtering by ancestor: listMeta
```

Workflows execution service

Workflows is a module built in the rest module, which provides the ability to control and manage the workflows.

The service requires a connection in deploy.json and a mandatory indication of options.dataRepo, options.metaRepo, options.auth and options.workflow, example:

```
"workflows": {
        "module": "modules/rest/lib/impl/workflows",
        "options": {
          "auth": "ion://auth",
          "dataRepo": "ion://dataRepo",
          "metaRepo": "ion://metaRepo",
          "workflow": "ion://workflow"
        }
}
```

All types of authorization are supported, by default is the authorization with credentials.

The service includes three methods:

- GET - no parameters, returns information on the current status in the workflow (possible transitions)
- PUT - transfers the object to the next specified stages of different workflows
- PATCH - performs a forced transition of an object to the specified stages of different workflows

For all the methods, the requests are accepted with the path of the type‘‘<server URL>/rest/<service name>/:class/: id‘‘ to identify the data object.

Specification of methods:

Get the current position of an object in the workflow: GET

GET method is used to get the current position of the object in workflows. The request is made along the path <URL сервера>/rest/<service name>/<class name>/<object id>, for example:

```
https://localhost:8888/rest/workflows/workflowBase@develop-and-test/1
```

In response, an object with the states of workflows will be received:

```
{ stages:
   { 'simpleWorkflow@develop-and-test':
     { stage: 'inProcess',
       since: '2020-05-12T06:36:06.045Z',
       next: [Object],
       workflowCaption: 'Simple WF',
       stageCaption: 'In process' } },
   itemPermissions: { read: true },
   propertyPermissions: {},
   selectionProviders: {} }
```

The example of GET request to workflows in dnt: test/modules/rest/workflows.spec.js

```
/checking workflows service/# accessing workflow statuses of the object: GET
```

Performing an object transition through a workflow: PUT

PUT method of the workflows service performs object transitions through the workflows (including sequential ones). The request is made with the path <server URL>/rest/<service name>/<class name>/<object id>.

The class name is specified with the namespace.

One of the options is passed in the request body:

- an object with attributes - the names of workflows that contain a list of transitions for these workflows.
- a list of strings of the following format <workflow name>.<transition name>

the names of workflows are indicated with a namespace.

Query example:

```
PUT
https://localhost:8888/rest/workflows/workflowBase@develop-and-test/1
body: {
      'simpleWorkflow@develop-and-test': [
        'startCheck',
        'accept'
      ]
    }
```

which is equivalent to:

```
PUT
https://localhost:8888/rest/workflows/workflowBase@develop-and-test/1
body: [
    'simpleWorkflow@develop-and-test.startCheck',
    'simpleWorkflow@develop-and-test.accept'
]
```

The transitions are performed sequentially. For each transition an attempt will be made, even if an error has occurred in one of them.

In response, a list of errors for each transition will be returned:

```
[ { code: 'workflow.ti',
    params: { workflow: 'Simple WF', trans: 'Start checking' },
    message:
     'Невозможно выполнение перехода \'Start checking\' рабочего процесса \'Simple WF\'.' },
{ code: 'workflow.ti',
    params: { workflow: 'Simple WF', trans: 'Accept' },
    message:
     'Невозможно выполнение перехода \'Accept\' рабочего процесса \'Simple WF\'.' } ]
```

or an empty list.

The examples of PUT``requests to ``workflows in dnt: test/modules/rest/workflows.spec.js

```
/checking workflows service/# move the object through workflow: PUT, list body
/checking workflows service/# move the object through workflow: PUT, object body
```

Moving an object to the specified workflow state: PATCH

** PATCH method* * is used to force the object to move to the specified states of business processes. The request follows the path <URL сервера>/rest/<service name>/<class name>/<object id>. The class name is specified with a namespace.

The request body passes an array of target states of workflows, which are specified as strings in the format <business process name>.<state>. The name of the business process is indicated with the namespace. An object sequentially moves to each of the states.

Query example:

```
PATCH
https://localhost:8888/rest/workflows/workflowBase@develop-and-test/1
body: [
    'simpleWorkflow@develop-and-test.canStart'
  ]
```

A list of errors that occurred during the movement or an empty list will be returned as a response.

The example of the PATCH request to the workflows in dnt: test/modules/rest/workflows.spec.js

```
/checking workflows service/# move the object to certain state in a workflow: PATCH
```

- Getting the current location of the object in the workflow: GET
- Performing the object transition through the workflow: PUT
- Moving an object to the specifies state of the workflow PATCH

Creating a service handler in the application

All services are implemented as heirs from Service - the functions of the rest module.

Each service has to export the handler function where the asynchronous method this._routeis implemented, in which the processed methods and paths through the functions this.addHandler returning Promise must be registered. The processing function will get access to optionsand to the data repositories, authorization,

metadata and classes through it (if they are specified in the deploy.json configuration file of the application). And also the processing function will get an object with a typical name req which is the request object of the library express. The data parsed to the object will be located in req.body.

The handler function must return a Promise that resolves to the processing result (for processing in Service modules/rest/lib/interfaces/Service.js), the handler will output it with the code 200 and the content type Content-Type:application/json. If during processing there is an error caught by catch, then for errors related to access control, a response will be returned with the error text and with the code 403 , and for all others, the response code is 500 and the error message Internal server error.

The header can be redefined, for this, in the response you need to give the headersheader type and the object in the data attribute

```
Promise.resolve({headers: ['Content-Type: image/png', 'Content-Length: 107'],
  data: Buffer.from([0x89, 0x50, 0x4E, 0x47, 0x0D, 0x0A, 0x1A, 0x0A, 0x00, 0x00, 0x00, 0x0D, 0x49,
         0x48, 0x44, 0x52, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x01, 0x08, 0x06, 0x00, 0x00, 0x00,
         0x1F, 0x15, 0xC4, 0x89, 0x00, 0x00, 0x00, 0x06, 0x62, 0x4B, 0x47, 0x44, 0x00, 0xFF, 0x00, 0xFF,
         0x00, 0xFF, 0xA0, 0xBD, 0xA7, 0x93, 0x00, 0x00, 0x00, 0x09, 0x70, 0x48, 0x59, 0x73, 0x00, 0x00,
         0x2E, 0x23, 0x00, 0x00, 0x2E, 0x23, 0x01, 0x78, 0xA5, 0x3F, 0x76, 0x00, 0x00, 0x00, 0x0B, 0x49,
         0x44, 0x41, 0x54, 0x08, 0xD7, 0x63, 0x60, 0x00, 0x02, 0x00, 0x00, 0x05, 0x00, 0x01, 0xE2, 0x26,
         0x05, 0x9B, 0x00, 0x00, 0x00, 0x00, 0x49, 0x45, 0x4E, 0x44, 0xAE, 0x42, 0x60, 0x82])
})
```

An example of the implementation of a service that produces lists of objects with filters for the class class_string is in the application develop-and-test. Пример реализации сервиса, выдающего списки объектов с фильтрами для класса class_string есть в приложении develop-and-test. Also, for the study, it is convenient to look at the crud method itself, located at modules/rest/lib/impl/crud.js

```
const Service = require('modules/rest/lib/interfaces/Service');

/**
 * @param {{dataRepo: DataRepository, echoClassName: String}} options
 * @constructor
 */
function listClassString(options) {

  /**
   * @param {Request} req
   * @returns {Promise}
   * @private
   */
  this._route = function(router) {
    this.addHandler(router, '/', 'POST', (req) => {
      return new Promise(function (resolve, reject) {
        try {
          let filter = [];
          if (req.body.string_text)
            filter.push({string_text: {$eq: req.body.string_text}});
          if (req.body.string_miltilinetext)
            filter.push({string_miltilinetext: {$eq: req.body.string_miltilinetext}});
          if (filter.length === 0)
            filter = {};
          else if (filter.length === 1)
            filter = filter[0];
           else
            filter = {$and: filter};
          options.dataRepo.getList(options.stringClassName, {filter: filter}).then(function (results) {
            let items = [];
```

(continues on next page)

(continued from previous page)

```
            for (let i = 0; i < results.length; i++) {
              const props = results[i].getProperties();
              const item = {};
              for (let p in props) {
                if (props.hasOwnProperty(p))
                  item[props[p].getName()] = props[p].getValue();
              }
              items.push(item);
            }
            resolve({data: items});
          });
        } catch (err) {
          reject(err);
        }
      });
    });
  }
}

listClassString.prototype = new Service();

module.exports = listClassString;
```

Request without attributes in the request body

```
curl -X POST -u demo@local:ion-demo https://dnt.iondv.com:8888/rest/string-list
```

will return the entire list:

```
[{"__class":"class_string@develop-and-test",
  "__classTitle":"Class \"String [0]\"",
  "id":"4567a830-b8ea-11e9-9cdf-7bd384cbb7a5",
  "string_text":"example1",
  "string_miltilinetext":"example1",
  "string_formattext":"<p>example1</p>"},
{"__class":"class_string@develop-and-test",
  "__classTitle":"Class \"String [0]\"",
  "id":"4a80bdc0-b8ea-11e9-9cdf-7bd384cbb7a5",
  "string_text":"example1",
  "string_miltilinetext":"example2",
  "string_formattext":"<p>example2</p>"},
{"__class":"class_string@develop-and-test",
  "__classTitle":"Class \"String [0]\"",
  "id":"66dbb3d0-5583-11e6-aef7-cf50314f026b",
  "string_text":"Example of the \"String [0]\" type in the \"Text [1]\" view",
  "string_miltilinetext":"Example of the \"String [0]\"\r\n in the Multiline text [7] view",
  "string_formattext":"Example of the \r\n \"String [0]\" type \r\n in the \r\nFormatted text [7] view"}]
```

A request with an attribute parameter equal to the value in the string_text attribute Example of the \"String [0]\" type in the \"Text [1]\" view

```
curl -X POST -d "string_text=Example of the \"String [0]\" type in the \"Text [1]\"" \
    -u demo@local:ion-demo https://dnt.iondv.com:8888/rest/string-list
```

will return objects that meet the condition:

```
[{"__class":"class_string@develop-and-test",
  "__classTitle":"Class \"String [0]\"",
  "id":"66dbb3d0-5583-11e6-aef7-cf50314f026b",
  "string_text":"Example of the \"String [0]\" type in the \"Text [1]\" view",
  "string_miltilinetext":"Example of the \"String [0]\"\r\n in the Multiline text [7] view",
  "string_formattext":"Example of the \r\n \"String [0]\" type \r\n in the \r\nFormatted text [7] view"}]
```


An example of registering a test service, for more detailes, see Registering a service in the application configuration

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "string-list": {
            "module": "applications/develop-and-test/service/String-list",
            "options": {
              "stringClassName": "class_string@develop-and-test",
              "dataRepo": "ion://dataRepo"
            }
          }
```

To implement multipart query processing, for example, for queries containing files, you can use the library multipart.js (rest/backend/multipart.js) of the REST module. There is an example of implementation in the CRUD service:

```
function reqToData(req) {
    return multipart(req).then(data => data || req.body);
}
```

The library util.js (rest/backend/util.js), which ensures correct actions when working with files and file storage, also serves this purpose , an example from CRUD:

```
reqToData(req)
  .then(data => <util.js.>prepareUpdates(options, data, cm, req.params.id))
```

Service registration in application configuration

To connect services in the application, they must be configured in the global settings of the rest module in the application's deploy.json file. Example:

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "simple": {
            "module": "applications/develop-and-test/service/SimpleRest"
          },
          "string-list": {
            "module": "applications/develop-and-test/service/String-list",
            "options": {
              "stringClassName": "class_string@develop-and-test",
              "dataRepo": "ion://dataRepo"
```

(continues on next page)

(continued from previous page)

```
    }
  },
  "crud": {
    "module": "modules/rest/lib/impl/crud",
    "options": {
      "auth": "ion://auth",
      "dataRepo": "ion://securedDataRepo"
    }
  }
```

The path to service registrations in the file deploy.json - modules.rest.globals.di. Next, the name of the service which will be available at https://domain.com/rest/serviceNameis specified , where serviceName is the name of the service, specified in di, for example simple or string-list in the example above .

In the module attribute the path to the js-file with the service handler is specified with the path relatively to the framework root. The handler can be either in the application or in any module or framework, including standard rest module handlers.

The options parameter the specific settings of the service are indicated. For example, for the crud service, the following settings are indicated:

- in the dataRepo field - data repository with access control used for operations with objects
- in the authfield, the authentication component used to get the current user account is specified.

And for the string-list service the following settings are specified:

- in the dataRepo field - the data repository used for data retrieval
- in the field stringClassName - the class name of the received objects, in this case the class class_string@develop-and-test will be passed to the getList method of the data repository

```
options.dataRepo.getList(options.stringClassName, {})
```

Service requests using the example of tests

Examples of requests to the main REST services can be found in the tests prepared for these services. Tests are written for mocha and are located in the app and develop-and-test repository.

To run the tests, you need to install the request-promise-native and mocha``packages , for example, by running ``npm install in the folder develop-and-test/test.

First you need to configure the server and user parameters in develop-and-test/test/modules/rest/config.js.

The launch is done by calling mocha/lib/cli/cli.js for the test script, for example:

```
node node_modules/mocha/lib/cli/cli.js modules/rest/echo.spec.js
```

You can run all the tests in the folder:

```
node node_modules/mocha/lib/cli/cli.js modules/rest/*.spec.js
```

It is also possible to run individual unit tests:

```
node node_modules/mocha/lib/cli/cli.js -g "<ЧАСТЬ НАЗВАНИЯ ТЕСТА>" modules/rest/*.spec.js
```

for example:

```
node node_modules/mocha/lib/cli/cli.js -g "creating an object" modules/rest/*.spec.js
```

REST module has several embedded services designed to implement typical operations with the module:

- Acceptor Service provides mass creation of objects.
- Token Service provides the issuance of a token for an authorized user.
- CRUD Service CRUD service for system objects.
- Meta Service provides access to the metadata interface.
- Workflows Service provides the ability to monitor and manage the workflows.

The development and connection of user services is also supported.

Developing a handler in the application more details.

Registering a service in the application configuration more details.

Module is an independent component with necessary or additional functionality and structured according to certain rules.

Name	Description
Registry	Module for data operating based on the metadata structure.
Report	Module for generating analytical reports.
Gantt-chart	Module for displaying data with dates.
Portal	Module for displaying arbitrary data templates.
Dashboard	Module designed to display brief information in the form of blocks.
Ionadmin	Administrative module for assigning rights.
Account	Personal account module that forms the display logics.
Soap	Module for the requests to services.
Image-storage	Module for image preview.
REST	Module providing data operation of IONDV apps through REST API.

ГЛАВА 5

5. Creating ION model project

5.1 Setting up the ION development environment

ГЛАВА 6

Functionality of IONDV. Framework and its modules

6.1 IONDV. Framework

IONDV. Framework provides the following functionality:

- descriptive metadata into the data storage structure in the DBMS;
- functionality to work with various DBMS (ORM technology);
- authorization in a system with different policies, by default oath2, with an open, configurable API for connecting passport library authorization modules which provides up to 500 different authorization policies;
- securing access to data - static securing to data types, to navigation, to stages of business processes, to actions on a form; dynamic securing- through the conditions in the data that the profile of the current user must correspond to (belonging to the unit or organization specified in the object, group or other conditions); through url; providing exceptions in authorization and security by url or for a special user;
- connecting modules that provide additional functionality and are implemented through access to the kernel interfaces (APIs) ;
- providing import, export of data in the system, metadata and security from files;
- providing interaction with the file system for storing data, including external file storages, such as nextcloud;
- calculating values with formulas and caching this data;
- providing eager loading and data filtering in connected collections;
- caching requests and sessions in memcached, redis;
- scheduled jobs;
- notifying users about events.

6.2 Modules

Additional functionality is implemented by standard modules.

6.2.1 Data accounting module - registry:

- hierarchical display of navigation;
- displaying lists of data objects according to navigation conditions, filters, search results;
- the ability to create objects;
- display of unified forms of objects with the ability to edit, delete, modify work-flows, implement the conditions for displaying and overloading the presentation of a form in a business process;
- display of various types of attributes, including related in the form of tables or links, geo objects (including search for coordinates by address);
- displaying data according to their semantics (the terms of changes);
- the ability to change the display and interaction with the attributes of objects through custom HTML templates that receive data by REST-API;
- preparation of printed forms in docx and xlsx format based on lists or object data;
- display of user notifications;
- the ability to implement your own action buttons with server data processing.

6.2.2 Reporting and Analytics Module - report:

- formation of calculated forms, with the ability to filter by values;
- data filtering;
- mathematical operations on data;
- pivot tables;
- REST API to report data.

6.2.3 Display of data with geo-coordinates – geomap:

- data layers implementation with filtering by conditions;
- ability to set data view icons according to data type;
- display a pop-up window with brief information on an object;
- display of a template of detailed information on an object;
- search for objects;
- arbitrary boundary filtering;
- zoning and filtering by district boundaries;
- report module data connection, including the calculated data for the region.

6.2.4 REST and SOAP integration modules with standard APIs and user security:

- various custom types of authorization: in the header, token (inclusion of the service receiving a token after authorization in the header), without authorization;
- receiving lists of objects of each type with different filtering conditions;
- CRUD service for any data type;
- work-flow transition service;
- metadata retrieval service;
- the ability to connect arbitrary custom processing services.

6.2.5 Dashboard module - dashboard:

- ensuring of the formation of information blocks with digital and graphic data;
- allows to adjust several groups of views and customize them for each user.

6.2.6 Administration module - ionadmin:

- provides user management, rights and roles control, user blocking;
- generation of security keys (tokens) for integration services;
- monitoring of key server resources (using the dashboard module);
- analysis of slow DBMS queries;
- scheduled tasks setting;
- tracking of system objects changes;
- data and metadata backup;
- recalculation of semantics and formulas caches;
- notification management.

6.2.7 Custom webpage creation module - portal:

- registration of arbitrary pages at the processing address (route);
- registration of static content;
- security access management;
- support for rendering pages from EJS templates.

6.3 The IONDV. Studio application for metadata creation:

- creation of navigation;
- creation of class structures;
- creation of views for classes;
- creation of work-flows;

- basic application setup;
- export and import of metadata;
- work with project files in standalone mode;
- online work with several projects hosted in the browser repository.

6.4 License Contact us English

Эта страница на Русском

ГЛАВА 7

IONDV. Framework

7.1 Description

IONDV. Framework is an opensource framework based on node.js for developping business web applications on the basis of metadata in the JSON/YAML format and standalone functional modules. Visual editor Studio allows to create applications using the "no code" technology and build applications with REST-API web services(module rest). The key module registry is the universal tool for presenting and editing data and processing it by workflows.

See the video that shows the technology of developing and building an application.

7.2 Free Demos

Check out our demo apps right now:

- Studio is an IONDV. Framework specialized IDE that helps you to speed and simplify the development of applications on the IONDV. GitHub Repo. Tutorial "How to create an app in IONDV. Studio"<https://github.com/iondv/nutrition-tickets/blob/master/tutorial/ru/index.md>._
- Telecom (in Russian) is a web application based on IONDV. Framework for accounting, storage, and display the data on the availability of communication services (Internet, mobile communications, television, mail, etc.) in the settlements of the region. 'GitHub Repo<https://github.com/iondv/telecom-en>'_
- DNT is an application for developing and testing the functionality of the framework, in which each accounting entity represents a metadata type, for example, the "string" class, or the "collection" class. This allows you to explore the capabilities of the framework through the application. GitHub Repository.
- War Archive (in Russian) - is the IONDV. Framework web-application designed to store, group and demonstrate the data based on archival documents about Great Patriotic War (World War II). 'GitHub Repo'_.

- Project Management (in Russian) - is a web application for organizing project activities of regional public authorities, which is aimed to monitor the results, to comply with deadlines and reduce the timing of achievements, to use effectively time, human and financial resources, making timely and informed management decisions. 'GitHub Repo'_
- CRM - coming soon on GitHub.

The login for access is - demo and the password is - ion-demo. No registration required.

7.3 Typical applications

Framework is a constructor for web applications of any specificity, since the target area is determined by the structure of metadata describing the behavior of the application. For example, you can create such applications as:

- CRM for customer relationship management;
- accounting and management of enterprise resources;
- automatization of company workflows and document management;
- collecting and storing any data, for example equipment metrics tracking (IoT);
- representation of data in the form of portals;
- REST-API for SPA applications;
- REST-API and background for mobile applications;

7.4 Functional features

IONDV. Framework provides the following functionality:

- descriptive metadata into the data storage structure in the DBMS;
- functionality to work with various DBMS (ORM technology);
- authorization in a system with different policies, by default oath2, with an open, configurable API for connecting passport library authorization modules which provides up to 500 different authorization policies;
- securing access to data - static securing to data types, to navigation, to stages of business processes, to actions on a form; dynamic securing- through the conditions in the data that the profile of the current user must correspond to (belonging to the unit or organization specified in the object, group or other conditions); through url; providing exceptions in authorization and security by url or for a special user;
- connecting modules that provide additional functionality and are implemented through access to the kernel interfaces (APIs) ;
- providing import, export of data in the system, metadata and security from files;
- providing interaction with the file system for storing data, including external file storages, such as nextcloud;
- calculating values with formulas and caching this data;
- providing eager loading and data filtering in connected collections;
- caching requests and sessions in memcached, redis;
- scheduled jobs;

- notifying users about events.

Framework structure: core + metadata + modules = application

At the picture: - ioncore - the core of the application in the form of IONDV. framework - * meta class*, meta view, meta navigation, meta workflow, meta security - functional metadata of the application: structures, views, navigation, workflows, and security respectively - registry module - supported functional modules, such as the registry module for viewing and editing data. Additional meta types and modules are listed below. They represent additional functionality and are applied in accordance with the application. Check for application dependencies in the package.json.

Since the application is a meta description of its behavior in JSON (YAML) format files, functional code and HTML templates that extend typical functionality, it is convenient to work with it through the git version repository. See examples on Github

You can find out more about the functionality of the framework and its modules.

7.5 Quick start

You can get access to the already built applications deployed on cloud servers or explore the different ways at the IONDV.Framework site, including:

- installer for Linux operating system installer
- clone the application repository and install all the components (see instructions below)
- docker-containers with already built applications

7.6 System environment

The framework is launched in the Node.js runtime. Version 10.x.x.

To store the data, you need to install and run MongoDb of version later than 3.6.

7.7 Installer

You can use IONDV. Framework apps installer iondv-app, which requires node.js, mongodb and git installed. During the installation, all other dependencies will be checked and installed, and the application itself will be built and launched.

Install in one command:

```
bash <(curl -sL https://raw.githubusercontent.com/iondv/iondv-app/master/iondv-app) -t git -q -i -m␣
↪localhost:27017 develop-and-test
```

Where the parameters for iondv-app localhost: 27017 is the MongoDB address, and develop-and-test is the name of the application. After starting, open the link 'http://localhost:8888', back office account: demo, password: ion-demo.

Another way is to clone - (git clone https://github.com/iondv/iondv-app.git) and install the application using the command bash iondv-app -m localhost:27017 develop-and-test.

You can also build the application in docker containers, then from the environment you only need docker and the mongodb DBMS in the docker container. More details on the IONDV. Framework application builder page iondv-app

7.8 Build application from the repository

7.8.1 Global dependencies

To build all the components and libraries of the framework, you need to install the following components globally:

- package node-gyp npm install -g node-gyp. For the Windows operating system, it is necessary to additionally install the windows-build-tools package npm install -g --production windows-build-tools.
- Gulp <http://gulpjs.com/>‘_installation package ‘‘npm install -g gulp@4.0‘. 4.0 is supported version of Gulp.
- for versions 3.x.x and earlier of the IONDV. Framework the manager of the frontend libraries packages Bower is required‘‘npm install -g bower‘‘. It’s not required for versions 4.x.x and later .

7.8.2 Manual core, modules and application installation

Let’s take the develop-and-test application as an example. Find the application‘‘develop-and-test‘‘ in the repository. Check the dependencies specified in the file package.json.

```
 "engines": {
  "ion": "1.24.1"
},
"ionModulesDependencies": {
  "registry": "1.27.1",
  "geomap": "1.5.0",
  "graph": "1.3.2",
  "portal": "1.3.0",
  "report": "1.9.2",
  "ionadmin": "1.4.0",
  "dashboard": "1.1.0",
  "lk": "1.0.1",
  "soap": "1.1.2",
  "gantt-chart": "0.8.0"
},
"ionMetaDependencies": {
  "viewlib": "0.9.1"
  "viewlib-extra": "0.1.0"
```

- Install the core, its version is specified in the "engines": {"ion": "1.24.1"} parameter. Copy the URL of the core repository and execute the command git clone https://github.com/iondv/framework. Go to the core folder and switch the tag of the version number git checkout tags/v1.24.1. Since compatibility is ensured at the metadata level, and new versions were released due to changes in building technology, you can use the latest version, for example 4.0.0.
- Further, install the modules listed in the‘‘”ionModulesDependencies”‘‘ parameter. Navigate to the module folder executing the cd modules command. Clone modules from the "ionModulesDependencies" list, for the registry module the command is git clone https://github.com/iondv/registry. Go to the folder of the installed module and switch the tag of the version number git checkout tags/v1.27.1. Repeat for each module. For most applications, you can use the latest core compatible modules.
- The installation of the application is carried out in the applications folder. If you are in the modules folder, move to the folder using the command cd ..\applications. Install by cloning the repository using the command git clone https://github.com/iondv/dnt_ru.

- Finally, install all necessary applications listed in the "ionMetaDependencies" parameter in the applications folder. Make sure that you're inside this folder. Clone the dependencies in ionMetaDependencies. For the‘‘viewlib‘‘application execute the command‘‘git clone https://github.com/iondv/viewlib‘‘. Go to the folder of installed application and switch to the tag of the version number git checkout tags/0.9.1. Repeat for each application.

7.8.3 Build, configure and deploy the application

Build of the application provides installation of all dependent libraries, importing data into the database and preparing the application for launch.

Create the configuration file setup.ini in the /config folder of the core with cloned framework to set the main parameters of the application environment.

```
auth.denyTop=false
auth.registration=false
db.uri=mongodb://127.0.0.1:27017/db
server.ports[]=8888
module.default=registry
fs.storageRoot=./files
fs.urlBase=/files
```

Open the file and paste the text above. The main parameter is db.uri=mongodb://127.0.0.1:27017/iondv-dnt-db. It shows the base name that we use for the application. The DB will be created automatically.

Set the NODE_PATH environment variable to the path to the core of the application using command set NODE_PATH=c:\workspace\dnt``for Windows and ``export NODE_PATH=/workspace/dnt for Linux, where workspace\dnt is the folder with the cloned framework.

At the first launch, you need to run npm install. It will install the key dependencies, including the gulp task runner locally.

Next, execute the command to build the application gulp assemble.

If you want to implement the data import in your project, check the folder data in the application and execute the command: node bin/import-data --src ./applications/develop-and-test/data --ns develop-and-test

Add the admin user with the 123 password by executing the command node bin/adduser.js --name admin --pwd 123.

Add admin rights to the user executing the node bin/acl.js --u admin@local --role admin --p full command.

7.8.4 Start

Launch the application from the core folder using command npm start or node bin\www .

Open the link http://localhost:8888 in a browser and log in. 8888 is a port in the server.ports parameter of the launch configuration.

7.9 Docker

Follow these steps to deploy docker container on the example of the develop-and-test application:

1. Run mongodb DBMS: docker run --name mongodb -v mongodb_data:/data/db -p 27017:27017 -d mongo

2. Run IONDV. develop-and-test docker run -d -p 80:8888 --link mongodb iondv/develop-and-test.
3. Open the link ‘‘http://localhost‘‘in a browser in a minute (the time required for data initialization). For log in use the standard login: demo, password: ion-demo

ГЛАВА 8

Documentation

The IONDV.Framework documentation is available in two languages — english and russian.

ГЛАВА 9

Links

Links for additional information on developing applications using the IONDV.Framework.

- Documentation
- Framework Homepage
- Feedback on Facebook
- Video tutorials on Youtube

ГЛАВА 10

License Contact us English