IONDV Documentation

Release latest

IONDV LLC and The IONDV Community

Aug 02, 2020

Оглавление

ГЛАВА 1

1. Развертывание системы

1.1 Шаг 1 Установка окружения

1.1.1 Предыдущая страница: Оглавление

Окружение - это список программ необходимых для запуска платформы с приложением:

- СУБД MongoDb версии 3.6.
- Среда разработки Node.js версии 10.x.x.

СУБД

1. Необходимо установить СУБД MongoDB. Проверенная версия 3.6.9 и 4.0.0.
2. Далее создаем папку data на диске C: и в ней подпапку db.
3. Для запуска базы данных переходим в папку расположения MongoDb, далее в папку server\bin и запускаем файл mongod.exe. Если есть необходимость использовать каталог с БД отличный от c:\data\db, тогда файл mongod.exe необходимо запустить с параметром --dbpath после которого указать путь к каталогу.

Среда выполнения Node.js

Node.js - является средой, в которой осуществляется выполнение компонентов.

1. Необходимо установить среду разработки Node.js. Проверенная версия node.JS 10.14.2.
2. По умолчанию установщик сам прописывает пути к Node.js в PATH, а также устанавливает менеджер пакетов npm.

Установка глобальных зависимостей

Установливайте глобальные зависимости в командной строке cmd.exe, запущенной от имени администратора, после установки node.js.

NB: команда node -v - показывает версию node.js.

1.1.2 Установка среды сборки под Windows

1. Установите глобально пакет node-gyp командой npm install -g node-gyp необходимый для сборки различных библиотек.
2. Для работы библиотеки под операционной системой семейства Windows дополнительно необходимо установить пакет windows-build-tools - npm install -g --production windows-build-tools.

1.1.3 Пакет сборщика проектов

Для организации тестирования и сборки дистрибутивов при разработке используется Gulp. Установите глобально командой npm install -g gulp@4.0. 4.0 - поддерживаемая версия Gulp.

1.1.4 Установщик фронтенд библиотек

Для установки библиотек фронтенд используется bower. Установите глобально командой npm install -g bower.

1.1.5 Следующая страница: Установка ядра, модулей и приложения

1.2 Шаг 2 Установка ядра, модулей и приложения

1.2.1 Оглавление

Предыдущая страница: Установка окружения

Клонирование приложения и его компонентов

NB: пути не должны содержать русских букв и пробелов. Мы советуем размещать приложение в c:\workspace.

Рассматриваем формирование проекта с модулями на примере приложения develop-and-test.

1. Находим приложение в репозитории github. Набираем искомое приложение develop-and-test в поле поиска и переходим на него.

2. Переходим в репозиторий файлов на ветку версии.
3. Открываем файл package.json в котором смотрим зависимости.

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

1. engines": "ion": 3.0.0 - версия ядра 3.0.0.
2. ionModulesDependencies - список модулей и их версий.
3. ionMetaDependencies - список других метаданных, необходимых для проекта, в данном случае исключение viewlib - библиотека представлений.

NB: для переключения на tag номера версии - смотрите версии в файле package.json.

Получение репозитория ядра

Ядро находится в репозитории framework. На главной странице есть поле с путем к репозиторию.

1. Запустите командную строку от имени администратора.
2. Скопируйте адрес репозитория, перейдите в папку workspace командой cd c:\workspace и выполните команду git clone https://github.com/iondv/framework. Эта команда создает папку framework и в неё клонирует репозиторий.

Получение модулей

1. Переходим в папку модулей командой cd framework\modules.
2. Для каждого модуля из списка package.json в свойстве ionModulesDependencies - находим репозиторий модуля среди группы модулей https://github.com/iondv/ION-MODULES.
3. Клонируйте все модули из списка ionModulesDependencies командой git clone https://github.com/iondv/registry.
4. Перейдите в папку установленного модуля, переключитесь на tag номера версии git checkout tags/v1.27.1. Например 1.27.1 - это номер версии модуля registry.
5. Повторите для всех модулей.

Получение приложения

1. Переходим в папку приложения. Если вы находитесь в папке модулей выполните команду cd ..\applications.

2. Далее вернитесь на страницу репозитория develop-and-test, скопируйте путь и клонируйте его командой git clone https://github.com/iondv/develop-and-test.
3. Перейдите в папку установленного приложения, переключитесь на tag номера версии git checkout tags/v1.17.0.
4. Установка зависимостей в ionMetaDependencies осуществляется в папку applications, для установки необходимо убедиться, что находитесь в папке приложений. Клонируем приложения из списка в параметре ionMetaDependencies. Для приложения viewlib клонируйте командой git clone https://github.com/iondv/viewlib.
5. Перейдите в папку установленного приложения, переключитесь на tag номера версии git checkout tags/v0.9.1. Повторите для каждого приложения.
6. Приложение скомпоновано.

NB: мы советуем создать для него проект в IDE, например Visual Studio Code и в нём создать конфигурациионный файл.

Конфигурациионный файл

Конфигурационный файл служит для задания основных параметров окружения приложения и настройки дополнительных параметров запуска.

1. Создайте конфигурационный файл setup с расширением ini в папке config.
2. Открываем файл в редакторе и вставляем содержимое.

Самый главный параметр - db.uri=mongodb://127.0.0.1:27017/db. Он указывает на название базы которую мы будем использовать для приложения. База данных будет создана автоматически.

Следующая страница: Шаг 3 Сборка, развертывание и запуск

1.3 Шаг 3 Сборка и запуск приложения

1.3.1 Оглавление

Предыдущая страница: Шаг 2 Установка ядра, модулей и приложения

Для всех дальнейших команд, необходимо запустить командную строку от имени администратора.

Перейдите в папку приложения cd c:\workspace\framework и задайте переменную окружения NODE_PATH равной пути к приложению. Для Windows команда - set NODE_PATH=c:\workspace\framework, для Linux - export NODE_PATH=/workspace/framework.

Сборка приложения

Сборка приложения обеспечивает установку всех библиотек, импорт данных в базу данных и подготовку приложения для запуска.

1. При первом запуске необходимо выполнить npm install - она поставит ключевые зависимости, в том числе локально сборщик gulp. Убедитесь, что версия Gulp - 4.0. Эта команда ставит все библиотеки из свойства dependencies файла package.json ядра.
2. После этого, а также все последующие разы выполняйте команду сборки приложения gulp assemble.

NB: Убедитесь, что стоит переменная окружения NODE_PATH, запущена база MongoDB, Gulp установлен глобально и локально и его версия 4.0.

1. Перед непосредственным запуском приложения необходимо добавить базового пользователя для входа. Откройте программу Mongo Compass и в базе данных найдите таблицу ion-user. Удалите все записи, которые увидите там. Далее вернитесь в консоль и выполните указанные ниже команды. Добавьте пользователя admin с паролем 123 командой node bin/adduser.js --name admin --pwd 123. Добавьте пользователю права администратора командой node bin/acl.js --u admin@local --role admin --p full.

Запуск приложения

После окончания сборки можно запускать приложение. Убедитесь, что стоит переменная окружения NODE_PATH. Без этого система выдаст ошибку, об отсутствии компонентов.

Запуск системы осуществляется командой npm start, альтерантивой является запуск node bin/www.

После запуска системы, откройте браузер с адресом http://localhost:8888 и авторизуйтесь в приложении, где 8888 - порт указанный в параметре server.ports конфигурации запуска.

Следующая страница: Описание системы - схема метаданных

ГЛАВА 2

2. System description

2.1 Metadata structure

2.1.1 Scheme of the main types of metadata

Index

Previous page: Step3 Building and running

Metadata (Meta) - 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.

Types of meta files

1. Meta class
2. Meta view
3. Meta navigation: meta navigation node, meta navigation section
4. Meta report
5. Meta admin
6. Meta work-flows
7. Geometa
8. Meta security

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.

> Meta class + Class attributes = Display attributes on the form (the user interface)

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.

Name of meta files:

Meta class	Meta view	Meta navigation
Located in meta directory and composed 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 is composed of the "name" + .section.json and is located in navigation directory. Example: Example: workflow.section.json.

The next page: Meta class - general part

2.1.2 Meta class - general part

Index

The previous page: Schema of the main types of metadata

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

Field description

Identifier	Name	Acceptable values	Description
"isStruct"	**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 attributes**	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"	**Version**	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 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"	**Time tag of created objects**	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"	**Time tag of committed changes**	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"	**Tag of the user who created the object**	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"	**Tag of the user who changed the object**	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

The next page: Meta class - the attribute part

Licence   Contact us   English

2.1.3 Meta view - general part

Index

The previous page: Attribute types in the meta class

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

List view

List view - allows to display the class objects in the form of a list.

JSON

Field description

Field	Name	Acceptable values	Description
"columns"	Columns	Array of objects	Columns of class attributes, described the attribute part of the meta view.
"styles"	**Highlight color 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"	**Search is available**	Logical	Allows or denies displaying the search form.
"pageSize"	Number of object per page	Positive integer	Specifies the number of objects on a single page by default.
"useEditModeEdit"	Edit form for detalization	Logical	Allows or denies the use of the edit form for data detalization of a class object.
"version"	**Version**	String	Metadata versions.
"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

Create and edit view - allows to create and edit the class objects.

JSON

Field description

The next page: Meta view - attribute part

2.1.4 Meta navigation

Index

The previous page: Meta view - view types

Meta navigation - adjusts the position of elements in the navigation block. Meta navigation is divided into meta navigation node and meta navigation section.

Meta navigation section

Meta section navigation consists of "name" + .section.json in the navigation directory. For example: workflow.section.json.

Meta navigation node

Meta navigation node consists of:

- For the first-order navigation nodes - those nodes that are directly in the navigation section: "code" + .json in a directory whose name is the same as the file name of the navigation section to which the navigation node belongs.

Example: In the navigation directory there is a file of a navigation section - simpleTypes.section.json. And there is a navigation node - classString.jsonin the simpleTypes directory. The navigation node file will have a path: navigation\simpleTypes\classString.json.

- For second order navigation nodes - those nodes that are of the group type (a special type of navigation nodes, the "type" field in which contains the value 0). The difference is that the "code" field of such nodes is composite and consists of the "code" field of the group and personal name.

Example: navigation\relations\classReference.refBase.json. This is the file of the navigation node refBase, which is located in the classReferense of the relations navigation section.

The next page: Meta section navigation

2.1.5 Meta workflow

Index

The previous page: Meta navigation - Sample conditions

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

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 business process statuses
"metaVersion"	Versioning	Metadata versions.

“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

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.

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:

The next page: Workflow statuses

Licence   Contact us   English

2.1.6 Meta security

Index

The previous page: Workflow transitions

Description

Meta security - configures the security rights of the system objects. It can be classified into two types: static security and dynamic security.

Static security - configures the access rights to the system objects for the particular role.

Dynamic security - configures the access rights to the system objects for the particular user, under some conditions, while group dynamic security is the access rights for the security group.

You can configure the dynamic security in the deploy.jsonand the acl/resources-and-roles.yml. But for the static security use the acl/resources-and-roles.yml file exclusively.

How to form resource identifier?

- navigation node - n:::namespace@code
- class - c:::classname@namespace
- object - i:::classname@namespace@id
- attribute - a:::classname@namespace.propertyname
- geometa:
  - navigation node: geonav:::code of the node@namespace
  - layer: geolayer:::code of the layer@namespace
  - data: geodata:::code of the layer@namespace@query index
- paths (of modules):
  - portal module: sys:::url:portal/*
  - geomap module: sys:::url:geomap/*

Types of rights

“Read”

read - is the right to view information about class objects. It sets the permission to view the class objects as “read-only” and prohibits the creation/editing of objects.

“Write”

write - is the right to create the class objects. It sets the permission to create new class objects and prohibits editing of existing ones.

“Use”

use - is the right to create the class objects. It sets the permission to create class objects and to use them in the references and collections.

Without use - references are also displayed in the collections. If there is read, but no use, then it is impossible to select an object and place it in the collection.

“Delete”

delete - is the right to delete the class objects.

“Full”

full - is the right of full access to the class objects.

If the project in stakeholders.id has a value associated with the current user (pulling an organization as a global user role is set), then you should consider the current user PROJECT_BENEFITIAR and check the rights to the pm:project resource - these rights will be the rights to the project.

pm:project - is a kind of virtual security resource. It is necessary to abstract the access settings from the checked object for different roles. You can specify different resources for one class and vice versa.

If you did not specify a resource, then the rights to the class object will be checked. Then this role can be used as static, which means, to issue static rights dynamically.

When specifying sids, each level of nesting arrays of values changes the type of operation AND/OR. At the first level, the OR is applied.

1. Register a user with full admin privilege - admin.
2. As admin in registry in the Security, Divisions section set up a hierarchy of divisions (division code = security identifier).
3. Register a user without full admin privilege - user.
4. As admin in registry in the Security, Divisions section create an Employee, specify the User with no rights in its User attribute. Bind the employee to the subordinate division.
5. Connect as user - you have no rights.
6. Connect as admin- give the rights to arbitrary classes and navigation nodes to roles, corresponding to the highest division.
7. Connect as user - you have an access to all objects of the division.

8. Similarly, we check the rights throughout the hierarchy of divisions.

Example of the configuration in deploy.json

How to display attributes and objects in accordance with specified rights?

The class [Projects] contains the attribute of the “Collection” type - [Events]. If the [Events] class does not have “read” access, then the attribute of this class is not displayed on the view form of the [Projects] class.

If there is dynamic security for a class, then whether you have read access to the class [Events] or not - the attribute on the form of the class[Projects] will be displayed, but the event objects will be displayed only if you have rights.

NB: it is necessary to set both static and dynamic securities for a class by the attribute reference to display the attribute and objects.

If there is a static “read-only” right for a class, the user will see all objects of this class, regardless of the dynamic rights. In addition, a sample of objects will be made. Objects that are configured for dynamic security will be displayed to the user in accordance with their settings.

The next page: Meta report

Licence   Contact us   English

2.1.7 Meta report

Index

The previous page: Meta security

Description

Meta report - is used to build a data minethat contains analytical information on data from the meta. The information is organized in the form of tables. In the meta of report module, the data sources are indicated, on the basis of which the information is generated to build a report. Further the report table columns are formed, indicating the resource for the data from the meta classes of the system.

Meta report is located in the bi folder of the project in the YML format.

NB: Definition of “Data mine”

YML example

Example description

Test report contains data from the “sourceClass” class. The data source “dataSource” retrieves data from the corresponding meta class specified in the results: property. Then the “test” subsection forms and transforms data (based on the data obtained from the source specified in the source: property) for correct display in the report tables. The joins: property sets the attribute, which is the identifier for building the report (in this case, object id).

Next, the system generates a report table, based on the converted data from the source, in the reports: section. The‘‘rangeFilters:‘‘ property contains information on filters that are configured for the report (in this case, you must specify a date range according to the data from the class). In the module, set the range filter by using the query parameters: ?rangeFld[]=0&rangeFld[]=5, where rangeFld is the field we use for searching. If you are searching by date - the date is sent in the locale format, which is transmitted in the http-header `'accept-language'`, or in the format ISO8601. The columns: property allows to form table columns (numbers are actual).

The result: a two-column table (Date and Name) in which class objects from the _”dataSource”_ source are displayed, according to the date filter configured in rangeFilters:. The number of objects in the table will be equal to the number of identifier values configured in the joins: property.

You can see the example of a simple report here.

Configuration of comparison

The configuration of strict comparison within the boundaries of a rangeFilters in the report:

both - both bounds can be equal to the sought values- left border (smaller one) can be equal to the sought values right - right border (bigger one) can be equal to the sought values

If inclusive is nor specified - the comparison is strict at both boundaries.

Hierarchical build

The configuration of the hierarchical build is necessary to process the initial data when building the mine:

- To make in one data source data extraction across the entire hierarchy in the DB
- To display data of the first column with indents depending on the nesting depth

Configuration of the hierarchical build in the data mine:

The "hierarchyBy" config is the objects with the following set of properties: id, parent, level, order.

where id - is the attribute in data, identifying element of the hierarchy

parent - is the attribute in data containing parent id

level - is the attribute in the resulting source where the nesting level of the element will be written

order - is the attribute in the resulting source where the value will be written to organize the hierarchy when displayed on the form.

The objLevel and‘‘objOrder‘‘ field - are the fields to write values (no need to calculate, aggregate etc.).

YML example

NB: Hierarchical build is possible only on the basis of the source and impossible on the basis of the class.

Build algorithm:

1. Create a result source.
2. Make a sample of root elements that have an empty parent field.
3. We sort and write elements in the result source (in the special attribute element_id - the object identifier (id), in level - the value 0, in order - the sequence number of the element in the sample, reduced to a string, supplemented up to 6 characters with leading zeros).
4. Iteratively, we make samples of the levels of nesting (starting from 0), until 0 objects are extracted at the final iteration. Samples are made by combining the initial source with the resulting one by the parent = element_id connection and the constraint level = current level of nesting.
5. At each iteration, we sort and write elements in the result source, and at the same time:
   - write the id of the object in the special attribute element_id,
   - write the current level of nesting in level,
   - in order, write an order concatenation of the parent element and the sequence number of the element in the sample, reduced to a string, supplemented up to 6 characters with leading zeros.

Configuration to hide objects

Configuration to hide all objects if tabular filters are not specified. Apply the "needFilterSet: true" setting to hide all objects when opening a report, until a value is selected from the list in the filter.

How to display sample parameters in the report header by using patterns

YML example

How to style lines of the report by using data

YML example

Use of ComboBox in the parameters and filters

YML example

Configuration of processing filter parameters on the report page

YML example

The year value in the $yearStart attribute is equal to the year value from the date in the :dateSelect attribute.

Paginator "pageSize"

NB: It is used in the reports of the type: list type.

We recommend using the “pageSize”, when the report contains many objects and you need to output the lines page-by-page, not to load the browser with heavy data processing.

YML example

Output line by line

Setting up the line by line output of nested data in the report is configured as follows:

YML example

Incremental load configuration

Specify the ‘‘append: true‘‘property to set the incremental load of data into the source when building the data mine.

It is used to upload the statistics for a day to the mine, so as not to recalculate the entire volume of source data and have a history by periods.

Specificity of object sorting

Taking into account the MongoDB aggregation functional — the sorting is possible only by the resulting fields. This means that for backward compatibility, you should name the result fields that are used for sorting the same way as the fields in the data source.

Example of sorting (the sortproperty):

The next page: Platform configuration - deploy.json

Licence   Contact us   English

2.2 Platform configuration

2.2.1 Configuration file - deploy.json

Index

The previous page: Meta report

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 way to set the configuration via multiple separate files:: doc:Deploy build from separate files <deploy_pulling>

Structure of the deploy.json file:

Field	Name	Description
"namespace"	Project name	The project namespace.
"parametrized": 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"	**Modules settings**	Module configuration settings.

The full example of the deploy.json file

The next page: Dependencies in package.json

Licence   Contact us   English

2.2.2 Dependencies in package.json

Index

The previous page: Configuration file - deploy.json

The package.json file - defines the structure of dependencies and the detailed composition of the system modules.

Specificity of connection using a script

- if there is no slash in the object name - / => “project-management”- substitute 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 “ION-METADATA/viewlib” - the path - //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 value begins with http:// or https:// - then this is the full path to the archive - pull and unzip. Not realized, as dapp does not support working with archives.

Example of the package.json file

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 already built project on the docker.
"bugs"	Bugs	Specifies the link to the application project in GitLab, where issues about bugs are collected.
"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” – registry module • “report” – report module • “rest”: “- REST module • “dashboard” – dashboard module • “geomap” - geo module • “gantt-chart” – gantt chart module • “portal” – portal module
"ionMetaDependencies"	Dependencies of ion metadata	Additional applications to operate the system.
"dependencies"	Dependencies	Other project dependencies.

The next page: Configuration parameters - ini-files

2.2.3 How to configure the parameters?

Index

The previous page: Dependencies in package.json

- 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.

How to configure 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:

Configure the parameters in deploy by use of the environment variable:

In the environment variables for the NODE set the following parameters:

Setting the limit

Setting the limit for switching items in the system menu 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 access rights to the module - the menu item is not displayed to go to the module page. Set the ‘‘auth.checkUrlAccess=true ‘‘ in the ini-file of the project to set the limit setting.

Changing all references to relative ones

To change all references to relative ones, in the ini-file set the following parameter:

If the path is not specified, then it is ‘/’ by default.

Setting the control unit to run tasks on a schedule in admin module

To display the control unit, in the ini-file of the project add the following setting:

This setting enable the scheduler in the process of the web server, which will give the opportunity to manage Jobs from the admin module.

Scheduler — manages task start timers.

Job — specific task run by a timer.

Setting to cache the data 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.

In the ini-file of the project, write the following:

Setting the time limit

connectTimeOut - the maximum time connection time.

operTimeOut - the maximum time to complete an operation.

Setting the minimum password length

You can override the setting for an application in the deploy.json file

The next page: Functionality

Licence   Contact us   English

2.2.4 Authorization and Security Settings

Application configuration options, deploy.json file

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

Password settings and requirements are set in di in auth component configuration of the module. But mostly 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
          "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 imported ones, are automatically set as required for the change. In order to avoid changing passwords during import, the needPwdReset: false parameter 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.

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"
   }
}
```

Settings for the framework and application in ‘config/setup.ini

Settings are used to specify and change the application parameters and initialized at start. Settings take precedence over configuration settings.

Application settings can also be set in environment variables, while environment variables take precedence over 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 redefine them via the platform settings or through environment variables.

Example of the setup file /config/setup.ini in which the values specified in the deploy.json file are redefined.

```
# 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": {
    "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:

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 caching settings and parameters to the deploy.ini file of the project .. code-block:

```
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 in the ini-file of the project, write the following:

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:

2.3 Functionality

2.3.1 Electronic digital signature

Index

Back: Functionality

Description

Electronic digital signature (EDS) - refers to data in electronic form, which is logically associated with other data in electronic form and which is used by the signatory to sign. It is designed 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 is 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 electronic signatures, you should have confirmation of compliance with legal requirements

Work specifics

The EDS utility works on the cryptoPro basis, so it should be installed on the same 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 (the project-management app example):

- 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 BP 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"
      }
    ],
    "conditions": []
  }
```

2.3.2 Module templates

Index

Back: Functionality

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",
```

License Contact us English

2.3.3 Notifications

Index

Back: Functionality

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 sending notifications to mail feature 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.

License Contact us English

Copyright (c) 2018 LLC “ION DV”. All rights reserved.

2.3.4 Printed forms

Index

Back: Functionality

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.

format:

```
${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 the 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}
```

License Contact us English

2.3.5 Scheduled tasks

Index

Back: Functionality

The launch of the scheduled tasks 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 in the ini-file the option jobs. enabled=true

In the second case, task management is possible to implement in a web application. Tasks 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
```

(continues on next page)

(continued from previous page)

```
    "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 task, in this case it should have the run method. A function can also be specified as a starting task. 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": {
       "launch": {
         "sec": 30
       },
       "worker": "dummy",
       "di": {
         "dummy": {
           "executable": "applications/develop-and-test/jobs/dummy"
         }
       }
     }
...
```

2.3.6 Utilities

Index

Back: Functionality

Description

Utilities - are additional programs for more specialized applications.

Core utilities

Core utilities are the most necessary utilities from installation to operating the application. They are stored in the bin folder. Before start the utilities, you must set the environment variable NODE_PATHif it was not set in advance (indicating the core folder). The core contains the following 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 (last 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 tasks
- bin/job.js - to run the task 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 tasks
- 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, the JavaScript language is used with the available functionality of the modules included in the application. :raw-html-m2r:‘
‘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 command should be the last line:

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

Connection 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
}
```

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": [
              {
              "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 configuration in deploy.json

Meta view - Commands

ГЛАВА 3

3. Description of the modules

3.1 Ionadmin Module

3.1.1 Index

Back to: modules

Administration module (Ionadmin) – is used for assigning rights, managing tasks on a schedule and other administrative tasks.

Ionadmin module configuration in the config.json file

3.1.2 Configure slow query recording

Setting up as a modal window on 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" propery is not set or null, then the data will be taken from the table:

```
{
  "profiling": {
```

(continues on next page)

(continued from previous page)

```
    "slowQuery": {
      "sources": null
    }
  }
}
```

If an empty array is set, then there are no sources.

3.1.3 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 made selections are stored in a separate table and do not depend on the current state of the log sources. You can add information by editing them. For example, comments or notes informing whether the problem is solved or not.

3.1.4 DB backup configuration

Setting in the ionmodule/config:

```
"backup": {
    "dir": "../ion-backups",
    "zlib": {
      "level": 1
    }
  }
```

- dir contains the path of the folder in which the node application was launched. "../ion-backups" - by default.
- zlib.level - the level of compression also affects the speed of the archive creation. 3 - by default.
- In addition, it is necessary that the export.js utility with the specified parameters worked correctly on its own.

3.1.5 Security user guide

Security user guide is here.

3.1.6 Licence Contact us   English

3.2 The “Registry” module

3.2.1 Index

Back to: modules

The registry module – is a central module designed specifically for working with data based on metadata structures - including project management, programs, events, etc.

Setting

- DI (treegridController)

Deploy

Setting polling frequency in deploy.json

Setting the frequency of server polling to ensure that the object is not blocked in deploy.json:

Setting “createByCopy”

Configuring the display of the “Create more” button” in deploy.json:

Filters

Learn more about filters here.

Setting filters “help”

“Help” is placed in the template file - modules/registry/view/default/templates/view/list-filter-helper.ejs

Code requirements

Frontend component writing style is here

3.2.2 Licence Contact us   English

3.3 The “report” module

3.3.1 Index

Back to: modules

The report module – is designed for the formation of analytical reports and reference information (on the basis of special metadata) in the form of graphs. Calculations can be performed on a schedule or be initiated by the operator.

Library to build the Pivot type report

The library for building reports of the Pivot type is PivotTable.js (Examples and description: https://pivottable.js.org) The functionality is rich, but at the same time difficult to build reports like in Excel or Word. For now it’s better not to require the functionality of the Word from it when designing a report - it is better to consult with the developers and put a task on the implementation of a Pivot type report.

Metadata

Metadata of the report module (data mine) Comments on the mine design

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 task at the start of the application and then every 6 hours, you need to configure the job as follows:

```
"jobs": {
    "report-builder": {
      "description": "Служба сборки шахт данных модуля отчетов",
      "launch": {
        "hour": 21600000
      }
    }
```

3.3.2 Licence Contact us   English

3.4 IONDV. REST

The IONDV.Framework REST module is used to create web services for various types of data, including only visual data created by designing in IONDV.Studio and is intended 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;
- the rapid development of our own web services, by registering them and writing code on a ready-made data manipulation API for implementing a microservice architecture
- integration of applications created on the ION DV. Framework with other systems using the REST API.

3.4.1 Description and purpose of the module

ION DV. REST - a module that provides work with the data of the ION DV application via the REST API. It is a wrapper for working with data via standard CRUD functions or it connects the application's own services, including those using the core API.

3.4.2 Contents

REST Services

The REST module has several built-in services designed to implement typical operations with the module:

- acceptor - the service provides mass creation of objects
- token - the service provides the issuance of a token for an authorized user
- crud - CRUD service for system objects
- meta - service provides access to metadata interface
- workflows - the service provides the ability to control and manage workflows

The development and connection of user services is also supported.

Service registration in application configuration

To enable services in the app, it's necessary to configure them in the global settings of the rest module in the deploy.json file of the applications. An example is given below.

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "simple": {
            "module": "applications/develop-and-test/service/SimpleRest"
          },
          "string-list": {
            "module": "applications/develop-and-test/service/String-list",
            "options": {
```

(continues on next page)

(continued from previous page)

```
        "stringClassName": "class_string@develop-and-test",
        "dataRepo": "ion://dataRepo"
      }
    },
    "crud": {
      "module": "modules/rest/lib/impl/crud",
      "options": {
        "auth": "ion://auth",
        "dataRepo": "ion://securedDataRepo"
      }
    }
```

The path to the registration service in the file deploy.json is module.reset.globals.di, then specify the name of the service that will be available at https://domain.com/rest/serviceName, where serviceName is the name of the service specified in di, such as in the example above simple or string-list. The attribute ‘‘ module‘‘ indicates the path to the js file with the service handler with the path relative to the root of the framework. The handler can be both in the application and in any module or framework, including sample 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. And for the service string-list:
- 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, {})
```

Built-in “acceptor” service

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 deploy.json application configuration file is required. At the same time, for the service, the repositories dataRepo and metaRepo must be specified in the options. 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 the main ones: doc: access types <../ authorization / index>.

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 ion-converter property, the name of the converter can be passed, which should be used when processing the data of both the request and the response. In this case, 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

The other values must match the class properties, including matching data types. 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
```

Example of a request to create objects for 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"
  }
]
```

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 “token” service

The token service is intended for issuing a token to an authenticated user for its further use in token authentication services.

Available at <server address>/test/token.

The service does not require registration in deploy.json. The service provides the issuance of a token for an authorized user if he has the use rights for the ws:::gen-ws-token resource or has administrator rights. In response to the request, a token of the form e444c69894d2087696e0a6c6914788f67ebcf6ee is returned. The default token lifetime is 100 years.

Example of a request using basicAuth authentication

```
curl -u demo@local:ion-demo https://dnt.iondv.com/rest/token
```

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 requests 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
```

You can add a resource for generating tokens for a role from the command line node bin/acl.js --role restGrp --p USE --res ws:::gen-ws-token (where restGrp is the name of an existing group)

The second way to add rights to a resource is to use the admin console of the ionadmin module, for example, by going to locahost:8888/ionadmin/:

- Select the “Security” navigation item
- Select the “Roles” navigation sub-item
- Select an existing role and click edit or create a new role
- In the role “Access rights” field, 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”

The crud service implements the REST API according to the model of basic CRUD operations (create, read, update, delete).

Available at <server address>/rest/crud.

The service requires registration in deploy.json of the application and requires specifying the data source data Repo in options of the service, as well as the auth authorization source for accessing user data. It is advisable to specify a repository with full security processing as the data repository in order to work out access to objects that take dynamic security into account. For example

```
{
  "modules": {
    "rest": {
      "globals": {
        "di": {
          "crud": {
            "module": "modules/rest/lib/impl/crud",
            "options": {
              "auth": "ion://auth",
              "dataRepo": "ion://securedDataRepo"
            }
          }
```

Authentication is performed through all the main access types.

Example

```
curl -X POST -u demo@local:ion-demo https://dnt.iondv.com/rest/crud
```

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 the main methods:

Creating an object: POST method

The object is created using the POSTmethod, and the code of the class with a namespace is specified, for example,“rest/crud/class_string@develop-and-test“. The object itself is passed in the request body in json format with the mandatory indication of the json content type Content-Type:application/json in the header. You don’t need to specify auto-generated fields.

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/
```

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 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
```

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
```

example of a request to the crud service to get an object with a eagerly 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 issued, 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/
```

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>"},
{"_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 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. Querying 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. Querying 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
```

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": {
    "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 the presence of an object: the HEAD method

The presence of an object is checked by the HEADmethod, and the class code with a namespace and the value of the object key are indicated, 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
```

example of a request to the crud service to check for 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 issued, if the object is not found - 404, if there are no permissions - 403.

Updating an object: PATCH and PUT methods

Object updates are performed using the PATCH or PUTmethod, 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. The object itself is passed in the request body in json format with the mandatory indication of the json content type Content-Type:application/json in the header.

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
```

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 issued, if the object does not find the code - 404, if the processing fails, the code is 500 , if there are no rights - 403.

Example of an object.

```
{
    "_editor": "admin@local",
    "_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 is performed using the DELETEmethod, and the code of the class with the namespace and the value of the object's key 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
```

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 issues the 200response code, 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.

Examples of POST requests with files to CRUD в 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)
```

Workflow execution service “workflows”

workflows - built-in service in the rest module, which provides the ability to control and manage 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 : doc:authorization types <../authorization/index> are supported, by default - authorization with credentials.

The service includes three methods:

- GET - without parameters, returns information about the current status in the workflow (possible transitions)
- PUT - transfers the object to the specified next stages of different workflows
- PATCH - performs a forced transfer of an object to the specified stages of different workflows

For all methods, requests are accepted along the path <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 obtain the current position of an object in workflows. The request is carried out using the path <server URL>/rest/<service name>/<class name>/<object id>, 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: {} }
```

Example of a GET``request to ``workflows in dnt: test/modules/rest/workflows.spec.js

```
/checking workflows service/# accessing workflow statuses of the object: GET
```

Transitioning an object through a workflow: PUT

The PUT method of the workflows service performs object transitions over the DB (including sequential ones). The request is made using the path <server URL>/rest/<service name>/<class name>/<object id>.

The class name is indicated with the namespace.

One of the options is passed in the request body:

- an object with attributes-names of workflows that contain a list of transitions for these workflows.
- a list of lines in the <workflow name>.<transition name> format

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',
```

(continues on next page)

(continued from previous page)

```
    'simpleWorkflow@develop-and-test.accept'
]
```

Transitions are performed sequentially. An attempt will be made for each transition, 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.

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
```

Movement of an object to the specified workflow state: PATCH

The PATCH method forces the object to be moved to the specified states of workflows. The request is made using the path <server URL>/rest/<service name>/<class name>/<object id>. The class name is indicated with the namespace.

The request body passes an array of target states of workflows, which are specified as strings in the format <business process name>.<condition>. 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.

Example of a 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
```

"Meta" metadata publishing service

meta - built-in service in the rest module, which provides access to the meta repository interface in the web service format.

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 : doc:authorization types <../authorization/index> are supported, by default - 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,
   readonly: false,
   indexed: true,
```

(continues on next page)

(continued from previous page)

```
  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,
  unique: false,
  autoassigned: false,
```

(continues on next page)

(continued from previous page)

```
  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 } ] }
```

This 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.

This 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.

This 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,
```

(continues on next page)

(continued from previous page)

```
  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',
  mode: 0 },
{ orderNumber: 30,
  name: 'text_multilinetext',
  caption: 'Multiline text [7]',
```

(continues on next page)

(continued from previous page)

```
  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,
  formula: null,
  definitionClass: 'class_text@develop-and-test',
  mode: 0 } ]
```

This 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.

This 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',
       code: 'class_boolean',
       orderNumber: 0,
       type: 0,
       caption: 'Class "Boolean [10]"',
       classname: null,
       container: null,
       collection: null,
```

(continues on next page)

(continued from previous page)

```
    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:
   { code: 'class_decimal',
    orderNumber: 0,
    type: 1,
    caption: 'Class "Decimal [8]"',
    classname: 'class_decimal@develop-and-test',
    container: null,
    collection: null,
```

(continues on next page)

(continued from previous page)

```
        url: null,
        hint: null,
        conditions: null,
        sorting: [],
        pathChains: [],
        metaVersion: '2.0.7',
        itemType: 'node',
        section: 'develop-and-test@simpleTypes',
        namespace: 'develop-and-test',
        children: [] }
    }
}
```

This 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',
children:
[ { code: 'semantic.semErrCatalog',
    orderNumber: 0,
    type: 1,
    caption:
     'Catalog for checking semantics (string1| |string2| |string3| |date| |integer)',
    classname: 'semErrCatalog@develop-and-test',
```

(continues on next page)

(continued from previous page)

```
  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: [] }]
}
```

This 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.

This 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',
    size: 2,
    maskName: null,
    mask: null,
    mode: null,
    fields: [],
    hierarchyAttributes: null,
```


(continues on next page)

(continued from previous page)


```
    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,
    commands: [],
    orderNumber: 40,
    required: false,
    visibility: null,
    enablement: null,
    obligation: null,
```


(continues on next page)

(continued from previous page)

```
    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: '' }
```

This 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.

This 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 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,
     enableCondition: null,
     needSelectedItem: false,
     signBefore: false,
     signAfter: false,
     isBulk: false } ],
siblingFixBy: null,
siblingNavigateBy: null,
historyDisplayMode: 0,
```

(continues on next page)

(continued from previous page)

```
collectionFilters: null,
version: null,
overrideMode: null,
metaVersion: '2.0.7',
type: 'item',
className: 'class_text@develop-and-test',
path: '' }
```

This 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 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',
     visibilityCondition: null,
     enableCondition: null,
     needSelectedItem: false,
     signBefore: false,
     signAfter: false,
     isBulk: false } ],
siblingFixBy: null,
siblingNavigateBy: null,
historyDisplayMode: 0,
collectionFilters: null,
version: null,
```

(continues on next page)

(continued from previous page)

```
overrideMode: null,
metaVersion: '2.0.7',
type: 'create',
className: 'class_text@develop-and-test',
path: '',
caption: '' }
```

This 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 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.

This 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' } ]
```


This 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,
     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 } ],
metaVersion: '2.0.61',
type: 'item',
className: 'workflowBase@develop-and-test',
path: 'workflows:simpleWorkflow@develop-and-test.canStart',
caption: '' }
```

This 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 simple Workflow workflow object will be requested for the workflowBase object class in the develop-and-test namespace, example of a response:

```
  { 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,
    conditions: null,
    itemPermissions: [Array],
    propertyPermissions: [],
    selectionProviders: [] },
   { name: 'accepted',
```

(continues on next page)

(continued from previous page)

```
    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],
    conditions: null,
    confirm: false,
    confirmMessage: null },
  { name: 'reject',
    caption: 'Reject',
```

(continues on next page)

(continued from previous page)

```
    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' }
```

This 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 carried out along the path <server URL>/rest/<service name>/getMask/<mask name>. For example:

```
https://localhost:8888/rest/meta/getMask/snils
```

the snils mask object will be requested.

This example in dnt: test/modules/rest/metadatasrv.spec.js

```
/checking metadata service/# access information about view mask: getMask
```

Creating a service handler in the application

All services are implemented as heirs from Service - functions of the rest module.

Each service must export a handler function that implements the asynchronous method this._route , in which it is necessary to register the processed methods and paths that return Promise through the functions this.addHandler. The processing function will have access to options, through which - access to data repositories, authorizations, metadata, and classes (if they are specified in the application configuration in the deploy file.json), and also gets an object with the typical name req - which is the request object of the express library . The data already 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])
})
```

Example of implementing a service that outputs lists of objects with filters for the class_string class in the develop-and-test application. It is also convenient to study 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 = [];
            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);
```

(continues on next page)

(continued from previous page)

```
            }
            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
```

Returns 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
```

Returns 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 details 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 , example from CRUD:

```
reqToData(req)
  .then(data => <util.js.>prepareUpdates(options, data, cm, req.params.id))
```

Service request authentication

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 through the administrator's console, go to the navigation item "Web Services Security Keys" of the ionadmin module, for example by going to the address locahost:8888/ionadmin/token

On the "Security Token Generator page":

- Enter the user name in the "User Name" field
- Enter "local" 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 a token is to use the web service of the rest module - token. You can get a token via an authenticated request to the address rest/token. More details: Built-in “token” service.

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
                  }
                }
              }
            }
          }
        }
```

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
```

Request example in dnt: test/modules/rest/gateway.spec.js

```
/Checking rest-api proxy
```

Authorization can be carried out in the following ways:

- Without authorization
- By account
- By token
- Oauth2

Services without authentication

To implement the operation of the service without authentication, it’s necessary to set it to none in the authMode setting in deploy.json

```
{
  "modules": {
    "rest": {
      "globals": {
        "authMode": {
          "echo": "none"
```

A request to the service will not require authentication, an example request is curl https://dnt.iondv.com/rest/echoo

An example of a service request 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
```

Basic Auth authorization request example 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
```

example request with credential authorization 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 with authentication through a token, you have to set the token value for it 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
```

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

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": {
        "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 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
```

Service request example with oauth2 authorization in dnt: test/modules/rest/echooauth.spec.js

```
/Checking echo-oauth service
```

3.5 The gantt-chart module

3.5.1 Index

Back to: modules

The gantt-chart module – is a module designed to display specific types of hierarchical data with dates.

Configuration in deploy.json

Setting the length of the search results

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

Configuration of view types

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 works only for lists in register and gant. As we move to a common syntax, we implement support throughout the core.

The adjustable filter is not applied to the root object explicitly specified via the URL parameter, or selected in the drop-down list. The filter is applied only at the SELECTION of the subnodes.

Sorting when displaying

When displaying the project, the events are sorting by the numEvent attribute at all levels of the hierarchy.

Setting an object selection list to display information

It is used if the filter is configured for a column and allows you not to display all objects at once, but choose from the list. If the value is "rootParamNeeded:true", then a blank screen and a window for selecting a project are displayed.

```
"gantt-chart": {
    "globals": {
      "rootParamNeeded": true
    }
  }
```

3.5.2 Licence Contact us   English

3.6 Soap module

3.6.1 Index

Back to: modules

NB. Soap module does not support GET requests for services. Partly because a SOAP request is transmitted in the request body, and the GET request does not have a body (if you can put it that way). For this reason, you need to send a POST request.

This can be done using the SOAP-UI utility (you can do it also in the browser, but in the request body you need to write a SOAP request that is WSDL-based and quite heavy).

Crud service data structure settings

The types option contains a mapping between a class (full name) and a map of published attributes of this class. In the map, the attribute name is the key, and the value is either a string alias or a Boolean value indicating whether the attribute is included in the schema or not. That means if an alias is specified, then the attribute appears in the schema under this alias, in all other cases except for specifying the false attribute appears under its name.

This setting is used for parsing classes when creating the service data schema, as well as for parsing incoming messages and generating responses. By replacing "normalize" with a function that correctly leads data to the schema.

For collections and references

If the values of collections and links also need to be parsed in another way, the objects that are in these properties can also be described by the map in deploy.json as follows:

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
              }
            }
          }
        }
```

The setting for removing system attributes from a request

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.

3.6.2 Licence Contact us   English

3.7 My account

3.7.1 Index

Back to: modules

Personal account (PA)is used to display various “instrumental” forms, i.e. in the applications, we set up the navigation (or the page-forms), the module reads these data and forms the navigation and the display of these pages.

Module configuration

1. Put the html-page in the application in a certain directory.
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. By 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.

3.7.2 Licence Contact us   English

3.8 Portal module structure

3.8.1 Index

Back to: modules

The portal module – is a module for displaying arbitrary data templates. The portal module performs the function of displaying the design of various information using markup language Markdown and HTML.

Structure

View layer - is responsible for displaying data:

1. Adapter Provider - performs the function of connection between the data and their display on the portal (sets 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 (updated each time).

The logic of displaying data on the portal

3.8.2 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 {
  padding: 5px;
}
...
```

3.8.3 Portal pages

Portal page templates are configured in the portal/view/templates folder.

The location of objects on the page is described in a 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 displaying 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.

3.8.4 Portal navigation

A portal navigation meta is a set of navigation nodes, each of which has a specified resource type.

Example of the navigation section:

```
{
  "code": "main",
  "caption": "Главное меню",
  "itemType": "section",
  "subNodes":["classes", "texts"]
}
```

- code - system object name
- caption - logical object name
- itemType - object display type
- subNodes - 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

3.8.5 Data styling

1. The format of paging information
2. The format of the correct display of the text of errors

1. The format for converting data to portal content

1. The format of the text display

3.8.6 Licence Contact us   English

3.9 The dashboard module

3.9.1 Index

Back to: modules

The dashboard module – is a module designed to display brief information in the form of blocks. The dashboard is based on the widget model.

Module structure

The control panel consists of three basic entities - the manager, the layout and the widget.

Manager

The manager - is the main component of the module which is responsible for creating and initializing widgets, layouts, connecting the panel to other modules.

Layout

Layout - is an EJS template, which defines the layout of the widgets, the parameters for the widget templates, a plugin for managing the layout grid on the client (for example, gridster), shared resources are connected. Basic layouts of the module are located in the /dashboard/layouts. Published from metadata in the /applications/${meta-namespace}/layouts folder. Each layout has a unique ID. When publishing a layout from a meta, a prefix is added to the ID.

When rendering a layout, you must pass an object to the manager.

Widget

Widget - is an object that is located on the layout and interacts with the server via ajax requests.

Basic widgets are located in the /dashboard/widgets. Published from the metadata are located in the folder /applications/${meta-namespace}/widgets.

A widget consists of 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 initial initialization of the widget 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 you publish a widget from meta, a prefix is added to the ID.

When rendering a widget view, you must pass an object to the widget.

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
```

Add the following section in the "modules"of the deploy.json configuration file to load the data from the meta to the dashboard module:

3.9.2 Licence Contact us   English

ГЛАВА 4

4. Creating a model project of ION

4.1 Setting up the ION development environment

ГЛАВА 5

7. Standard functionality extension and development

5.1 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, utilities that are called by a scheduled task, or utilities that are called during a transition in a workflow.

5.1.1 Contents

Utilities for scheduled tasks

Back: Functional utilities of the application

Utilities for scheduled tasks (jobs) are designed to automate the regular execution of certain actions at certain intervals. To do this, each utility must be defined in the deploy.json application in the globals.jobs object, for example:

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

Here the script runs on a schedule applications/khv-ticket-discount/lib/overnightTicketClose.js. launch can be an object containing the following fields: month, week, day, dayOfYear, weekday, hour, min, minute, sec, second - they set the interval between task completion; check - the interval for checking the execution condition, in milliseconds, is 1000 by default. For example, if check is 5000 and sec is 2, the task will be executed only every 10 seconds when the interval between checks coincides with the execution interval If the task execution interval is not set, it will be completed when the application starts and after each verification interval. - time in milliseconds after which a running task is interrupted by timeout;

launch can also be an integer - the interval of the task in milliseconds, while the task 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 can be specified, which will become available in the script through the fields of the object passed as an argument to the main function of the module.

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

Utilities for workflow

Back: Functional utilities of the application

Workflow utilities are designed to automate certain actions when the state of a workflow changes. The utility connects to the application in deploy.json in the global.plugins object, for example:

The wfEvents utility is connected here. The script containing the description of actions when changing the state of the workflow is located on the path applications/sakh-pm/lib/wfEvents.js. In the options field, any variables and their values can be indicated, that will become available in the script through the fields of the object passed as an argument to the main function of the module.

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

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

Utilities for action buttons

Back: Functional utilities of the application

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,
      "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,
```

(continues on next page)

(continued from previous page)

```
    "isBulk": false
  }
 ],
```

There are three buttons available on this form: SAVE, SAVEANDCLOSE и CREATE_INDICATOR_VALUE. Custom buttons must be specified before connecting to the form in deploy.json in the modules.registry.globals.di.actions.options.actionsobject:

```
{
 "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 specify the parameters of the handler module used by the button:

```
{
 "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"
      }
     },
```

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

The contents of the script:

```
/**
```

(continues on next page)

(continued from previous page)

```
 * 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
   * @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!');
        }
```

(continues on next page)

(continued from previous page)

```
        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 web service (rest)

Back: Functional utilities of the application

Web service utilities are designed to implement the processing of various types of server requests. The service connects to the application in deploy.json in the modules.rest.globals.di object, for example:

In this case, the service acceptor is connected, it will become available for requests at the url https://dnt.iondv.com/rest/acceptor. A functional description of interaction with requests should be contained in a script‘‘modules/rest/lib/impl/acceptor.js‘‘. In the options field, any variables and their values can be indicated, which will become available in the script through the fields of the object passed as an argument to the main function of the module.

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

For a detailed description of how to create a service, see https://github.com/iondv/rest/blob/master/README_RU.md #### Development of a service handler in the application

Utilities for printed forms

Back: Functional utilities of the application

Utilities for printed forms (injectors) are designed to process the data output to a template, including intermediate calculations and formatting. The printed form for which the injector will be used must be defined in deploy.json, for example:

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, you need to create a folder with the name of the namespace, and then put a file with the name of the class you need to unload from this namespace in it, for example:

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:

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

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

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:

Example of the monthTicketStats.js file:

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

Here you should pay attention to the params field - in it you can specify the parameters available in the export form in the application web service, as of 12.24.2019 the following types of parameters are possible: “string” - text enter string, “date” - an interactive calendar where you can select a date of interest “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 parameter of the this.inject function.

- utilities for scheduled tasks
- utilities for workflow
- utilities for action button
- utilities for web-service (REST module)
- utilities for printed forms

5.2 Using templates for data entry fields in a web form

Templates are used for setting custom parameters for building data entry fields in a web form. To connect a template to a web form field, specify it in the options of the corresponding json field of the form:

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

Templates in .ejs format are loaded using the path specified in modules.registry.globals.templates from deploy.json. For example, if the path applications/khv-ticket-discount/templates/registryis specified, then the applications/khv-ticket-discount/templates/registry/capitalize.ejs script will be loaded for the property field in the previous example.

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, read more: Options.

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">
```

(continues on next page)

(continued from previous page)

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

5.3 Key core 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]) ...

Эта страница на Русском

IONDV. Framework - is a node.js open source framework for developing accounting applications or microservices based on metadata and individual modules. Framework is a part of instrumental digital platform to create enterprise (ERP) apps. This platform consists of the following open-source components: the 'IONDV. Framework<https://github.com/iondv/framework>'_, the modules and ready-made applications expanding it functionality, visual development environment Studio to create metadata for the app.

глава 6

Description

IONDV. Framework — is a tool for creating high-level web applications based on metadata. You can change the system by adding the additional components to change functionality. There are ready-made modules, but nothing limits you to create new ones to personalize the application. Moreover, it's low-code framework.

The main purpose is the implementation of complex data registry systems. The functional base is the data registry - the Registry module. This is a key module designed specifically to work with data based on metadata structures - including the management of projects, programs, activities, etc.

IONDV. Framework is an open source software in JavaScript with open metadata structure in human-readable JSON files.

ГЛАВА 7

How to design an application?

What? Business application of any class.

How? Describe the data and apply ready-made modules that you can adjust for specific tasks.

core + metadata + modules = application

In the square frames - ioncore, meta class, meta view, meta navigation и registry module - are the base of the simplest application. Below are additional types of meta and modules. They represent additional functionality and could be applied in accordance with the application. Look for the application dependencies in the package.json file.

7.1 Typical applications

We give you a frame for creating applications in JavaScript, both enterprise level and highly functional - from the portal to analytics:

- Document Management;
- Accounting and Reporting;
- Enterprise Resource Management;
- Workflow Management and Project Activities;
- Data Capture;
- Business Analytics;
- System Integration.

7.2 Free Demos

For now, we have three demos to show you:

- 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>‘_
- DNT - is our application for development and testing, on the basis of which new meta components are implemented and tested. So almost all elements of the system are in the DNT app. GitHub Repo.
- 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 enterprise application based on IONDV. Framework. Project management system allows you to organize project activities: to monitor the results, to comply with and reduce the deadlines, to use effectively temporary, human and financial resources, making timely and informed management decisions. GitHub Repo
- Telecom (in Russian) - is a web application based on IONDV. Framework. It is used as a registry to account, store, and present the data on the availability of communication services (Internet, mobile communications, television, mail, etc.) in populated areas of the region. ‘GitHub Repo<https://github.com/iondv/telecom-ru>‘_
- CRM - coming soon on GitHub.

The login for access is - demo and the password is - ion-demo. No registration required.

ГЛАВА 8

Top 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;
- connection of modules providing additional functionality and implemented through access to the kernel interfaces (APIs);
- providing import, export of data in the system, metadata, 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 tasks;
- notification of users by events.

You can find out more about the functionality of the framework and its modules.

ГЛАВА 9

Quick start

You can get access to the already built applications deployed on Cloud servers or explore the different ways on the IONDV.Framework site, for example:

- gitclone with this repository
- installer for linux operating system
- docker-container with the already built application
- archive with the already built application

9.1 Software requirements

Install Node.js runtime and npm package manager to run the IONDV.Framework. Version 10.x.x.

Install and run the MongoDB DBMS to store the data. Version 3.6.

9.2 Installer

You can use IONDV. Framework 'apps installer'_, requiring installed node.js, mongodb and git. During the installation, all other dependencies will be checked and installed, and the application itself will be built and run.

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.

Also 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

9.3 Gitclone with repository

9.3.1 Global dependencies

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

- package node-gyp npm install -g node-gyp. For the Windows operating system, it is additionally necessary to 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 - supported version of Gulp.
- package manager of frontend libraries ‘Bower<https://bower.io>‘_ npm install -g bower.

9.3.2 Core, modules and application

Let’s consider the example of the develop-and-test application. At the place of the develop-and-test application, the path may indicate namespace. This means that you must put the name of the application in the path yourself, instead of namespace. Find the develop-and-test application in the repository. Look at the dependencies specified in the 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": "3.0.0"} parameter. Copy the URL of the core repository and execute the command git clone https://github.com/iondv/framework.git dnt, where dnt is a application name, for example full path is /workspace/dnt’. Go to the core folder and switch the tag of the version number git checkout tags/3.0.0.
- 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.git. Go to the

folder of the installed module and switch the tag of the version number git checkout tags/3.0.0. Repeat for each module.

- To install the application, go to the application folder executing the cd ..\applications command, if you're in the module folder. Clone the path to repository by git clone https://github.com/iondv/develop-and-test.git command. Go to the folder of installed application and switch the tag of the version number git checkout tags/2.0.0
- 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, in particularly viewlib - a additional application - library of views templates. Execute the git clone https://github.com/iondv/viewlib.git to clone to the applications folder. 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.

9.3.3 Building, configuring and deploying the application

Building 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 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 which is equal to the path of the application core. For Windows the command is set NODE_PATH=c:\workspace\dnt, for Linux - export NODE_PATH=/workspace/dnt, where /workspace/dnt is the directory of the application.

The npm install installs all key dependencies, including locally the gulp build-tool. Please make sure that the Gulp version - is 4.0.

Further, execute the gulp assemble command to build and deploy the application.

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

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

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

9.3.4 Running

Run the app, executing the npm start or node bin/www command.

Open this link http://localhost:8888 in a browser and log in. 8888 — is a port in the server.ports parameter.

9.4 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/dnt.
3. Open the http://localhost link in the browser in a minute (it takes time to initialize the data). For back office login: demo, password: ion-demo

ГЛАВА 10

Documentation

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

ГЛАВА 11

Reference

Some handy links to learn more information on developing applications using IONDV.Framework.

- ‘Documentations https://iondv.readthedocs.io/en/latest/index.html>‘_
- Homepage
- Feedback on Facebook

ГЛАВА 12

License Contact us English