Catalogue of Interoperability Solutions: CatIS

Administration Guide

2017-11-22

Acknowledgments

This guide is meant for users, who will define and manage the application forms, data structures, workflow process rules, etc.

Table of Contents

1. Users, accounts and user rights

After installing a new catalogue environment there are no users defined in the system. To add the first administrator user, navigate to the Log In page and click "Sign up now". On the self-registration page, create a new user. This very first user will automatically be set as application administrator and depending on the system configuration (see separate Installation and Configuration Guide for details), the self-registration could be disabled from there on. After the first admin logs in, he can add new users in Administration->Users and designate them as application administrators if needed, or if the self-registration is not disabled, new users can register accounts themselves.

1.1. Application administrators

A user can be designated as application administrator by checking the checkbox in the user's profile (only existing administrators can change this setting).
The application administrators have access to all parts of the application and edit rights to all objects, therefore this level of access should only be granted to as few people as possible.
Only application administrators can modify forms, designate other users as application admins, edit and delete user accounts.
In addition, only application administrators can create new institutions and delete them.

1.2. Institution users

When a new institution has been created in the catalogue, it is then possible to register new user accounts for that institution's users. When the application admin registers a new institution (either via registration request or directly), the first user for that institution should be also be registered (or he/she could already be self-registered) and assigned to this institution as institution's admin / responsible user. This is done on the "User Accounts" tab of the institution page. This user can later add other user accounts and possibly mark them as institution admins as well.
Users can be added to institution by creating a completely new user or by assigning an existing user as this institution’s user.
All users created from the institution "User Accounts" page are automatically related to that institution.

1.3. Owner roles

Each object in the catalogue can have one or more owners. Owners have edit rights for the object (in addition, application administrators have edit rights to all objects). Institution owners can be specified on the institution "User Accounts" page, clicking on the "Edit owner access" button. Select the institution users that should be able to edit institution data and click "Save changes".

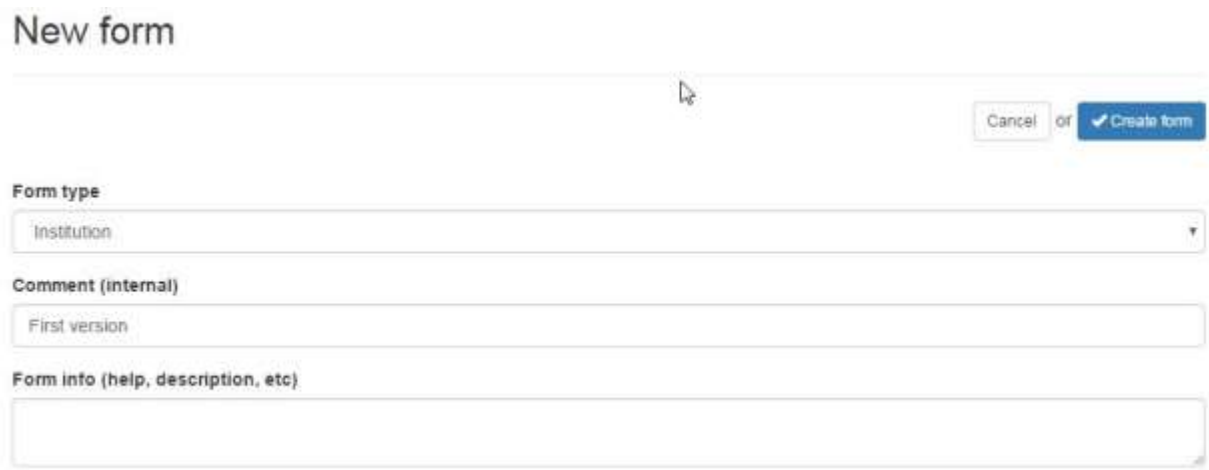
All other objects (information systems, services and assets) should have owners assigned too. Owners can be selected from the institution users on the Owners tab on the object's page. Only users assigned as owners can edit object's data (as well as application administrators).

2. Defining forms

One of the first things to do in a new application installation is to define the data structures of the application objects (institutions, systems, assets, etc).

2.1. Form types

There are five main object types in the catalogue: Institutions, Systems, Services, Assets and Registration Requests. For each object type, the catalogue holds certain data about that object; the exact data structure is configurable by the catalogue administrators.
One of the first things after new catalogue installation is to define the data structure and input forms for these five object types. To define the forms, log in as an administrator and navigate to Administration->Forms. If there are no forms listed yet or if not all types are visible, create a new form definition by clicking "Add new form". Select a form type and enter some descriptive information about the form. This information is displayed for the users whet they edit the respective object data, so it should describe the main purpose of the object, could contain helpful hints, etc.



You can create many forms of the same type, for example when preparing a new form while the old version is still in use or to try out different combinations, etc. You decide which of the forms from the same type is actually used by marking that form as active (NB! the "Active" checkbox is not visible on new forms, you must save the form first and then edit it again to make it active).When creating multiple forms of the same type, use the Comment field to describe the purpose or version of each form so you can tell them apart later.
Make sure that there is at least one form defined for each object type, 4 forms in total, add any that are missing. One (and only one) of the forms from each type should be set as active.

2.2. Default fields

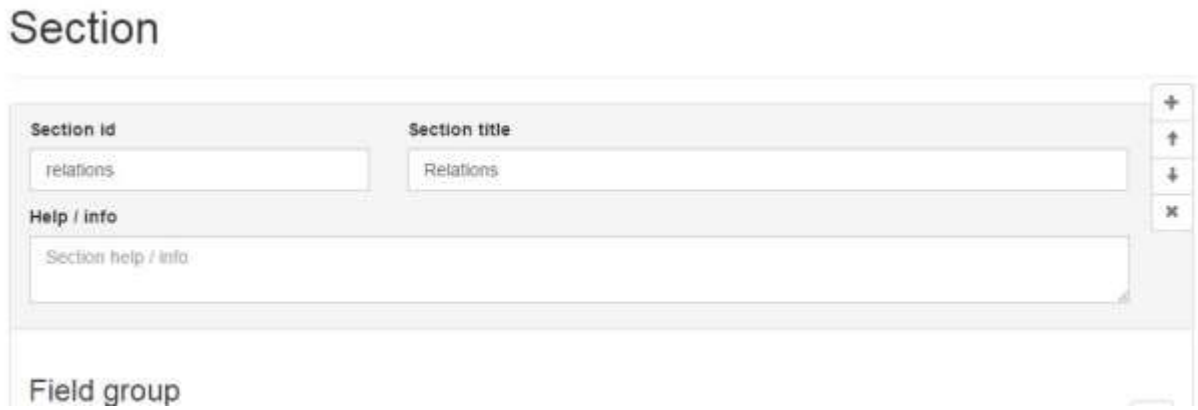
Most of the data is freely configurable for each object type, however certain fields are always present on the final forms and should not be defined separately. These are "name" and "identifier". Name is the name of the object, e.g. institution name; identifier is for example registration code or a short name, etc.

2.3. Form structure

All forms are defined in the same way and consist of sections, field groups and fields. At the top level the form contains one or more sections, each sections contains one or more field groups, each field group contains one or more fields.

2.4. Sections

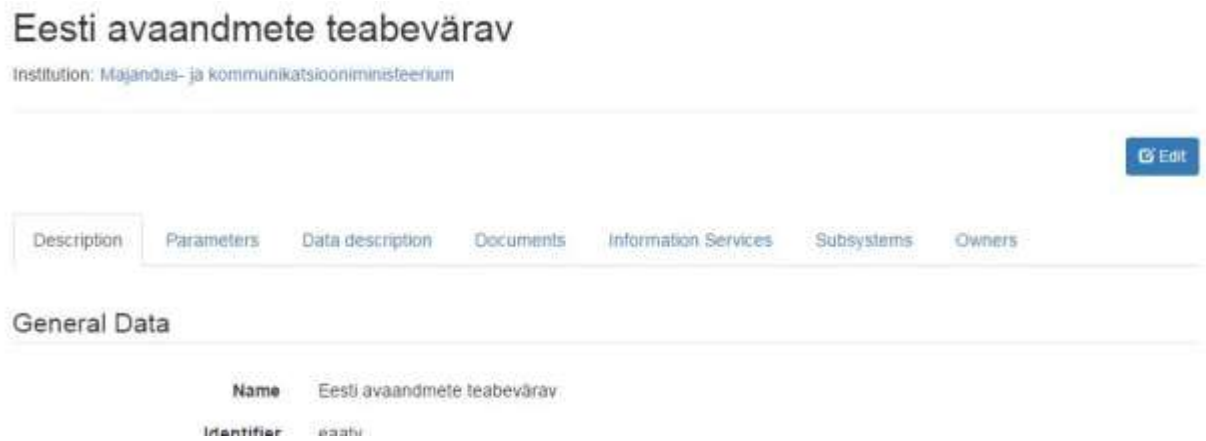
Sections define the main sections of the resulting form. When creating a new form, by default one empty section definition is displayed. To add new sections, click on the "Add new section" button at the end of the page (hover mouse over the "+" and other buttons to view the button description) or if you want to insert a new section before the current one, the "Insert new section" at the beginning of each section definition. You can also rearrange sections by clicking "Move section up" or "Move section down". The buttons are on the right side of the form.



`Section id`: internal identifier of the data section. It is best to use short but descriptive names, e.g. "contacts", "documents", etc. NB! Changing the identifiers later when there is already data in the catalogue will result in the old data no longer being displayed!

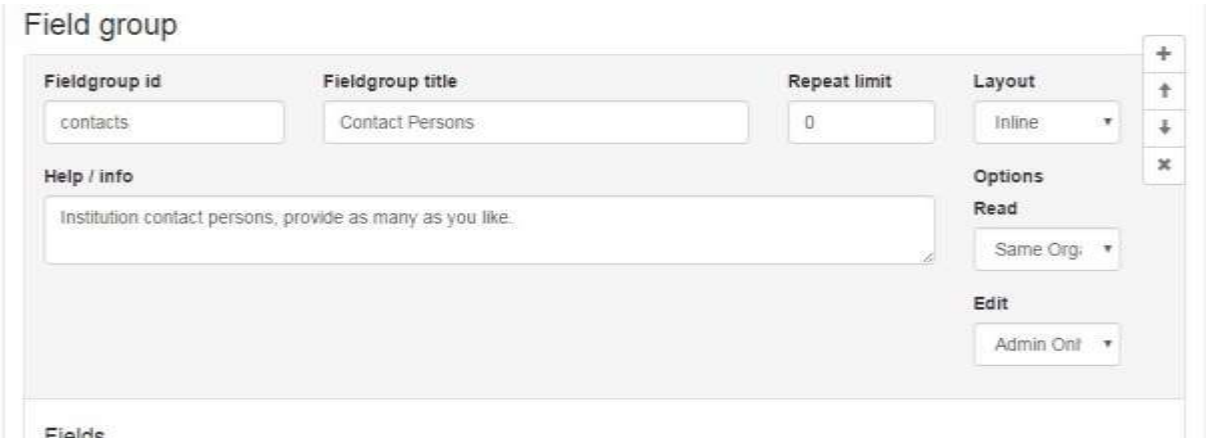
`Section title`: the title of the section as displayed to the users.

`Help / info`: any helpful information you want to be displayed to the users when they are filling out the form, e.g. purpose of the section or any special rules for inputting data, etc. Later when the users of the catalogue view information about the object (e.g. Institution), the sections are displayed as tabs in the tabbed layout, for example the following picture shows sections "Description", "Parameters", "Data description", etc.



2.5. Field groups

Each section consists of one or more field groups. Field groups organize and group related data and can have different visual representations to present the data fields in most meaningful way, for example field group "Contacts" could group fields "First name", "Last name", "E-mail" and "Phone" and present them later in a table layout.



`Fieldgroup id`: the internal identifier of the field group. It is best to use short descriptive names, for example "contacts". Like section id, changing this later can result in data loss.

`Fieldgroup title`: Visible name for the field group, for example "Contact Persons".
Help / info: any helpful information you want to be displayed to the users when they are filling out the form

`Repeat limit`: field group content can be repeated when filling out the forms, for example, users can add more than one contact person for an institution. The "Repeat limit" setting controls this as follows: if the setting is 0, there is no limit; fields are repeatable as many times as needed. If the setting is 1, the fields are not repeatable (can only appear once). Any other number represents the number of times the group of fields can be repeated, for example you can specify that only up to five contact persons can be entered by setting this to 5.

`Layout:` How the group of fields should be displayed to the users. The options are "Vertical", "Horizontal" and "Inline". The vertical layout has field labels on top of the fields and is best for long fields. Horizontal layout has field labels in front of the fields and is better for compact forms.

Both horizontal and vertical layouts repeat the group of fields as a whole with all labels and fields. In contrast, "Inline" layout displays the data as a table, with labels as column headers and each group of fields as a row in the table.

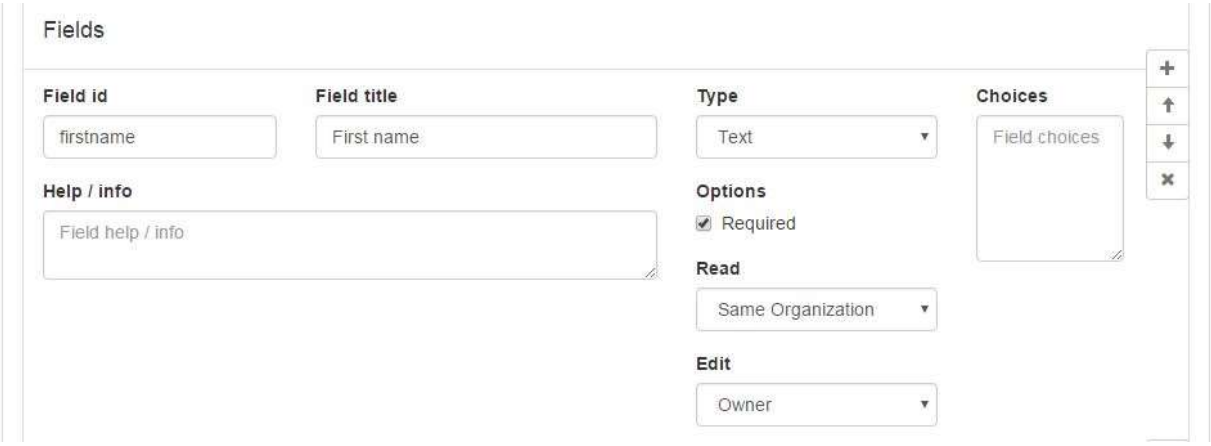
`Options:`

- `Read:` controls who can see this field group. The options are Everyone (no restrictions), Authenticated (not publicly visible, only visible to logged in users), Same Organization (the field group is only visible to users from the same organization that the object belongs to), Admin Only (only CatIS administrators can see this field group). Define the visibility according to the sensitivity of the data, for example contact information and email addresses should not be publicly visible on the Internet, etc. CatIS administrators can always see all field groups irrespective of this setting.

- `Edit:` controls who can edit data in this fieldgroup. Options are Everyone (everyone, even anonymous users, can edit data), Owner (only users marked as this object's owner can edit data) and Admin Only (only CatIS administrators can edit). The Everyone option is most useful on new registration request forms, where the user is not yet authenticated. For other forms the default should be Owner, but some data could be restricted to CatIS admins only. CatIS administrators can always edit any field group irrespective of this setting.

2.6. Fields

Field groups contain fields. Each field is defined by several attributes, fields can be added, inserted, moved around just like sections and field groups.



`Field Id:` the internal identifier if the field. It is best to use short descriptive names, for example "firstname". Like other id's, changing this later can result in missing data. The field ids do not need to be unique within the whole form, but they must be unique within the same field group.

`Field title:` Visible name for the field, for example "First Name".

`Help / info:` any helpful information you want to be displayed to the users when they are filling out the form

`Type:` field type. The available types for the fields are:
text - simple one-line text input
number - shorter input field, validates numerical format, has spinner controls
date - shorter input field, validates date format. Has a popup calendar to choose a date from.
email - validates email format, in view mode presents as emails links
URL - in view mode presents input as URL link
textarea - multiline text box for longer text
boolean - one checkbox with true/false value
checkbox - one or more checkboxes with textual values
select one - input list for selecting a single value
select multiple - input list for selecting one or more values
file - file upload
asset - input list to select an existing asset from the catalogue
service - input list to select an existing service from the catalogue
system - input list to select an existing information system from the catalogue
institution - input list to select an existing institution from the catalogue

`Choices:` for field types that have choices (checkbox, selects) enter possible choices here, each entry on a new line

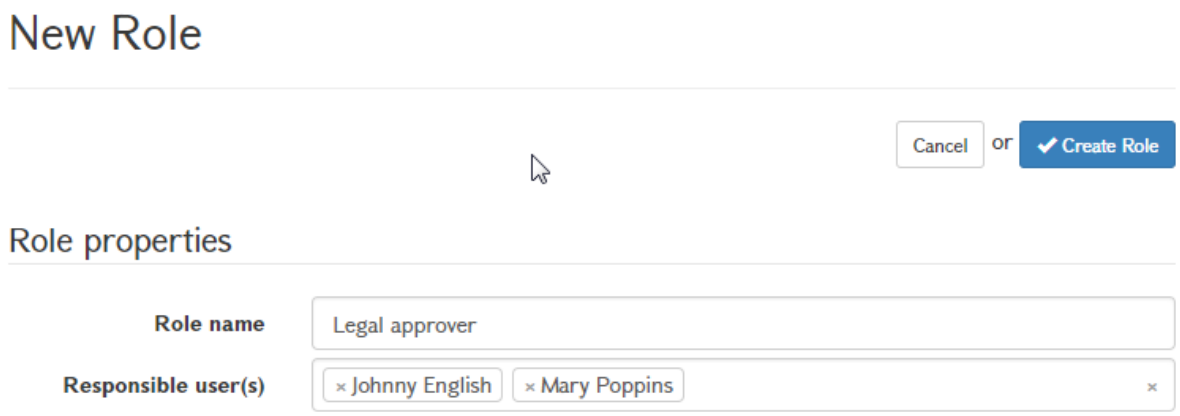
`Options:`
- `Required:` if the field is required to be filled out (cannot be left empty)

- `Read:` define who can see this field. See the description of the fieldgroup above for possible values. Use if you only want to protect some fields from the field group. Note that if the read access on the containing field group is more restrictive, it already hides all the fields that belong to the field group and setting this per single field has then no effect.

- Edit: define who can modify this field. See the description of the fieldgroup above for possible values. Use if you only want to protect some fields from the field group. Note that if the edit access on the containing field group is more restrictive, it already disables editing of all the fields that belong to the field group and setting this per single field has then no effect.

3. User roles

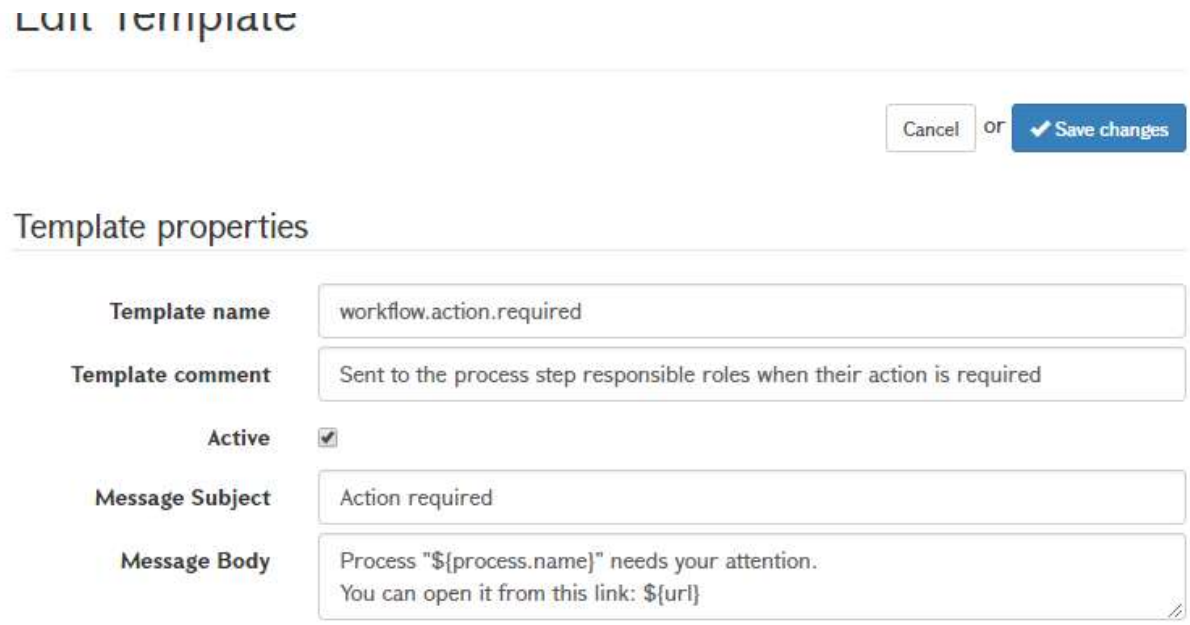
Roles are used in workflows to connect a workflow role to a specific person or persons. For example, you could define a role "Legal reviewer", assign it to a certain person and use that role in multiple workflows. Then for example when the person leaves, you can just change the associated person in that role and all workflows continue working, without the need to update all the workflow definitions where that person might have been included.
To define roles, navigate to Administration –> User Roles and click on Add new role. Fill in the role name and select one or more responsible users.



A role can have more than one responsible users and a person can belong to any number of roles.
When the role is later used in the workflow, any of the listed users can perform that role, so that when one person is unavailable, another can step in.
Role name can be translated to different languages, see Chapter 10 Languages and Translations.

4. Email templates

During the workflow process, the application notifies the responsible users of the required actions by sending e-mail messages. The templates of those messages should be defined before the messages can be sent (when there is no template for some messages, no error is generated and the message is just ignored).
To define the message templates, navigate to Administration -> Email templates



Template name is used as an identification key within the application and is not visible to users. It should be unique and concise.

Template comment is for the admins to keep track of what template is meant for what use, etc.

Active should be checked, if this template is to be used in the workflows. You deactivate a template (and therefore that message) without actually deleting it or you can keep several templates with the same key in the application, for example, when working on different template versions, but at most one of the templates with the same key (template name) should be marked active at the same time.

Message Subject is the subject line of the message and Message Body is the message itself.

Email template Subject and Body texts can be translated to different languages, see Chapter 10 Languages and Translations. The workflow notifications are automatically sent according to the user's preferred language (set in the user profile), if the language version exists, otherwise the default language template is used.

4.1. Variable substitution

You can use certain process variables in the email message Subject and Body texts. The format of the variables is ${<variable name>}, where <variable name> can be a process property, a related object property (a related object is the "parent" object of the workflow process instance, e.g. and institution or an asset) or a literal name "url".
The properties of the workflow process are prefixed with the word "process" and the properties of the related object with the word "object".

The properties of the process are:

process.name – name of the process, from workflow definition

process.info – process description from the workflow definition

process.createComment – comment or additional information provided by the person who started the process

The properties of the object are:

object.name – the name of the object, e.g. institution name

object.<section>.<fieldgroup>.<field> - fields as defined on the corresponding object form. NB! Case must be taken when using the fields in such way. If the field does not exist (e.g. the form is changed sometime later), the template must be changed also.

The URL to the object can be included in the message using a literal name "url". The configuration property "play.http.urlprefix" (see Chapter 2 Configuration) must point to the proper scheme, server and port for the links to work.

5. Defining workflows

To support various business processes the application incorporates role-based workflow functionality. Workflow processes can be started automatically (e.g. somebody creates a new institution registration request, a review process could be started behind the scenes to review and accept/decline the application) or manually by selecting a process from a list and starting it (e.g. submit an application to connect an information system to the information exchange infrastructure).

Filters are used in the workflow definition to restrict specific workflows to certain objects and their statuses, e.g. “Connect an IS to Xroad” process is only available on an IS form when the status of the IS is “Production”, etc.

Roles are used to support workflow processes, where steps in the process are assigned to roles instead of individual users. Roles are then mapped to users in a many-to-many relationship, i.e. each role could be fulfilled by more than one user and each user can have more than one role.

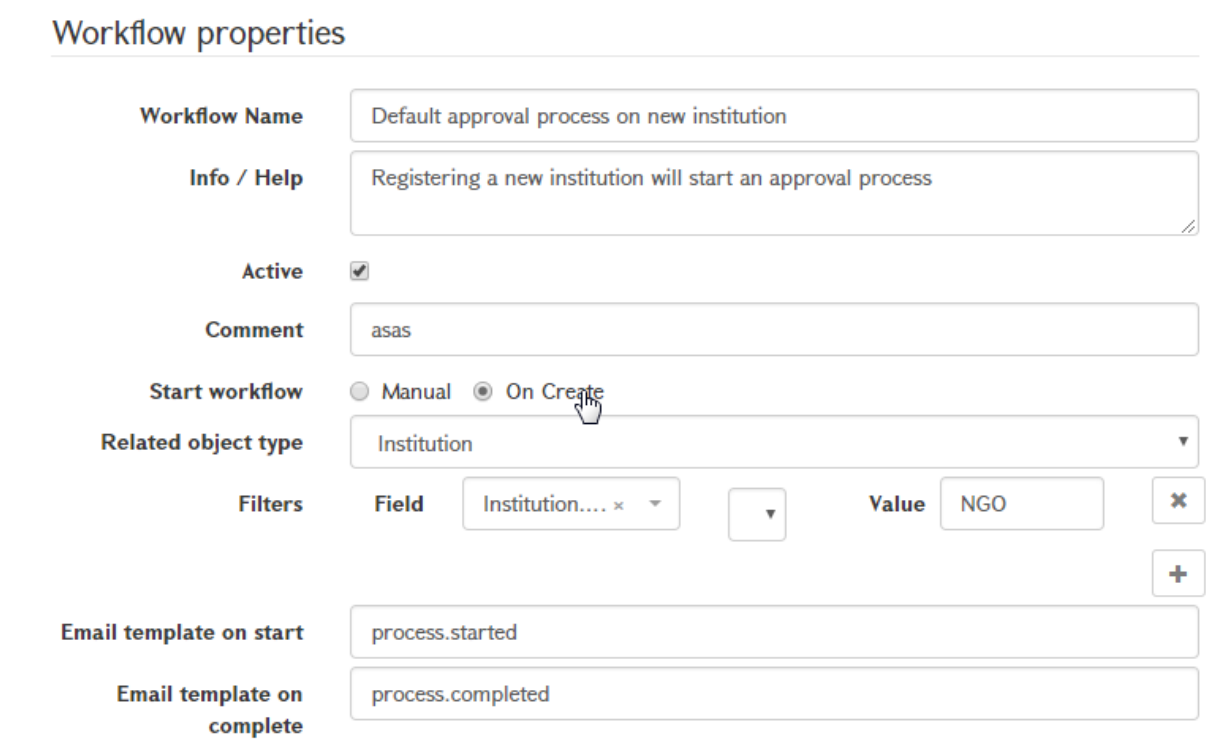
Workflow processes are composed of steps or states, where each step represents some action that has to be taken or a decision to be made, e.g. review or approval. Each step is assigned to some role who is responsible for that action. The system will automatically resolve roles to assigned users, send them email notifications and give them appropriate access to perform that step.

Steps can include time limits; the system will send email reminders to the responsible users when the action is not performed in the specified time.

Steps can be optional, depending on some attribute of the object the process is related to, e.g. if the IS contains sensitive data, additional approvals may be needed in such cases, but skipped if the IS does not contain any sensitive data.

Currently executing and past process results are visible on the respective form so that the decisions made and who made them and when can be verified later.

To define workflow processes, go to Administration -> Workflow Definitions, click on "Add new Workflow Definition". A new workflow definition form opens:



5.1. General properties of the workflow process

`Workflow name` – the name of the workflow process. This name is visible to the users; it should be short and descriptive.

`Info / Help` – description of the process or informational text for the users. It should clarify the purpose of the process.

`Active` – if this process can be selected/started

`Comment` – internal comment about the process. This comment is not visible to the users and is meant for the application admins to keep track of the processes.

`Start Workflow` – the workflow process can be started either manually selecting it from a list or automatically, when a new object is created.

`Related object type` – the type of the object that this workflow applies to. The workflow is only visible (or can be automatically started) in the context of this object type, e.g. Institution, Asset, etc.

`Filters` – to restrict visibility of the workflows, filters can be added to the workflow definition. The workflow definition is only visible in the context of those concrete objects (that

must be of correct type first), that match the filter(s). If more than one filter is added, the object must match all conditions.

- `field` – the field from the form definition, e.g. institution type

- `condition` – could be either "equals", "does not equal" or "matches regular expression"

- `value` – the value to test against, e.g. "NGO"

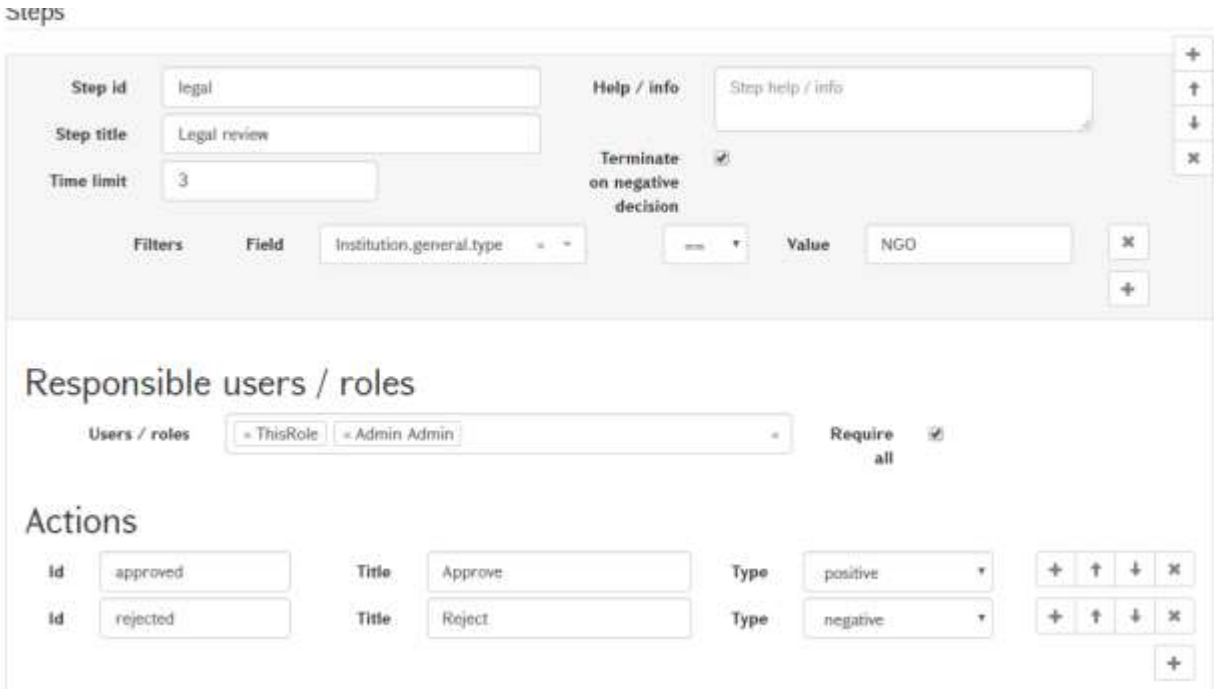
`Email template on start` – email template to use for messages that are sent when a new process is started (currently not used)

`Email template on complete` – email template to use for messages that are sent when the process completes. The messages are sent to the requester, i.e. the person who started the process. If this field is empty, the default template key `"default.process.completed"` is used. If the template with this key (or the default key) is not found, the message is not sent (no error is generated).

`Email template on step` – email template to use for messages sent to step responsible users. If this field is empty, the default template key `"default.action.required"` is used. If the template with this key (or the default key) is not found, the message is not sent (no error is generated).

5.2. Steps of the process

The workflow process consists of one or more steps. A step is one concrete task or decision performed by one or more roles or persons. Steps can be added, inserted, reordered and deleted using the buttons to the right of the step properties block.



Each step has the following properties (some of them, like filters, are optional):

`Step id` – a unique (within the workflow definition) identifier

`Step title` – a user visible title or name of the step

`Help / Info` – a clarification of the step's purpose, to help responsible persons understand what they are required to do.

`Time limit` – time limit in days. If this many days has passed, but the step is not completed, the system will send out email notifications to the responsible persons. NB! This notification looks for an email template with the key `"default.action.reminder"`, this is not currently configurable.

`Terminate on negative decision` – when checked, a negative decision in this step will complete the whole process. If unchecked, the decision is recorded, but the process will continue.

`Filters` – not required for simpler workflows, but in a more complex process, some steps can be optional, depending on some field value of the related object. If the related object does not pass the filter(s), the step is skipped. A step can have one or more filters, if multiple filters are specified, all of them must match for the step to be included in the process.

- `field` – a field from the object form definition

- `condition` – a test condition, either "equals", "does not equal" or a regular expression for more complex conditions

- `value` – a value that the field should have (or not have) to pass the filter

`Responsible users/roles` – one or more roles or specific users who are responsible for performing the step actions (making a decision or doing something). A selection from all defined roles and users. If a role is selected, the persons responsible for that role are responsible for that step. If no suitable role is defined, specific persons could be selected also.

When the step is started, an email message is sent to the responsible persons automatically. This notification looks for the email template with the key specified in the workflow definition properties block "`Email template on next step`". If this field is empty, a default key "`default.action.required`" is used. This key is not currently configurable individually for each action.

`Require all` – if this is checked, all roles/persons selected in this field must perform their action before the step is completed. If unchecked, only one of them is sufficient for deciding the outcome of the step. NB. Each role is always performed by only one of the role responsible, regardless of this setting.

`Actions` – this defines the possible actions that the responsible persons can take in this step. A step can have one or more actions (two is the most common). These actions are then presented to the responsible persons when the process is executing.

- `id` – a unique (within the step) system id of the action

- `title` – a user visible title of the action. This is displayed in the possible actions list to the users

- `type` – type of the action, either positive, negative or neutral. For example in an approval process, a positive action marks approval, negative action marks rejection. In some processes, e.g. a FYI type of process, an action can be neutral, marking that the user has seen the document or has done what was requested, but he/she does not decide the the result of the process.

Actions can be added, inserted, deleted, reordered using the buttons to the right of the action.

6. Registration forms and new registration requests

External people can request that an institution be registered in CatIS by using the "Register new Institution" button in the Institutions view. When a new registration request is received in the system, an email notification is sent to the address specified with the `registration.sendto` parameter in `application.conf` configuration file (usually CatIS administrators). Then, CatIS administrator can find the new registration requests in the Administration-Registration Requests view, from where it is possible to review, modify accept or deny the request. When accepting a request, a "Create new institution" form is opened automatically with predefined data from the request. An email notification about the decision is also sent out to the address given with the registration request.

Before submitting registration requests will be possible, there must be a form defined for these requests. It is best to define this registration form after the institution form has stabilized and use the same section/fieldgroup/field names as on the Institution form. That way, when CatIS admin approves the registration request and starts to create a new Institution in the system, all provided data can be carried over from the registration request.

There are three predefined fields on the registration forms that cannot be removed and should not be duplicated: Institution name, registration code and contact email. Institution name and registration code are carried over to the Institution form when the request is accepted and the contact email address is used to send notifications about the request status or decisions made.

Also please note that the fieldgroups and fields that should be filled before submitting the request, should specify access right as "editable by everyone" in the form definition.

7. Defining views

Views, or lists of objects (e.g. index view), can optionally be modified to change default columns and show most relevant data for each type of object. To define or modify views, log in as an administrator and navigate to Administration->Views. Create a new view definition by clicking on "Add new view". Select a view type and enter the title for the view. You can add some informative text for the users or some internal comment (not visible to users).

A view consists of columns, for each column you need to enter a title and select source field. The selection available for source field depends on the view type (what objects are displayed in this particular view) and are fetched from the respective form definition. Naming of the source fields is based on the form definition and is composed of section name, fieldgroup name and field name. Therefore, views should be defined after forms are ready and each time a form is changed later, the views should be verified and if necessary changed to reflect new fields.

Add/delete/rearrange more columns using the buttons on the right side, but be conservative, do not make the view too crowded.

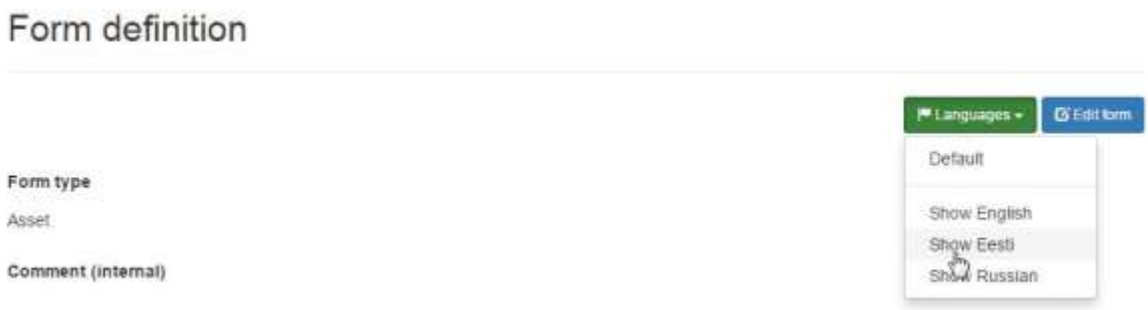
If you want the new definition to take effect, you also need to tick the "Active" checkbox. Then Save the view definition and verify it shows what you wanted.

Similar to forms, you can have multiple view definitions for the same view type, but only one of them should be made active. If there is no (active) view definition for particular type, the system defaults (name, id) are used.

8. Languages and translations: translating forms, processes, etc.

If the application should be used in multiple languages, the forms and many other objects defined in the system should be translated as well. The following description is based on a form definition, but the procedure is the same for view, workflows, email templates, etc.

Translations can be added after the form sections/fieldgroups/fields are (mostly) defined and the form definition is saved. To see the different language versions of the form, select another language from the Languages drop-down:

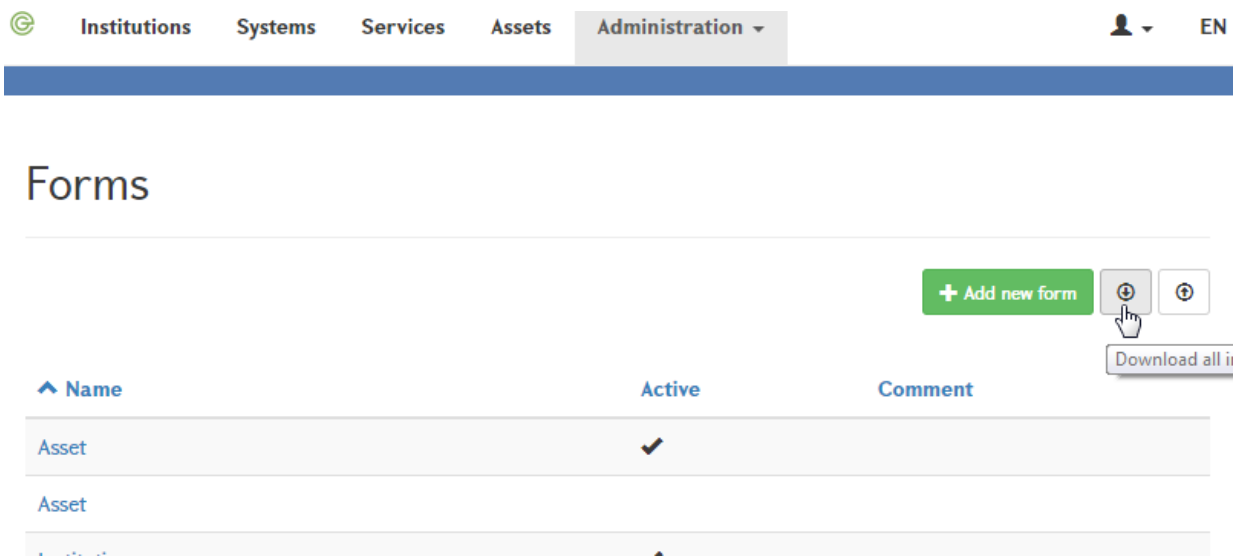


If the language texts need to be changed, click on the Edit form button and replace the labels, help texts/etc with translated text, then save the form definition.

When editing language version of the form, only translatable labels, help, etc. can be modified, the sections, fieldgroups or fields cannot be added or modified. To modify the form structure, use the Languages - Default option to switch to actual form definition and then click Edit form.

9. Bulk downloading and uploading of application configuration objects

To support easier first configuration of the application or migrating from one instance to another, the configurable elements of the application (forms, views, roles, workflow definitions) can be downloaded in bulk from one instance of the application and then uploaded to another. For this, the corresponding views in the administrative section of the application have buttons on the right:



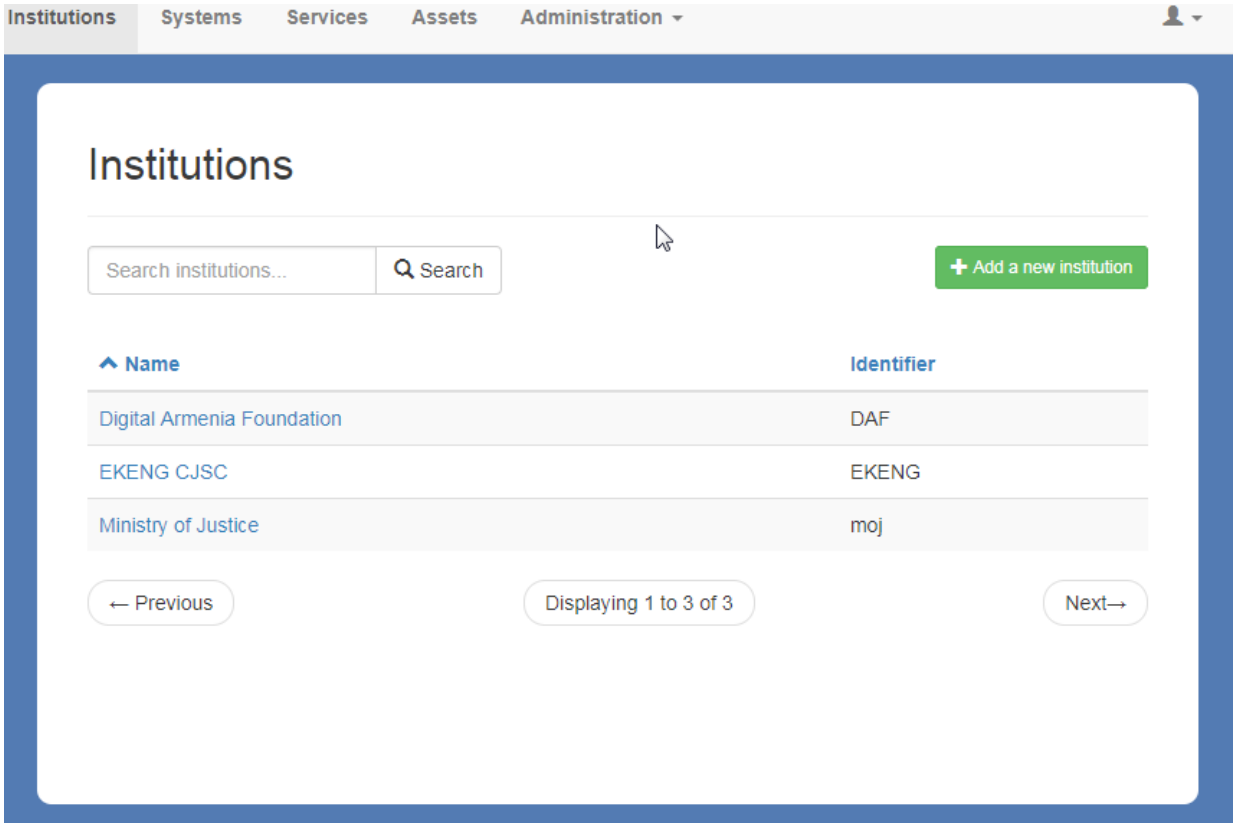
Clicking on the "Download all in this view" will download all currently configured objects (e.g. forms, as shown in the above picture) in one JSON file. This file can then be uploaded to another instance of the application.

10. JSON and XML API

The following chapters are meant for developers and other people. Users acting as administrative users of the application may find this information useful; however, it is not required to support the day-to-day operation of the application.

For integrating with other external systems, the application provides a REST-style API that can be used to query the data in a few different formats. All the URI-s are the same, normally, for browser requests, the application responds with HTML content. If the API client specifies another format in the HTTP Accept: header, the application can respond with JSON or XML data respectively.

For example, making a browser request to `http://server.example.com/catis/institutions` could result in a page like this:



If the request header includes `Accept: application/json`, the same set of data would look like this:

```
{
    "page": 0,
    "offset": 0,
```

```
    "total": 3,
    "orderBy": 1,
    "filter": "",
    "items": [
        {
            "id": "IN00001",
            "name": "Digital Armenia Foundation",
            "href": "/am/catis/institutions/IN00001"
        },
        {
            "id": "IN00002",
            "name": "EKENG CJSC",
            "href": "/am/catis/institutions/IN00002"
        },
        {
            "id": "IN00003",
            "name": "Ministry of Justice ",
            "href": "/am/catis/institutions/IN00003"
        }
    ]
}
```

If the request header has Accept: application/xml, the result would be:

```
<?xml version="1.0" encoding="UTF-8"?>
<items page="0" offset="0" total="3" orderBy="1" filter="">
    <institution id="IN00001" name="Digital Armenia Foundation"
  href="/am/catis/institutions/IN00001"/>
    <institution id="IN00002" name="EKENG CJSC"
  href="/am/catis/institutions/IN00002"/>
    <institution id="IN00003" name="Ministry of Justice "
  href="/am/catis/institutions/IN00003"/>
</items>
```

To fetch the data about some concete entity, the API client simply needs to follow the href attribute and make another request(s). The XML schema of the metadata is provided in Appendix 1. It must be noted that the scema does not specify the actual data itself (since the concrete dataset is provided via application configuration, i.e. what data is included with each object, is specified in the forms definition process), but rather the format of presenting the metadata. An example of an institution metadata in XML format is provided in Appendix 2 and in JSON format in Appendix 3.

Appendix 1. XML Schema for metadata

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
           xmlns="http://ega.ee/catis"
targetNamespace="http://ega.ee/catis">

    <xs:element name="institution" type="institutionType"/>
    <xs:element name="system" type="systemType"/>
    <xs:element name="service" type="serviceType"/>
    <xs:element name="asset" type="assetType"/>

    <xs:complexType name="institutionType">
        <xs:complexContent>
            <xs:extension base="compositeType"/>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="systemType">
        <xs:complexContent>
            <xs:extension base="compositeType">
                <xs:sequence>
                    <xs:element name="_stdsys"
type="xs:boolean"/>
                    <xs:element name="_references">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="institution"
type="refType"/>
                                <xs:element
name="standardSolution" type="refType" minOccurs="0"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="serviceType">
        <xs:complexContent>
            <xs:extension base="compositeType">
                <xs:sequence>
                    <xs:element name="_references">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="institution"
type="refType"/>
```

```
                                <xs:element
name="parentInfoSystem" type="refType"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="assetType">
        <xs:complexContent>
            <xs:extension base="compositeType">
                <xs:sequence>
                    <xs:element name="_references">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="institution"
type="refType"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:complexType name="compositeType">
        <xs:sequence>
            <xs:element name="_ident" type="xs:string"/>
            <xs:element name="_owners">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="owner" type="refType"
minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="_history" type="historyType"/>
            <xs:element name="sections">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="section"
type="sectionType" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
```

```
        <xs:attribute name="name" type="xs:string"
use="required"/>
        <xs:attribute name="_id" type="xs:string"/>
    </xs:complexType>

    <xs:complexType name="refType">
        <xs:sequence>
            <xs:element name="id" type="xs:string"/>
            <xs:element name="name" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="historyType">
        <xs:sequence>
            <xs:element name="createdBy" type="xs:string"/>
            <xs:element name="createdAt" type="xs:dateTime"/>
            <xs:element name="modifiedBy" type="xs:string"/>
            <xs:element name="modifiedAt" type="xs:dateTime"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="sectionType">
        <xs:sequence>
            <xs:element name="fieldGroups">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="fieldGroup"
type="fieldGroupType" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string"
use="required"/>
    </xs:complexType>

    <xs:complexType name="fieldGroupType">
        <xs:sequence>
            <xs:element name="fieldGroupContents">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="fieldGroupContent"
maxOccurs="unbounded">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element name="field"
type="fieldType" maxOccurs="unbounded"/>
                                </xs:sequence>
                            </xs:complexType>
```

```
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string"
use="required"/>
    </xs:complexType>

    <xs:complexType name="fieldType">
        <xs:choice>
            <xs:element name="text" type="xs:string"/>
            <xs:element name="textlist">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="text"
type="xs:string" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="number" type="xs:integer"/>
            <xs:element name="numberlist">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="number"
type="xs:integer" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="date" type="xs:dateTime"/>
            <xs:element name="datelist">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="dateTime"
type="xs:dateTime" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="boolean" type="xs:boolean"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:string"
use="required"/>
    </xs:complexType>

</xs:schema>
```

Appendix 2. Example of metadata in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<institution xmlns="http://ega.ee/catis" name="Digital Armenia
  Foundation" _id="IN00001">
    <_ident>DAF</_ident>
    <_owners>
    </_owners>
    <_history>
        <createdBy>Admin Admin</createdBy>
        <createdAt>2017-11-07T12:53:51.820+02:00</createdAt>
        <modifiedBy>Admin Admin</modifiedBy>
        <modifiedAt>2017-11-07T12:58:58.200+02:00</modifiedAt>
    </_history>
    <sections>
        <section name="Institution">
            <fieldGroups>
                <fieldGroup name="general">
                    <fieldGroupContents>
                        <fieldGroupContent>
                            <field name="englsh-name">
                                <text></text>
                            </field>
                            <field name="regcode">
                                <text></text>
                            </field>
                            <field name="URL">

  <text>http://www.gov.am/en/</text>
                            </field>
                            <field name="description">
                                <text></text>
                            </field>
                            <field name="type">
                                <textlist>
                                    <text>Central
  Government</text>
                                </textlist>
                            </field>
                            <field name="role">
                                <textlist>
                                    <text>Solution
  Developer</text>
                                </textlist>
                            </field>
                            <field name="status">
```

```
                                <textlist>
                                    <text>Application</text>
                                </textlist>
                            </field>
                            <field name="X-Road">
                                <textlist>

                                </textlist>
                            </field>
                        </fieldGroupContent>
                    </fieldGroupContents>
                </fieldGroup>
                <fieldGroup name="contacts">
                    <fieldGroupContents>
                        <fieldGroupContent>
                            <field name="name">
                                <text>Eduard Nersisyan</text>
                            </field>
                            <field name="email">

<text>eduard.nersisyan@gov.am</text>
                            </field>
                            <field name="phone">
                                <text>+374 10 515710; +374 55
413088</text>
                            </field>
                            <field name="function">
                                <text>x</text>
                            </field>
                        </fieldGroupContent>
                    </fieldGroupContents>
                </fieldGroup>
                <fieldGroup name="events">
                    <fieldGroupContents>
                        <fieldGroupContent>
                            <field name="type">
                                <text></text>
                            </field>
                            <field name="date">
                                <date></date>
                            </field>
                            <field name="decision">
                                <text></text>
                            </field>
                        </fieldGroupContent>
                    </fieldGroupContents>
                </fieldGroup>
            </fieldGroups>
        </section>
```

```
    </sections>
</institution>
```

Appendix 3. Example of metadata in JSON


```
{
    "_id": "IN00001",
    "_owners": [],
    "_history": {
        "createdBy": "Admin Admin",
        "createdAt": "2017-11-07T12:53:51.820+02:00",
        "modifiedBy": "Admin Admin",
        "modifiedAt": "2017-11-07T12:58:58.200+02:00"
    },
    "_name": "Digital Armenia Foundation",
    "_ident": "DAF",
    "Institution": [
        {
            "general": [
                {
                    "englsh-name": "",
                    "regcode": "",
                    "URL": "http://www.gov.am/en/",
                    "description": "",
                    "type": ["Central Government"],
                    "role": ["Solution Developer"],
                    "status": ["Application"],
                    "X-Road": []
                }
            ]
        },
        {
            "contacts": [
                {
                    "name": "Eduard Nersisyan",
                    "email": "eduard.nersisyan@gov.am",
                    "phone": "+374 10 515710; +374 55 413088",
                    "function": "x"
                }
            ]
        },
        {
            "events": [
                {
                    "type": "",
                    "date": "",
                    "decision": ""
                }
            ]
```

```
            }
        ]
    }
```