Dynamic forms (eav)

configuration, eav, form, attachment

Dynamic form instances (values) are saved in the individual tables according to the entity which they are linked to ⇒ which is their owner (e.g. the entity `IdmIdentityFormValue`, `IdmRoleFormValue`). Form values are not saved, if `null` value (by persistent type) is given ⇒ filled values are saved only.

`FormService` service for working with the extended attributes on the back-end. `FormInstance` utility is useful on BE - contains value transformation to maps by attributes etc.. Use this service in your custom module, benefits:

- single autowired service for work with definitions, attributes and values
- cache `core:form-definition-cache` for loading form definitions (with attributes) is effective here.

On the front-end, the editing of dynamic forms is done by the component `EavForm`.

Saving form values for the form definition work as **PATCH**. When attribute value has to be deleted, then form value with **null** has to be given (use it even for multi valued attributes).

If single attribute is saved (`FormService#saveAttribute`), then event EAV_SAVE is not published. Save all attributes (`FormService#saveValues`), if publishing event EAV_SAVE for owner is needed.

Dynamic form attribute supports data types (`persistentType`):

- `CHAR` - one character
- `TEXT` - strings (long text). Searching by TEXT is not supported, column is not indexed - **`SHORTTEXT` usage is preferred (+ indexed)**.
- `SHORTTEXT` - strings (2000 chars). Indexed.
- `INT` - integer
- `LONG` - long
- `DOUBLE` - saved as bigdecimal
- `BOOLEAN` - true / false / null
- `DATE` - date (without time)
- `DATETIME` - date with time
- `BYTEARRAY` - byte[]
- `UUID` - uuid identifier. Indexed.
- `ATTACHMENT` - attachment (~binary file). Read more about attachments.
- `CODELIST` - referenced code list - persists items "code" into short text. Uses face type as code list code.
- `ENUMERATION` referenced frontend enumeration - persists items "code" into short text. Uses

face type as enumeration name.

Changing `persistentType` and `confidential` is possible only for attributes without persisted values ⇒ when attribute is not used for some values. Data migration, when attribute's `persistentType` or `confidential` is changed is not supported now.

with properties:

- `readonly`
- `multi values` - Is represented on the front-end by a textarea, where a line is a value (a new line separates the values). This property is supported for persistent types `CHAR`, `TEXT`, `INT`, `LONG`, `DOUBLE` and `UUID`.
- `confidential` - .The values are stored in an confidential storage). Stored values of these attributes - substitute characters only - are loaded on the front-end. The value can only be changed and determined whether it is filled in. This property is supported for persistent types `CHAR`, `TEXT`, `INT`, `LONG`, `DOUBLE`, `UUID`, `BYTEARRAY`.
- `required` - value validation, read more.
- `unique` - value validation, read more.
- `min` - value validation, read more.
- `max` - value validation, read more.
- `regex` - value validation, read more.
- `validationMessage` - custom message, when some validation fails, read more.

Dynamic form attributes can be rendered differently on frontend. Face type (faceType) property is used for choosing frontend renderer. The default renderer is chosen by persistent type (e.g. UUID → UUID).

Renderer is a frontend component, superclass component `AbstractFormAttributeRenderer` is used for all renderer implementations. Renderer is responsible for `IdmFormValue ⇔ input value` transformation.

Renderers are registered in module's `component-descriptor.js` as single component with attributes:

- `id` - unique component identifier
- `type = form-value` - static component type is used for all `form-value` renderer
- `persistentType` - which persistent type renderer supports
- `faceType` - renderer face type ⇒ key. Unique face type should be given (by persistent type). Its optional `persistentType` is used as default, when no `faceType` is given.
- `component` - renderer implementation (`AbstractFormAttributeRenderer` descendant).
- `labelKey` - localization key ⇒ renderer name. Its optional, `faceType` is used, when no `labelKey` is given.

All component descriptor features are supported. Read tutorial, how to create custom form attribute renderer.

Custom configuration can be added to registered renderers (@since CzechIdM 10.8.0) - use `AbstractFormAttributeRenderer` on backend to define additional renderer properties.

Adding the support of extended attributes for a new entity

Backend

- Adding an interface implementation `FormableEntity` to the new entity,
- creating a manager implementation by inheriting `AbstractFormableService` for the new entity,
- creating the entity by inheriting `AbstractFormValue`, which will represent the values of extended attributes for the new entity (owner),
- creating a repository by inheriting `AbstractFormValueRepository` for the values of the extended attributes,
- creating the manager by inheriting `AbstractFormValueService` for the values of the extended attributes.

Frontend

- issuing a REST endpoint for saving the extended attributes from the FE - e.g. `IdmIdentityController` - controller has to evaluate security to read / save form values by their owner (e.g. by identity),
- creating a service and redux manager communicating with REST by inheriting `FormableEntityManager` - e.g. `IdentityManager`,
- using the component `EavForm` for filling in and sending the values of the extended attributes from the FE to the BE - e.g. `IdentityEav` content.

Agenda for working with forms

On the FE, there is an agenda of forms - their definition and attributes. Each definition can contain zero or more attributes. To maintain the integrity, an interface UnmodifiableEntity has been created, which allows adding a flag that the entity has been created by the system and cannot be modified (or some of its attributes) and deleted (this logic now needs to be implemented manually into the relevant controllers), for example in IdmFormAttributeController.

Data migration, when attribute's `persistentType` or `confidential` is changed is not supported now.

Localization

Beware, **form type, form code and attribute code** is used for composing the key for localization and in the string **all special characters (white spaces, dots, colons etc.) will by replaced by dash** (`spinal-case` or `kebab-case` on frontend).

Example form code **eu.bcvsolutions.idm.acc.entity.SysSystem** will be transformed into **eu-**

bcvsolutions-idm-acc-entity-syssystem

Read more in tutorial

Validation

validation

For form attribute values is possible to configure prepared validations. Validation are evaluated (**on the backend**), when form with extended attributes is saved and sent to backend. Simple validations as `required`, `min`, `max` are evaluated on frontend after value is changed.

Validations are suported for single attribute values only for now (feature request #1874).

Required

Value is required.

Unique

Value has to be unique.

Unique validation is not supported for `BYTEARRAY` and `ATTACHMENT` persistent types.

Min, Max

Value has to be greater than (lesser than) or equal given `min` (`max`) values. Real number (38,4) can be configured.

Min and max validation is supported for numeric `DOUBLE`, `INT`, `LONG` persistent types.

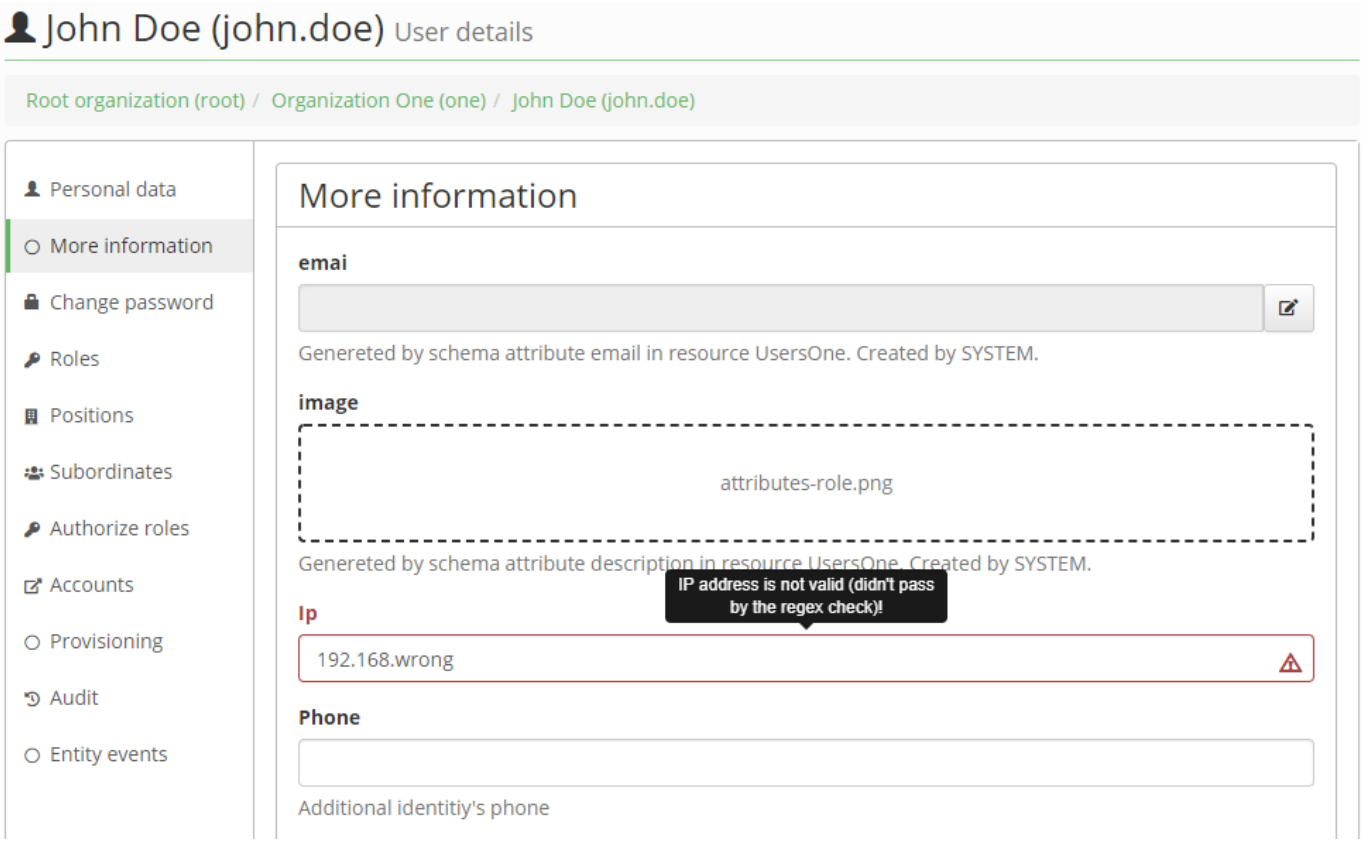
Regex

Value has to match given regular expression (java pattern is used).

Unique validation is not supported for `BYTEARRAY` and `ATTACHMENT` persistent types.

Use single back slash for configure regex on GUI ⇒ use double back slash in java. Example regex for the ip v4 address:

- GUI:
  `^([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])$`
- java:
  `^([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])\\.([01]?\\d\\d?|2[0-4]\\d|25[0-5])$`



Validation message

Custom validation message. If message is not defined, then default message by invalid validation type will be shown.

Can contain localization key (e.g. `core:validationError.invalid.unique`). Parameters `min`, `max`, `regex`, `unique`, `required` is available for localization message.

Code lists

Use `CodeListManager` for creating and providing code lists and items through application on backend (e.g. available for scripts, which could be used in provisioning).

Use `CodeListSelect` and `CodeListValue` for rendering code lists on frontend.

Frontend localization is supported in the item's name. For example item with name `environment.development.title` will be localized.

Authorization policies support

Identity form values can be secured by authorization policies when some identity extended attributes have to be secured - see how in configure authorization policies.

Authorization policies only support identity extended attribute values. Support for other entities can be added in the future.

Future development

- Form value data migration, when persistent type is changed.
- Attachment renderer: support multiple files, validation support (now is validation on input)
- Created deep copy, when form values are copied ⇒ attachment is linked to two form values and is removed, when the first one is deleted.
- #1874: Support unique validation for multivalued eav attributes