Events - processing of events

event, architecture, configuration, processor

An event mechanism has been designed to make extending and overlapping of the CzechIdM core functionality within any module possible.

The event (`EntityEvent`) with type (`EventType`) is published via `EntityEventManager` from different places in the application (⇒ `hook`). A number of processors can react to the event (`AbstractEntityEventProcessor`) in a defined order (number ⇒ the smaller it is, the sooner the processor is run). Processors run synchronously at default and in one transaction (see next section). Processors with the same order will be run in a random order (OrderComparator) - it's good practice to design and set different processor's order (think about it in design). Instead of the annotation `@Order`, the method `getOrder` needs to be overloaded (see the example). Event content could be any `Serializable` object, but `AbstractDto` descendant is preffered - see original source lifecycle feature. Event content is required, event without content couldn't exist.

Event lifecycle

1. Event is created with given content

```
EntityEvent<IdmIdentityDto> event = new
IdentityEvent(IdentityEventType.UPDATE, updateIdentity);
```

1. then is given to the processing via `EntityEventManager`

```
EventContext<IdmIdentityDto> context = entityEventManager.process(event);
```

1. when event is published and their content is descendant of `AbstractDto`, then original source is filled to the event - original source contains previously persisted (original) dto and could be used in "check modification" processors. If event creating new dto, then original source is `null`. Original source could be set externally - then no automatic filling occurs.
2. returning context contains results from all reacting processors in defined processors order.
3. the processor can label the event as (`closed`) or (`suspended`) and therefore skip all the other processors. If the suspended event is published again via `EntityEventManager`, the processing will continue where it was suspended, if context (with processed results) is preserved. If the processing of the event is (`suspended`), the called method should return the adequate `accepted` state.
4. when event walk through processors, then event's processed order is incremented - this order is used after event suspending and run again - event processing will continue with processor with the next order. Look out: when event is persisted, she will lost their context ⇒ when event is recreated from persistent state, she will continue with the next order.

Event types

Event is published with content and specific event type (e.g. CREATE, PASSWORD). Processors can be register by content type and event type - e.g. process event with `IdmIdentityDto` content and event type CREATE ⇒ other event contents and other event types will not be processed by this

processor.

Event types has to be compared by their string representation, NOT by instance. Concrete event types e.q. `IdentityEventType` are used for documentation reason only - which domain type supports which event. Event types can be added in different modules with different type, but processor can react across all module (⇒ is registered to string event type representation `eventType.name()`).

```
...
EntityEvent<IdmIdentityDto> event = new
CoreEvent<IdmIdentityDto>(CoreEventType.CREATE, identity);
if (event.hasType(IdentityEventType.CREATE)) {
 // do something
}
...
```

Module can publish their own event types. Basic (core) event types

- CREATE - synchronous, new entity is created
- UPDATE - synchronous, entity is updated
- DELETE - synchronous, entity is removed
- NOTIFY - asynchronous, notify about entity is changed (CU).
- EAV_SAVE - synchronous, some entity extended attribute is modified (CUD) - it is synonym for NOTIFY type ⇒ notify about entity extended attribute is changed (only extended attribute is changed). When entity is saved together with extended attributes, then only source event (e.g. CREATE) on owner entity is published (⇒ eav attributes are saved after owner entity is saved in once process line - e.g. CREATE).

Different (and custom) event types can be used for different entities.

Entities with event support

Supported events for individual entities:

- core:
  - IdmAuthorizationPolicyDto - authoriyation policies
    - supports base event types CREATE, UPDATE, DELETE
  - IdmAutomaticRoleAttributeDto - automatic role by attribute
    - supports base event types CREATE, UPDATE, DELETE
  - IdmAutomaticRoleAttributeRuleDto - rule of automatic role by attribute
    - supports base event types CREATE, UPDATE, DELETE
  - IdmIdentityDto - operation with the identity
    - supports base event types CREATE, UPDATE, DELETE, EAV_SAVE, NOTIFY
    - adds event type PASSWORD - changes password
    - adds event type PASSWORD_EXPIRED - password expires
  - IdmProfileDto - operation with the identity profile
    - supports base event types CREATE, UPDATE, DELETE, NOTIFY
  - IdmPasswordDto - operation with the CzechIdM password
    - supports base event types CREATE, UPDATE, DELETE

        - event is propagated when identity log in (e.g. last login date is changed)
    - `IdmTokenDto` - operation with the CzechIdM token
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
        - event is propagated when token is created, disabled (⇒ updated), deleted - e.g. when identity log in / out (token type `cidmst`)
    - `IdmRoleDto` - operation with the role
        - supports base event types `CREATE`, `UPDATE`, `DELETE`, `EAV\_SAVE`, `NOTIFY`
        - adds event type `DUPLICATE` - duplicate role, event is published for create / update role duplicate, read more here.
    - `IdmRoleGuaranteeDto` - operation with the role guarantee by identity
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
    - `IdmRoleGuaranteeRoleDto` - operation with the role guarantee by role
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
    - `IdmRoleRequestDto` - role requests
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
        - adds event type `EXCECUTE` - execute role request (*i know EXECUTE!, but it's too late ...*)
        - supports event type `NOTIFY` - is published, when request is completely approved or executed - provisioning listen this event mainly.
    - `IdmRoleCatalogueDto` - operation with the role catalogue
        - supports base event types `CREATE`, `UPDATE`, `DELETE`, `NOTIFY`
    - `IdmRoleCompositionDto` - operation with the business roles
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
    - `IdmIdentityRoleValidRequestDto` - role starts to be valid
        - adds event type `IDENTITY\_ROLE\_VALID`.
    - `IdmIdentityRoleDto`- assigning a role to the user
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
        - supports base event type `NOTIFY` - when parent role request event is propagated, then provisioning listener skips this elementary event processing and waits to role request is completed - listen `NOTIFY` on `IdmRoleRequestDto`.
    - `IdmIdentityContractDto` - labor-law relation
        - supports base event types `CREATE`, `UPDATE`, `DELETE`, `EAV\_SAVE`, `NOTIFY`
    - `IdmContractPositionDto` - other contract position
        - supports base event types `CREATE`, `UPDATE`, `DELETE`, `NOTIFY`
    - `IdmRoleTreeNodeDto` - automatic role by tree structure
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
    - `PasswordChangeDto` - used in password prevalidation only
        - adds event type `PASSWORD\_PREVALIDATION` - evaluates registered password policies - show policies setting before password is changed.
    - `IdmContractGuaranteeDto` - manually added guarantee to contract
        - supports base event types `CREATE`, `UPDATE`, `DELETE`, `NOTIFY`
    - `IdmTreeTypeDto` - tree structure type
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
    - `IdmTreeNodeDto` - tree structure node
        - supports base event types `CREATE`, `UPDATE`, `DELETE`, `EAV\_SAVE`, `NOTIFY`
    - `IdmPasswordPolicyDto` - password policy
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
    - `IdmLongRunningTaskDto` - long running task
        - supports base event types `CREATE`, `UPDATE`, `DELETE`
        - adds event type `END` - long running task ended.
    - `IdmEntityEventDto` - persisted event

        - supports base event types CREATE, UPDATE, DELETE
        - adds event type EXECUTE - executes persisted event.
    - IdmEntityStateDto - persisted entity / event state
        - supports base event types CREATE, UPDATE, DELETE
    - IdmFormInstanceDto - eav attribute values from single definition.
        - supports base event types UPDATE - update event type is used for saving eav attributes (⇒ CUD form values)
    - IdmCodeListDto - code lists
        - supports base event types CREATE, UPDATE, DELETE
    - IdmCodeListItemDto - code list items
        - supports base event types CREATE, UPDATE, DELETE
    - IdmNotificationTemplateDto - notification templates
        - supports base event types CREATE, UPDATE, DELETE
    - IdmIdentityProjectionDto - identity form projections
        - supports base event types CREATE, UPDATE
    - ModuleDescriptorDto - application modules
        - supports base event types INIT, ENABLE, DISABLE
    - IdmScripDto - groovy scripts
        - supports base event types CREATE, UPDATE, DELETE
    - IdmMonitoringDto - monitoring evaluators
        - supports base event types CREATE, UPDATE, DELETE
        - adds event type EXECUTE - run monitoring evaluator
    - IdmMonitoringResultDto - monitoring results (returned from evaluator)
        - supports base event types CREATE, UPDATE, DELETE
- acc:
    - AccAccountDto - Accounts on target system
        - supports base event types CREATE, UPDATE, DELETE
        - adds event type START - starts provisioning for given account.
    - AccIdentityAccountDto - Identity accounts on target system
        - supports base event types CREATE, UPDATE, DELETE
    - SysSystemDto - System in ACC module
        - supports base event types CREATE, UPDATE, DELETE
    - SysRemoteServerDto - Remote server in ACC module
        - supports base event types CREATE, UPDATE, DELETE
    - SysSystemMappingDto - Mapping between system and his mapping of provisioning or sync
        - supports base event types CREATE, UPDATE, DELETE
    - SysSchemaAttributeDto - Connector schema on system
        - supports base event types DELETE
    - SysProvisioningOperationDto - execute provisioning operation. Look out, persisting provisioning operation into queue itself doesn't support events. Events are added for executing operations from queue:
        - adds event type CREATE - execute provisioning for CREATE operation
        - adds event type UPDATE - execute provisioning for UPDATE operation
        - adds event type DELETE - execute provisioning for DELETE operation
        - adds event type CANCEL - cancels provisioning operation
    - AbstractSysSyncConfigDto (SysSyncConfigDto, SysSyncContractConfigDto, SysSyncIdentityConfigDto) - synchronization
        - supports base event types CREATE, UPDATE, DELETE
        - adds event type START - starts synchronization

        - adds event type `START\_ITEM` - starts synchronization of one item (~entity)
        - adds event type `CANCEL` - cancels synchronization
    - `SysSyncItemLogDto` - synchronization item
- `vs`:
    - `VsRequestDto` - Request for account change in virtual system
        - adds event type `EXCECUTE` - executes request
- `rpt`:
    - `RptReportDto` - generate report
        - adds event type `GENERATE` - generates request

A page has been created directly in the application on the module page for an overview of all entity types and event types migrating through event processing. All the registered processors including the configuration are listed there:

IdmIdentity

Modul	Název	Popis	Typy událostí	Pořadí	Neaktivní
acc	identity-password-validate-processor	Validates identity's and all selected systems password, when password is changed.	PASSWORD	-1000	☐
core	identity-password-validate-processor	Validates identity's password, when password is changed.	PASSWORD	-100	☐
core	identity-create-validate-password-processor	Validates identity's password before identity is created.	CREATE	-10	☐
acc	identity-delete-processor	Ensures referential integrity. Could not be disabled.	DELETE	-1	☐
core	identity-delete-processor	Deletes identity.	DELETE	0	☐
core	identity-save-processor	Persists identity.	CREATE, UPDATE	0	☐
core	identity-password-processor	Persist identity's password.	PASSWORD	100	☐
acc	identity-password-provisioning-processor	Identity's and all selected systems password provisioning.	PASSWORD	1000	☐
acc	identity-save-processor	Executes provisioing after identity is saved.	CREATE, UPDATE, EAV_SAVE	1000	☐

The default order for listeners (- ⇒ before, + ⇒ after).

Basic interfaces

- `EntityEvent` - an event migrating through the processors. The content of the event can be `BaseEntity`, **`BaseDto`** or any serializable content.
- `EventContext` - holds the context of the processed event - which processors it has been processed by, with what results, if the processing is suspended, closed, etc.
- `EventResult` - the result of processing of the event by one processor.
- `EntityEventProcessor` - event processor. Processor has to have unique identifier by module.
- `AsyncEntityEventProcessor` - asynchronous entity event processor. Processor can control asynchronous processing priority.
- `EntityEventManager` - ensures publishing of the event to processors.
- `EventableDtoService` - adds event processing support to service - event processors have to be provided (e.g. for save, delete).

Basic classes

- `AbstractEntityEvent` - an abstract event migrating through the processors - when adding a proper one can be simply inherited from.
- `DefaultEventContext` - the default context of the processed event - all abstract and default events and processors use it.

- `DefaultEventResult` - The default event result processed by one processor - all abstract and default events and processors use it.
- `AbstractEntityEventProcessor` - abstract event processor - when adding a proper one can be simply inherited from.
- `AbstractApprovableEventProcessor` - the event processor will send the whole event with dto (or serializable) content to WF for approval. It is necessary to configure the definition of the WF where the event will be sent to.
- `AbstractPublishEntityChangeProcessor` - publish asynchronous `NOTIFY` event.
- `DefaultEntityEventManager` - ensures publishing of the events to processors.
- `AbstractEventableDtoService` - adds default event processing support to service - event processors have to be provided (e.g. for save, delete).

AbstractEntityEventProcessor

Use this super class, when creating new processor implementation (this class contains some boring parts).

Methods, which have to be implemented:

- `getName()` - Unique (module scope) configurable object identifier. Its used in configuration key etc.
- `process(event)` - the main processors method with business logic.
- `getOrder()` - when will be processor processed. Processors are executed in defined order.

Methods, which could be implemented [optional]:

- `supports(event)` - Returns `true`, when processor supports given event. Default implementation takes processor's template entity class and event type given in constructor (or configured by `eventTypes` property).
- `conditional(event)` - Returns `true`, when processor supports given event. Returns `true` by default. Override this method for adding some condition.
- `isClosable()` - Returns `true`, when processor could close event (only documentation purpose now). Returns `false` by default.

AsyncEntityEventProcessor

Use this super class, when creating new asynchronous processor implementation.

Methods, which could be implemented [optional]:

* `getPriority(event)` - Registered async processor can vote about priority of processing for given event. Returns `null` by default ⇒ processor doesn't vote about priority - preserve original event priority. Use `IMMEDIATE` to execute whole event synchronously. All registered processors votes about event priority - whole event will be processed with the highest priority.

AbstractInitApplicationProcessor

Use this super class for providing application and module init data.

Transactions

Transactional processing is controlled before the event publishing itself - the whole processing now takes place in a one transaction and all processors run synchronously by default. In case of an error in any processor, the whole transaction is rolled back, which has some advantages:

- simple adding of validation or referential integrity
- repeating the whole chain

and disadvantages as well:

- having to catch all the exceptions properly to avoid "breaking the chain"
- saving logs and archives in the new transaction (`Propagation.REQUIRES_NEW`)

Event properties

Event properties (`Map`) can be specified for the event. Properties could be used in event processing.

Event properties are propagated automatically from parent into child event:

- if parent event is given, when child event is published
- if child event doesn't contain property with the same key - property can be preset manually and has a higher priority.
- properties needed for internal event mechanism are not propagated. Property keys can be found in `EntityEvent`.

When `NOTIFY` event is published and event will be processed synchronously (asynchronous event processing is disabled or event has `IMMEDIATE` priority), the properties set by processing `NOTIFY` event is also available in the original event.

Asynchronous event processing

CzechIdM 8.0.0 brings new feature - asynchronous event processing. New event type `NOTIFY` was added, all previous events (`CREATE`, `UPDATE`, `DELETE`, `EAV_SAVE` etc.) are still synchronous.

Asynchronous `NOTIFY` event is published for dtos:

- `IdmIdentityDto` - published, when identity is created or updated (or eav is saved, updated, deleted)
- `IdmIdentityContractDto`- published, when contract is created or updated (or eav is saved, updated, deleted)
- `IdmContractGuaranteeDto`- published, when contract guarantee is created or updated (or eav is saved, updated, deleted)
- `IdmIdentityRoleDto`- published, when identity role is created or updated (or eav is saved, updated, deleted)
- `IdmRoleDto`- published, when role is created or updated (or eav is saved, updated, deleted)

- `IdmRoleRequestDto`- published, when request is completely approved or executed - provisioning listen this event mainly.
- `IdmTreeNodeDto`- published, when tree node is created or updated (or eav is saved, updated, deleted)
- `IdmRoleCatalogueDto`- published, when identity role catalogue is created or updated (doesn't have eav attributes)

As you can see, entity `DELETE` event is still synchronous.

Other entities will be added soon, when new asynchronous entity event processors will be implemented.

`NOTIFY` event type is processed asynchronously:

- order **500** - by automatic roles in core module,
- order **1000** - by account management in acc module, then provisioning is executed.

When asynchronous event is published, it's persisted into event queue (`IdmEntityEvent`). Internal scheduled task executes events from queue - all registered processors for event type `NOTIFY` is processed - the same behavior as standard event processing, processors are called in defined order synchronously by defult. Event (~entity) states (`IdmEntityState`) are persisted during event is processed (created / running / failed). Successfully processed events are deleted from queue by processor EntityEventDeleteExecutedProcessor. When exception occurs, event stays in queue with appropriate result code. Event agenda is available under `APP_ADMIN` permission on frontend from audit menu (shortcut tab can be added on all entity details e.g. see identity detail).

Features

Process event from queue

Events from queue are processed by event owner id - one event for one owner can be executed in the same time ⇒ we need to preserven event order by created date for one owner. Super owner id (`EntityEventManager.EVENT_PROPERTY_SUPER_OWNER_ID`) can be used for setting custom event owner - this property will be resolved for evaluating running events for the same owner concurrency.

Events from queue can be deleted only (events without children can be deleted now from FE). Operation for retry failed events on truncate all events in queue will be developed in future.

Event priority

Before event is persisted into queue, then event priority is evaluated, priority types:

- `IMMEDIATE` - immediate ~ synchronously. Event will be executed synchronously.
- `HIGH` - asynchronously (7 / 10 in one cycle, batch size can be configured)
- `NORMAL` - asynchronously (3 / 10 in one cycle, batch size can be configured)

Events are processed from queue by internal scheduled tasks by priority. Events with `HIGH` priority will use 7 slots, events with `NORMAL` priority will use 3 slots ⇒ events will be processed **7 / 3**, when internal scheduled task for processing events will be executed.

Priority can be set to event manually or registered processors can vote about event's priority - see AsyncEntityEventProcessor - the highest priority is used.

Execute date

Execute date can be set to event manually. Event with priority `HIGH` or `NORMAL` will be processed after given date. Can be used for events, which could be executed sometimes "in night".

Parent event

Event can be published by another event ~ event chain (tree) is persisted. For example, when contract is saved, then contract `NOTIFY` event is published. This event is processed by provisioning processors - but only `NOTIFY` event with contract's identity is published here only. Provisioning is physically executed in other processor, which processes identity `NOTIFY` event.

Event parameters

When asynchronous event is published, then event content (and previous ~ original content) and event parameters is persisted into queue. This persisted attributes are used, when event is resurrected from queue and executed. Attributes are available in registered asynchronous processors - evaluate modifications, skipping by event parameter value etc. can be implemented in processors business logic.

Remove duplicate events

When internal scheduled task for executing event from queue is processed, then duplicate events are removed.

Duplicate event is event with the same:

- owner
- event type
- event properties
- original source (embedded dtos and audit fields are ignored).

Older duplicate events are removed - the newest event is used. Events are processed by priority in batch, default batch size can be configured ⇒ duplicates are removed only in this batch (not configurable for now, see future development). Batch size is designed this way, because events are

processed by priority - event with `HIGH` priority should not wait too long for another bulk is begin to process. Remove duplicates should be redesigned from scratch - remove duplicates through whole queue.

Entity state

Persist event (~entity) state, when event is processed. State can be persisted manually, even without event processing. This state will be shown on entity detail soon (new frontend component).

Notification

Notification about registered asynchronous processors is prepared, when asynchronous event is published. Notification is send into topic `core:event` - uses console log by default and is send to currently logged identity - e.g. identity is saved, but provisioning will be executed asynchronously. Localization for asynchronous processors was added on frontend (see key `acc:processor.identity-save-processor`).

Configuration

- Scheduler - configure internal scheduled tasks for processing events from queue.
- Event processing - configure asynchronous event processing
- Processors - configure entity event processors.

Predefined processors order

- **0** - basic / core functionality - operation `save`, `delete` etc.
- **50** - save eav, which are send together with owner's dto
- **100** - automatic roles computation.
- **1000** - after `save` - e.g. sends notifications.
- **10000** - publish `NOTIFY` event about entity is changed.
- **-1000** - before `delete` provisioning (before identity role is deleted).
- **Identity**:
  - **-2000** - validate password in acc module - checks all system password policies and idm default policy ⇒ all policies are evaluated in one request. If acc module is enabled, then core password validation processor can be disabled.
  - **-1000** - validate password in core module - checks idm default policy.
  - **100** - persist password
- **Contract**:
  - **100** - Automatic roles recount while identity contract is saved, updated or deleted / disabled.
  - **200** - Contract exclusion, end and enable.
- **LongRunningTask**:
  - **100** - execute scheduled long running tasks, which depends on currently ended scheduled task.
- **Provisioning**:

- **-5000** - check disabled system
- **-1000** - compute attributes for provisioning (read attribute values from target system)
- **-500** - check readonly system
- **0** - execute provisioning (create / update / delete)
- **1000** - execute `after` provisioning actions (e.g. sends notifications)
- **5000** - archive processed provisioning operation.

Other orders can be found directly in application, see **supported event types**.

Processor configuration

Processors can be configured through ``Configurable`` interface by standard application configuration.

Implemented processors

Basic processors for simple operations (e.g. save, delete) are not listed. All registered processors can be listed in agenda (Settings - Modules - Processors).

Automatic roles processors

```
##
## approve create automatic role
idm.sec.core.processor.role-tree-node-create-approve-processor.enabled=true
# wf definition
idm.sec.core.processor.role-tree-node-create-approve-processor.wf=approve-create-automatic-role
##
## approve delete automatic role
idm.sec.core.processor.role-tree-node-delete-approve-processor.enabled=true
# wf definition
idm.sec.core.processor.role-tree-node-delete-approve-processor.wf=approve-delete-automatic-role
```

Notification on change monitored Identity fields

- Check if defined fields on identity was changed. If yes, then send notification.
- As default is used this system template **identityMonitoredFieldsChanged**.
- Extended attributes is not supported now.
- Order of processor is **Integer.Max - 100**. We want to send notification on end of chain (after identity is presisted or provisioning are completed).

```
# Identity changed monitored fields - Check if defined fields on identity was changed. If yes, then send notification.
# Default is disabled
```

```
idm.sec.core.processor.identity-monitored-fields-processor.enabled=false
# Monitored fields on change (for Identity, extended attributes are not
supported)
idm.sec.core.processor.identity-monitored-fields-
processor.monitoredFields=firstName, lastName
# Notification will be send to all identities with this role
idm.sec.core.processor.identity-monitored-fields-
processor.recipientsRole=superAdminRole
```

Change user permissions workflow

- Name of processor "role-request-approval-processor".
- This process ensures the approval of the request for change premissions.

```
# Default is enabled
idm.sec.core.processor.role-request-approval-processor.enabled=true
# Workflow process for change permissions (as default is "approve-identity-
change-permissions")
idm.sec.core.processor.role-request-approval-processor.wf=approve-identity-
change-permissions
```

Change user permissions workflow - Approval by the helpdesk department

- The approving task will be assigned to all users with role Helpdesk.

```
# The role can be changed in the application configuration
"idm.sec.core.wf.approval.helpdesk.role", the default setting is Helpdesk.
idm.sec.core.wf.approval.helpdesk.role=Helpdesk
# Default is disabled
idm.sec.core.wf.approval.helpdesk.enabled=false
```

Change user permissions workflow - Approval by the manager

- The approving task will be assigned to all users evaluated as the managers of the applicant. The manager is defined based on the industrial relations of the applicant.

```
# Default is disabled
idm.sec.core.wf.approval.manager.enabled=false
```

Change user permissions workflow - Approval by the user administration department

- The approving task will be assigned to all users with role Usermanager.

```
# The role can be changed in the application configuration
"idm.sec.core.wf.approval.usermanager.role", the default setting is
```

```
Usermanager.
idm.sec.core.wf.approval.usermanager.role=Usermanager
# Default is disabled
idm.sec.core.wf.approval.usermanager.enabled=false
```

Hr processes processors

```
##
## HR process - enable identity's contract process. The processes is started
## for contracts that are both valid (meaning validFrom and validTill and
disabled state) and
## not excluded.
idm.sec.core.processor.identity-contract-enable-processor.enabled=true
# wf definition
idm.sec.core.processor.identity-contract-enable-
processor.wf=hrEnableContract
##
## HR process - end or delete of identity's contract process. The processes
is started
## for contracts that are not valid (meaning validFrom and validTill or
disabled by state) and deleted.
## If the processed contract was the last valid contract of the identity,
the identity is disabled.
## Additionally all added roles, which were assigned to the ended contract,
are removed by the process.
idm.sec.core.processor.identity-contract-end-processor.enabled=true
# wf definition
idm.sec.core.processor.identity-contract-end-processor.wf=hrEndContract
##
## HR process - identity's contract exclusion. The processes is started for
## contracts that are both valid (meaning validFrom and validTill) and
excluded.
## If the processed contract was the last valid contract of the identity,
the identity is disabled.
idm.sec.core.processor.identity-contract-exclusion-processor.enabled=true
# wf definition
idm.sec.core.processor.identity-contract-exclusion-
processor.wf=hrContractExclusion
```

Provisioning after create, update or delete manually added guarantee for contract

- Provisioning after manually add, update or remove guarantee is controlled by these two processors: **ContractGuaranteeSaveProvisioningProcessor** and **ContractGuaranteeDeleteProvisioningProcessor**. Provisioning for update or create is done **after** success save entity, but provisioning for delete is done **before** delete entity. Both processors are enabled by default.

```
## Provisioning identity after add or update IdmContractGuaranteeDto
idm.sec.acc.processor.contract-guarantee-save.enabled=true
##
## Provisioning identity before IdmContractGuaranteeDto will be removed
idm.sec.acc.processor.contract-guarantee-delete.enabled=true
```

LongRunningTaskEndProcessor

When some long running task ends, then END event is fired. This processor persists task's state.

LongRunningTaskExecuteDependentProcessor

When some long running task ends, then END event is fired. This processor executes scheduled long running tasks, which depends on currently ended scheduled task.

ReportGenerateProcessor

Processes GENERATE event type with `RptReportDto` content, order `-1000`. Generates output data for report by long running task.

```
## Enable / disable
idm.sec.rpt.processor.report-generate-processor.enabled=true
```

ReportGenerateEndProcessor

Processes GENERATE event type with `RptReportDto` content, order `0`. Saves generated report metadata (binary data are stored as attachment).

```
## Enable / disable
idm.sec.rpt.processor.report-generate-end-processor.enabled=true
```

ReportGenerateEndSendNotificationProcessor

Processes GENERATE event type with `RptReportDto` content, order `1000`. Sends notification after report is generated to report creator.

```
## Enable / disable
idm.sec.rpt.processor.report-generate-end-send-notification-
processor.enabled=true
```

IdentitySetPasswordProcessor

Processes UPDATE event type with IdmIdentityDto content, order 200. When identity starts to be valid and has at least one account on target system, then new password is generated and changed on all identity's accounts ⇒ identity ha the same password in all accounts. Notification is send (see acc:newPasswordAllSystems template) to identity about new password on which accounts.

Identity is starting, when their state is changed from CREATED, NO_CONTRACT, FUTURE_CONTRACT to the VALID state.

```
## Enable / disable
idm.sec.acc.processor.identity-set-password-processor.enabled=true
```

EntityEventStartProcessor

- Event content: IdmEntityEventDto
- Event type: EXECUTE
- Default order: **-1000**

Start execution of entity event.

```
## Enable / disable
idm.sec.core.processor.entity-event-start-processor.enabled=true
```

EntityEventExecuteProcessor

- Event content: IdmEntityEventDto
- Event type: EXECUTE
- Default order: **0**

```
## Enable / disable
idm.sec.core.processor.entity-event-execute-processor.enabled=true
```

Execute entity event - resurrects entity event and process her - execute all registered processors.

EntityEventEndProcessor

- Event content: IdmEntityEventDto
- Event type: EXECUTE
- Default order: **1000**

End execution of entity event - persist state only.

```
## Enable / disable
idm.sec.core.processor.entity-event-end-processor.enabled=true
```

EntityEventDeleteExecutedProcessor

- Event content: IdmEntityEventDto

- Event type: EXECUTE
- Default order: **5000**

Delete successfully executed entity events.

```
## Enable / disable
idm.sec.core.processor.entity-event-delete-executed-processor.enabled=true
```

RoleCompositionAfterCreateProcessor

@since 9.0.0

- Event content: `IdmRoleCompositionDto`
- Event type: `NOTIFY`
- Default order: **0**

Assign sub roles for currently assigned roles, after composition (business role) is created. Update role composition is not supported now - `NOTIFY` event is propagated, when composition is created only.

```
## Enable / disable
idm.sec.core.processor.core-role-composition-after-create-
processor.enabled=true
```

IdentityRoleAssignSubRolesProcessor

@since 9.0.0

- Event content: `IdmIdentityRoleDto`
- Event type: `NOTIFY`
- Default order: **500**

Assign sub roles of currently assigned identity roles:

- assign direct sub roles only, works recursively
- prevents cycles (just for sure) - adds processed roles into event property

```
## Enable / disable
idm.sec.core.processor.core-identity-role-assign-subroles-
processor.enabled=true
```

IdentityRoleDeleteAuthoritiesProcessor

- Event content: `IdmIdentityRoleDto`
- Event type: `DELETE`
- Default order: **Integer.MAX_VALUE**

Checks modifications in identity authorities after role removal and disable authentication tokens.

```
## Enable / disable
idm.sec.core.processor.identity-role-delete-authorities-
processor.enabled=true
```

ContractPositionAutomaticRoleProcessor

@since 9.1.0

- Event content: `IdmContractPositionDto`
- Event type: `NOTIFY`
- Default order: **500**

Automatic roles recount while contract position is created or updated.

```
## Enable / disable
idm.sec.core.processor.core-contract-position-automatic-role-
processor.enabled=true
```

FormableSaveProcessor

@since 9.2.0

- Event content: `FormableDto`
- Event type: CREATE, UPDATE
- Default order: **50**

Persists formable entity's (owner's) prepared eav attribute values.

```
## Enable / disable
idm.sec.core.processor.core-formable-save-processor.enabled=true
```

FormInstanceSaveProcessor

@since 9.2.0

- Event content: `IdmFormInstanceDto`
- Event type: UPDATE
- Default order: **0**

Persists form instance (eav attributes).

```
## Enable / disable
idm.sec.core.processor.core-form-instance-save-processor.enabled=true
```

RoleCodeEnvironmentProcessor

@since 9.3.0

- Event content: `IdmRoleDto`
- Event type: `CREATE`, `UPDATE`
- Default order: **-100**

Appends environment into role code. Checks filled code, base code and environment.

```
## Enable / disable
idm.sec.core.processor.core-role-code-environment-processor.enabled=true
```

Duplicate role processors

Processors

Implemented processors in the product sorted by order of the processing:

DuplicateRolePrepareProcessor

@since 9.5.0

- Event content: `IdmRoleDto`
- Event type: `DUPLICATE`
- Default order: **-1000**

Prepares role's basic properties.

Register custom processor after this processor's order, if some role basic property has to be overriden (or filled by different business logic).

```
## Enable / disable
idm.sec.core.processor.core-duplicate-role-prepare-processor.enabled=true
```

DuplicateRoleSaveProcessor

@since 9.5.0

- Event content: `IdmRoleDto`
- Event type: `DUPLICATE`
- Default order: **0**

Here is the role persisted into database.

Register custom processor after this processor's order, if some related entities has to

be duplicated (e.g. guarantees).

```
## Enable / disable
idm.sec.core.processor.core-duplicate-role-save-processor.enabled=true
```

DuplicateRoleFormAttributeProcessor

@since 9.5.0

- Event content: `IdmRoleDto`
- Event type: `DUPLICATE`
- Default order: **50**

Duplicate role form attributes - parameters for the identity (~assigned) roles. Parameters are created for the target role or updated - extended attribute code is used for pairing.

Parameters provided to the bulk action form:

- **Duplicate role form attributes** - if role form attributes will be duplicated.

Configuration properties:

```
## Enable / disable
idm.sec.core.processor.core-duplicate-role-form-attribute-
processor.enabled=true
```

DuplicateRoleCompositionProcessor

@since 9.5.0

- Event content: `IdmRoleDto`
- Event type: `DUPLICATE`
- Default order: **100**

Duplicate configured role composition (sub roles by business role definition) and duplicate sub roles recursively. If the same environment is selected, the only role composition is created - existing sub roles are used. If the different environment (~target environment) is used, then sub roles with the same environment as original are duplicated recursively into target environment.

Parameters provided to the bulk action form:

- **Duplicate sub roles (by business role definition)** - if business role configuration will be duplicated (recursively).

Overidable methods (can be used for on the projects, e.g. example below):

- `duplicateRecursively` - Returns true, when role should be cloned recursively - can be overriden, if some role hasn't be cloned recursively, if doesn't exist on the target environment before.

- `includeComposition` - Returns true, when role composition should be included in the target role - can be overriden, if some role hasn't be cloned recursively, if doesn't have the same environment etc.

Configuration properties:

```
## Enable / disable
idm.sec.core.processor.core-duplicate-role-composition-
processor.enabled=true
```

Custom processor example

Core processor can be disabled and overriden by processor implemented in custom module, if behavior of the core processor has to be changed.

```
/**
 * Project specific processor for duplicate role composition.
 */
@Component(CustomDuplicateRoleCompositionProcessor.PROCESSOR_NAME)
@Description("Duplicate role - composition and recursion.")
public class CustomDuplicateRoleCompositionProcessor extends
DuplicateRoleCompositionProcessor {

    public static final String PROCESSOR_NAME = "custom-duplicate-role-
composition-processor";

    @Override
    public String getName() {
        return PROCESSOR_NAME;
    }

    /**
     * Returns true, when role should be cloned recursively
     * - it's not cloned, if application sub role doesn't exist on the
target environment before.
     *
     * @param event processed event
     * @param originalSubRole original sub role
     * @param targetSubRole duplicate sub role. {@code null} if target role
has to be created.
     * @return
     */
    @Override
    public boolean duplicateRecursively(EntityEvent<IdmRoleDto> event,
IdmRoleDto originalSubRole, IdmRoleDto targetSubRole) {
        return (targetSubRole != null && targetSubRole.getId() != null) ||
originalSubRole.getChildrenCount() > 0;
    }
```

```
    /**
     * Returns true, when role composition should be included in the target
role
     * - it's not included, when sub role doesn't have the same environment
     *
     * @param event processed event
     * @param composition source composition
     * @return
     */
    @Override
    public boolean includeComposition(EntityEvent<IdmRoleDto> event,
IdmRoleCompositionDto composition) {
        IdmRoleDto subRole = DtoUtils.getEmbedded(composition,
IdmRoleComposition_.sub);
        //
        return Objects.equals(event.getOriginalSource().getEnvironment(),
subRole.getEnvironment());
    }
}
```

DuplicateRoleAutomaticByTreeProcessor

@since 9.5.0

- Event content: `IdmRoleDto`
- Event type: `DUPLICATE`
- Default order: **200**

Duplicate configured automatic roles by tree structure. Automatic roles are duplicated recursively, if composition is duplicated recursively (see `DuplicateRoleCompositionProcessor` above).

Parameters provided to the bulk action form:

- **Duplicate automatic roles** - if automatic roles will be duplicated (both by tree structure and attribute).

Configuration properties:

```
## Enable / disable
idm.sec.core.processor.core-duplicate-role-automatic-by-tree-
processor.enabled=true
```

DuplicateRoleAutomaticByAttributeProcessor

@since 9.5.0

- Event content: `IdmRoleDto`
- Event type: `DUPLICATE`
- Default order: **300**

Duplicate configured automatic roles by attribute. Automatic roles are duplicated recursively, if composition is duplicated recursively (see `DuplicateRoleCompositionProcessor` above).

Parameters provided to the bulk action form:

- **Duplicate automatic roles** - if automatic roles will be duplicated (both by tree structure and attribute).

Configuration properties:

```
## Enable / disable
idm.sec.core.processor.core-duplicate-role-automatic-by-attribute-processor.enabled=true
```

2019/03/15 11:47 · tomiskar

Example

Synchronous processor

If we want to get hooked after updating the `identity`, we should implement a processor to the event type `IdentityEventType.UPDATE` with an order number higher than **0**:

```
@Enabled(ExampleModuleDescriptor.MODULE_ID)
@Component("exampleLogIdentityUpdateSyncProcessor")
@Description("Logs after identity is updated")
public class LogIdentityUpdateSyncProcessor
        extends CoreEventProcessor<IdmIdentityDto>
        implements IdentityProcessor {

    /**
     * Processor's identifier - has to be unique by module
     */
    public static final String PROCESSOR_NAME = "log-identity-update-sync-processor";
    private static final org.slf4j.Logger LOG = org.slf4j.LoggerFactory
            .getLogger(LogIdentityUpdateSyncProcessor.class);

    public LogIdentityUpdateSyncProcessor() {
        // processing identity UPDATE event only
        super(IdentityEventType.UPDATE);
    }

    @Override
    public String getName() {
        // processor's identifier - has to be unique by module
        return PROCESSOR_NAME;
    }
```

```
    @Override
    public EventResult<IdmIdentityDto> process(EntityEvent<IdmIdentityDto>
event) {
        // event content - identity
        IdmIdentityDto updateddIdentity = event.getContent();
        // log
        LOG.info("Identity [{},{}] was updated.",
updateddIdentity.getUsername(), updateddIdentity.getId());
        // result
        return new DefaultEventResult<>(event, this);
    }

    @Override
    public int getOrder() {
        // right after identity update
        return CoreEvent.DEFAULT_ORDER + 1;
    }
}
```

Asynchronous processor

If we want to implement the same feature as above but asynchronously, we can process asynchronous `IdentityEventType.NOTIFY` instead `IdentityEventType.UPDATE`. When we need to change synchronous processors to asynchronous, we can simply change processed event type and add some condition, when only some original event types has to be processed ⇒ asynchronous `NOTIFY` event type is published for CREATE, UPDATE and EAV_SAVE event types.

```
@Enabled(ExampleModuleDescriptor.MODULE_ID)
@Component("exampleLogIdentityUpdateAsyncProcessor")
@Description("Logs after identity is updated")
public class LogIdentityUpdateAsyncProcessor
        extends CoreEventProcessor<IdmIdentityDto>
        implements IdentityProcessor {

    /**
     * Processor's identifier - has to be unique by module
     */
    public static final String PROCESSOR_NAME = "log-identity-update-async-
processor";
    private static final org.slf4j.Logger LOG = org.slf4j.LoggerFactory
            .getLogger(LogIdentityUpdateAsyncProcessor.class);

    public LogIdentityUpdateAsyncProcessor() {
        // processing identity NOTIFY event only
        super(IdentityEventType.NOTIFY);
    }

    @Override
    public boolean conditional(EntityEvent<IdmIdentityDto> event) {
        // we want to process original UPDATE event only
```

```
        // async NOTIFY event is published for CREATE, UPDATE, EAV_SAVE
event types
        return super.conditional(event)
                &&
IdentityEventType.UPDATE.name().equals(event.getProperties().get(EntityEvent
Manager.EVENT_PROPERTY_PARENT_EVENT_TYPE));
    }

    @Override
    public String getName() {
        // processor's identifier - has to be unique by module
        return PROCESSOR_NAME;
    }

    @Override
    public EventResult<IdmIdentityDto> process(EntityEvent<IdmIdentityDto>
event) {
        // event content - identity
        IdmIdentityDto updateddIdentity = event.getContent();
        // log
        LOG.info("Identity [{},{}] was updated.",
updateddIdentity.getUsername(), updateddIdentity.getId());
        // result
        return new DefaultEventResult<>(event, this);
    }

    @Override
    public int getOrder() {
        // notify event has their own process line - we can use default
order
        return CoreEvent.DEFAULT_ORDER;
    }
}
```

Future development

- Skip duplicate event in "the bigger window".
- Support to persist confidential event properties.
- Support retry failed events
- Create FE component with entity state
- Notification about failed events (or report)
- Automatic role event doesn't propagate parent event id ⇒ processed by LRT.