

# Provisioning

identity, provisioning

Provisioning ensures the implementation of accounts settings on target systems according to the settings in IdM. The provisioning itself then only propagates information to the target system. It does not make the evaluation of which identities should be subject to provisioning on a particular system. This is the task of **accounts management** which is an integral part of IdM and precedes the provisioning itself.



Provisioning is an integral part of the ACC module (Account management)



Since version **9.4.0** are lists in provisioning compares **independent on order of values**. For example: If value from IdM is **[A,B,C]** and value from the system **[C,A,B]**, then this values are for provisioning same. It means none provisioning on system will be executed. If you need to ensure that comparing of lists will be dependent on value order, then you can override method **isAttributeValueEquals** in service **ProvisioningService**.

As it has been already said, the provisioning is preceded by **accounts management**. In most cases, provisioning is therefore run by the service ensuring accounts management. If this service evaluates that such a change of the account which requires provisioning has occurred, it will call the service [ProvisioningService](#). This service includes the following methods:

- **doProvisioning(AbstractDto dto)** - will do the provisioning for all the accounts themselves and systems related to a given dto (e.g. dto for identity, role, role catalogue, tree node).
- **doProvisioning(AccAccountDto account)** - will do the provisioning for this account (for the dto to which the account is related and which is marked as the account owner)
- **doProvisioning(AccAccountDto account, AbstractDto dto)** - will do the provisioning for this account and for the given dto (in case a link to the account marked as the "account owner").
- **doDeleteProvisioning(AccAccountDto account)** - will delete the account from the target system
- **changePassword(AbstractDto dto, PasswordChangeDto passwordChange)** - will change the password of a given dto (mainly is used for identity) also on the systems which have been set in the PasswordChangeDto object (contains also a new password). In PasswordChangeDto is collection AccAccountDto and for them will be change password.
- **doProvisioningForAttribute** - enables to do the provisioning only for a specific attribute. This is for example what needs to be done to change the password.
- **authenticate** - will perform the authentication test on the target system for a given username and system.



The term **account owner** stands for the link between an dto and an account (e.g. AccIdentityAccountnDto). Provisioning is done exclusively for the links which are marked as the **account owner**. Otherwise such a link is only an "informational" one. For example, the administrator's account may have more identities, but only its owner



may change it.

## Supported dto types

Provisioning is implemented for:

- IdmIdentityDto - identities
- IdmRoleDto - roles
- IdmRoleCatalogueDto - role catalogue items
- IdmTreeNodeDto - tree nodes (structures)

## Provisioning operation life cycle

- When dto (e.g. identity, see above) is changed, then update (create, delete) event on this dto is fired.
- in acc module are event processors, which processes update event on supported dto types and calls provisioning for updated dto - see above `ProvisioningService.doProvisioning(dto)`.
- `ProvisioningService` resolves concrete provisioning executor implementation (`ProvisioningEntityExecutor<DT0>`) for given dto and calls `ProvisioningEntityExecutor<DT0>.doProvisioning` ⇒ provisioning can be implemented different way for each supported dto type. Standard implementation is in `AbstractProvisioningEntityExecutor` - use this class, when provisioning for new dto type will be needed (contains some boring parts etc.).
- `ProvisioningEntityExecutor<DT0>` finds all dto's accounts and starts provisioning for them - fires `ProvisioningEvent START` with account as content. This is one place, when some module specific processor can extend provisioning functionality - see `ProvisioningStartProcessor`. As you can see - **accounts has to be prepared before**. Accounts are created by [account management](#). Account contains identifier only in this step - account management executes transformation for identifiers and creates / prepares accounts with them.
- `ProvisioningStartProcessor` is default processor (order 0), which processes `ProvisioningEvent START` and calls `ProvisioningEntityExecutor<DT0>.doInternalProvisioning(account, dto)`. In this step is `SysSystemEntityDto` found (by account identifier) or created (as "wish") for given account. `SysSystemEntityDto` is entity on the target system - we need to know real identifier on target system, which could be different with account identifier, etc.. **Account attributes are evaluated for given account** (create "wish" - what IdM want's to be provisioned to target system. Attributes transformation to the target system are called here). At last, provisioning operation (`SysProvisioningOperationDto`) is constructed with account attributes ("wish") and system entity (`SysSystemEntityDto`). Created provisioning operation is given to the provisioning executor (`ProvisioningEntityExecutor`), which operates above provisioning queue. He is responsible for assigning `SysProvisioningBatch` for given provisioning operation (read more bellow) and then he puts operation into the queue for processing ⇒ persists given `SysProvisioningOperationDto` with state `CREATED`. All provisioning

operations are stored in the queue. Operations in the queue can be processed two ways:

- **synchronously** - `ProvisioningEntityExecutor` fires `UPDATE (CREATE, DELETE)` event with `SysProvisioningOperationDto` content immediately, when provisioning operation is persisted. This is the default way. Then the queue is used as provisioning log, if everything is processed successfully. If some `ProvisioningException` occurs, then operation remains in the queue with state `EXCEPTION` and can be canceled or executed again (e.g. when target system was unavailable). Look out, make sure you handle all exceptions, which could be thrown in your custom provisioning processor and throw `ProvisioningException` descendants only (used `@Transactional` mechanism can handle `ProvisioningException` only and doesn't do rollback for them - this is needed to persist exception reason and operation state in the queue).
- **asynchronously** - target system can be switched to use [asynchronous provisioning](#), then `ProvisioningEntityExecutor` doesn't fire an event immediately. Long running task `ProvisioningQueueTaskExecutor` works above queue and fires event periodically for created operations placed in the queue. The same rule for `ProvisioningException` applies here.
- When `UPDATE (CREATE, DELETE)` event with `SysProvisioningOperationDto` content is fired ⇒ there is the place, where provisioning to target system really starts. Respectively - this is the place, when provisioning operation processing begins. All processors registered for `SysProvisioningOperationDto` content (see package `eu.bcvsolutions.idm.acc.event.processor.provisioning`) are executed in defined order (default):
  - -5000: `DisabledSystemProcessor` - checks disabled system before provisioning is called. If system is disabled, then operation remains in the queue as `NOT\_EXECUTED` and event processing ends.
  - -1000: `PrepareConnectorObjectProcessor` - actually changed attributes (connector attributes) are evaluated by given account attributes ("wish") and their [strategies](#). Target system is contacted - processor reads attributes on target system, creates diff (by attribute strategy etc.) and results are stored in provisioning operation context as connector attributes (actually changed attributes). This attributes can be found in provisioning operation detail - left side is the "wish" (IdM account attributes after transformations), right side is connector attributes (actually changed attributes). Used provisioning attributes are logged here, after connector object is prepared.
  - -500: `ReadOnlySystemProcessor` - checks readonly system before provisioning is called. If system is readonly, then operation remains in the queue as `NOT\_EXECUTED` and event processing ends. This mode can be used for debugging, if target system is online and we need to know, which attributes will be really provisioned.
  - 0: `ProvisioningCreateProcessor`, `ProvisioningUpdateProcessor`, `ProvisioningDeleteProcessor` - **execute provisioning to target system.**
  - 5000: `RemoveProcessedOperationProcessor` - successfully processed (or canceled) provisioning operation is moved from the queue to the archive. Archive is used as log. Archive can be truncated. When operation is moved into archive, then all used provisioning attributes are moved into archive too (is still available to find archived operation by used attributes).

## Provisioning queue

All provisioning operations are processed through provisioning queue. The individual provisioning operations (create, update, delete) are serialized, saved in the queue, and then processed. If

provisioning operation is completed, then archive provisioning operation record is created ⇒ active provisioning operation is executed and is moved into archive.

Three entities was created to persist provisioning operations and their state:

- **SysProvisioningOperation** - active (unresolved) provisioning operations. Contains operation type, state and provisioning context with transformed attributes ("wish" from IdM account), which have to be provisioned and really provisioned attributes (connector attributes), read more below.
- **SysProvisioningBatch** - container for active provisioning operations for one system entity on one system. Aggregates active operations for single dto (e.g. identity, role), respectively for their system entity. When provisioning fails or is stopped (e.g. when system is switched to be read only, disabled, asynchronous), then all provisioning operations for single system entity are grouped in one batch. Operations can be executed in the same order, as they was inserted into queue ⇒ preserve system entity state on target system. Operations from queue are processed by their batch ⇒ respectively batches are processed from queue.
- **SysProvisioningArchive** - archived (processed) provisioning operations - logged operations. Last archived provisioning operation can be used for getting provisioned context with contains really provisioned attributes.

Supported provisioning operation states:

- **CREATED** - newly created, not processed operation
- **EXECUTED** - the operation was successfully executed
- **EXCEPTION** - there was an exception during execution
- **NOT\_EXECUTED** - the operation was not executed because of some reason - operation for the same entity is already in the queue, readonly system etc.
- **CANCELED** - canceled by some reason (administrator etc.)

To extend or reconfigure the provisioning mechanism, the processing has been moved to the events on the dto **SysProvisioningOperationDto** and to the individual processors treating this dto (the list of the registered processors has been extracted to the application on the module page). Calling the provisioning then takes places through publishing the events with the **SysProvisioningOperationDto** content through **ProvisioningExecutor**. The individual operations are serialized, persisted in the queue, and transmitted to the processors. If a **ProvisioningException** occurs or some of the processors stops working, then it is possible to find what happened in the agenda placed above the queue (error code, return state etc.). The **GuardedStrings** (e.g. passwords) are not saved in the serialized state - the values are replaced and saved in a [confidential storage](#) from which they are only read when needed (account re-calculation, provisioning implementation).



Make sure you handle all exceptions, which could be thrown in your custom provisioning processor and throw **ProvisioningException** descendants only (used **@Transactional** mechanism can handle **ProvisioningException** only and doesn't do rollback for them - this is needed to persist exception reason and operation state in the queue).

The operation content in **SysProvisioningOperationDto** is called **ProvisioningContext** and is

divided into two logical units:

- **accountObject** - counted account attributes according to the set-up mapping on the IdM page ⇒ **wish**.
- **connectorObject** - real attributes sent to provisioning counted after contacting the target system (reading of the existent object and changes comparison).

First the **accountObject** ("wish"), then the **connectorObject** which is the real provisioning input is calculated. When repeating the operation, a new **connectorObject** from an **accountObject** is always calculated so that the possible changes which could occur right on the target system are taken into consideration.

Once the object (defined by the identifier on the system and the idm entity) is in the queue, all the other requests for the provisioning of the same objects are persisted and remains in the queue - consistent sequence of operations on one object in the target system.

Provisioning queue functions:

- **Start provisioning** - provisioning is started by publishing START event on IdM account, then provisioning operation for this account is created and inserted into queue for processing.
- **Disabled system** - requests for provisioning stays in the queue with state ``NOT\_EXECUTED``. The target system is not contacted at all.
- **Read only system** - requests for provisioning are prepared, the actually changed attributes on the target system are counted, the request is persisted and stays in the queue as ``NOT\_EXECUTED``. The check and reading of the original existing object on the target system take place - this is how the real operation UPDATE vs. CREATE is specified, if the object exists / does not exist on the target system. An active operation for provisioning does not take place.
- **Retry mechanism** - the requests ending with an error are persisted in the queue and new running time is scheduled to them = another attempt will be executed by long running task [RetryProvisioningTaskExecutor](#).
- **Possibility of asynchronous processing** - target system can be switched to use [asynchronous provisioning](#).
- **Break configuration**



More about attribute strategies (attributes merge and etc.) is [here](#).

## Provisioning attributes

Entity **SysProvisioningAttribute** contains attributes used in provisioning context connector object. Provisioning attribute fields:

- **name** - schema attribute name. Attribute name on the target system.
- **removed** - flag indicates, if attribute is used in context with empty value (null, empty) ⇒ attribute will be removed on the target system.

Is possible to filter provisioning operations and archives with used attributes. Is possible to filter empty provisioning (~ without attributes). Active provisioning operations in the queue are evaluated as empty only if attributes was already computed (after prepare provisioning object processor), so can be used for redonly systems, but only the first operation contains attributes (other operations are

put into queue without attributes).

**Older records** (created before version 9.6.3 was installed) **will be filtered as empty provisioning without attributes** (attributes are empty).

Search user or role

Development

admin

Josef Karel Tester, MBA (jtester, 1234567) User details

Karel Tester, MBA (jtester, 1234567)

Provisioning operations log

Active operations

Archive

Date

Result

Operation

System

With filled attributes (OR)

With removed attributes (empty attribute values, OR)

All operations ...

Filter


Result	Created	Processed	Operation	System name	Identifier in system	Request	Id
Executed	26.04.2021 10:11:29	26.04.2021 10:11:29	Update	VirtualTestMulti	jtester		ID:93c320af Ti:0001f376
Executed	26.04.2021 10:11:07	26.04.2021 10:11:08	Update	VirtualTestMulti	jtester		ID:91334990 Ti:04e83d77
Executed	26.04.2021 10:10:20	26.04.2021 10:10:20	Update	VirtualTestMulti	jtester		ID:25fc19e6 Ti:9539c8aa
Executed	26.04.2021 10:09:25	26.04.2021 10:09:26	Update	VirtualTestMulti	jtester		ID:f163c99a Ti:7e1ffb7b
Executed	26.04.2021 10:09:10	26.04.2021 10:09:11	Update	VirtualTestMulti	jtester		ID:ec4b1bdd Ti:cf38d1ec
Executed	19.04.2021 11:10:33	19.04.2021 11:10:40	Create	VirtualTestMulti	jtester		ID:6d6fd4ed Ti:5d0105d3
Executed	19.04.2021 11:10:21	19.04.2021 11:10:21	Delete	VirtualTestMulti	jtester		ID:23d415a7 Ti:cf8b87f7
Failed	19.04.2021 10:29:09	19.04.2021 11:10:07	Create	VirtualTestMulti	jtester		ID:c489d077 Ti:4d2805fa
Failed	19.04.2021	19.04.2021	Create	VirtualTestMulti	jtester		ID:724f335c

Retry mechanism

Provisioning operations ending with an error are persisted in the queue and new running time is scheduled to them = another attempt will be executed by long running task periodically - long running task [RetryProvisioningTaskExecutor](#) configuration is needed. Only failed operations are processed from queue by retry mechanism.

Asynchronous provisioning

Target system can be switched to use asynchronous provisioning - flag on the system detail. Then requests for active provisioning operations (create, update, delete) are persisted in the queue as ``CREATED`` and their processing is delayed. Operations in queue are processed by long running task [ProvisioningQueueTaskExecutor](#), which operates above the queue periodically and starts ``CREATED`` provisioning operation processing. Make sure you have [ProvisioningQueueTaskExecutor](#) configured, if you have some target system switched to use asynchronous provisioning.

**Change password operation is still synchronous** - is needed to change passwords immediately.



## Provisioning of attachment

### attachment

Since version **9.4.0** is supported provisioning for **EAV attributes** with **attachment**.

During provisioning is loaded ``IdmAttachmentDto`` first (by **ID of attachment** stored in the **EAV attribute value**). ``IdmAttachmentDto`` contains metadata (as file name, size, ...), but doesn't contains data. For it was created DTO ``IdmAttachmentWithDataDto``, where was added field **data** (byte array). Then are loaded data for this attachment and given input-stream is converted to the array of bytes. From this moment is value (for EAV attribute with attachment) replaced by instance of ``IdmAttachmentWithDataDto`` (contains metadata + data). It means the transformation to the system will have as input value (attributeValue) this new DTO.



Structure of **IdmAttachmentWithDataDto** is [here](#).

This **DTO** is useful in situation when we need to do provisioning for some attachment's metadata field (for example **name of file**). In this case we only need to create **transformation to the system**:

```
// Transformation for get name of file
if (attributeValue != null) {
    return attributeValue.getName();
}
return attributeValue;
```

### Example of use

- We have system with attribute **image-attribute**. That attribute has in the schema type array of bytes (**[B]**).
- This attribute is mapped on EAV attribute **image**. It means for this attribute exists **EAV attribute definition** too.



The EAV attribute must have the **Attachment** type set.

- If provisioning for identity where attachment for EAV attribute **image** exist is executed, then data (in byte array format) are propagated to the system (**image-attribute**).



**None** additional transformation is **required**. Load of attachment and transformation to the byte array is automatical (**if transformation to the system is blank**).

Last  
update: 2021/04/26 14:26  
devel:documentation:provisioning:dev:provisioning <https://wiki.czechidm.com/devel/documentation/provisioning/dev/provisioning>

---

From:  
<https://wiki.czechidm.com/> - **IdStory Identity Manager**

Permanent link:  
<https://wiki.czechidm.com/devel/documentation/provisioning/dev/provisioning>

Last update: **2021/04/26 14:26**

