Synchronization - contractual relationship

sync, relationship, contract

Identity (contractual) relationship synchronization works mostly according to the same rules as identity synchronization. In this page we will described only extra behavior specific for this synchronization.

What is contractual relationship

They define the link between the identity and the tree structure. In the application, we advance the logic according to which every identity has at least one contractual relationship. **Typically one contractual relationship is equals to one contract in company for the identity.**

Actions after end of sync

Actions executes after end of sync are **not executed** when user **canceled sync**!

HR processes

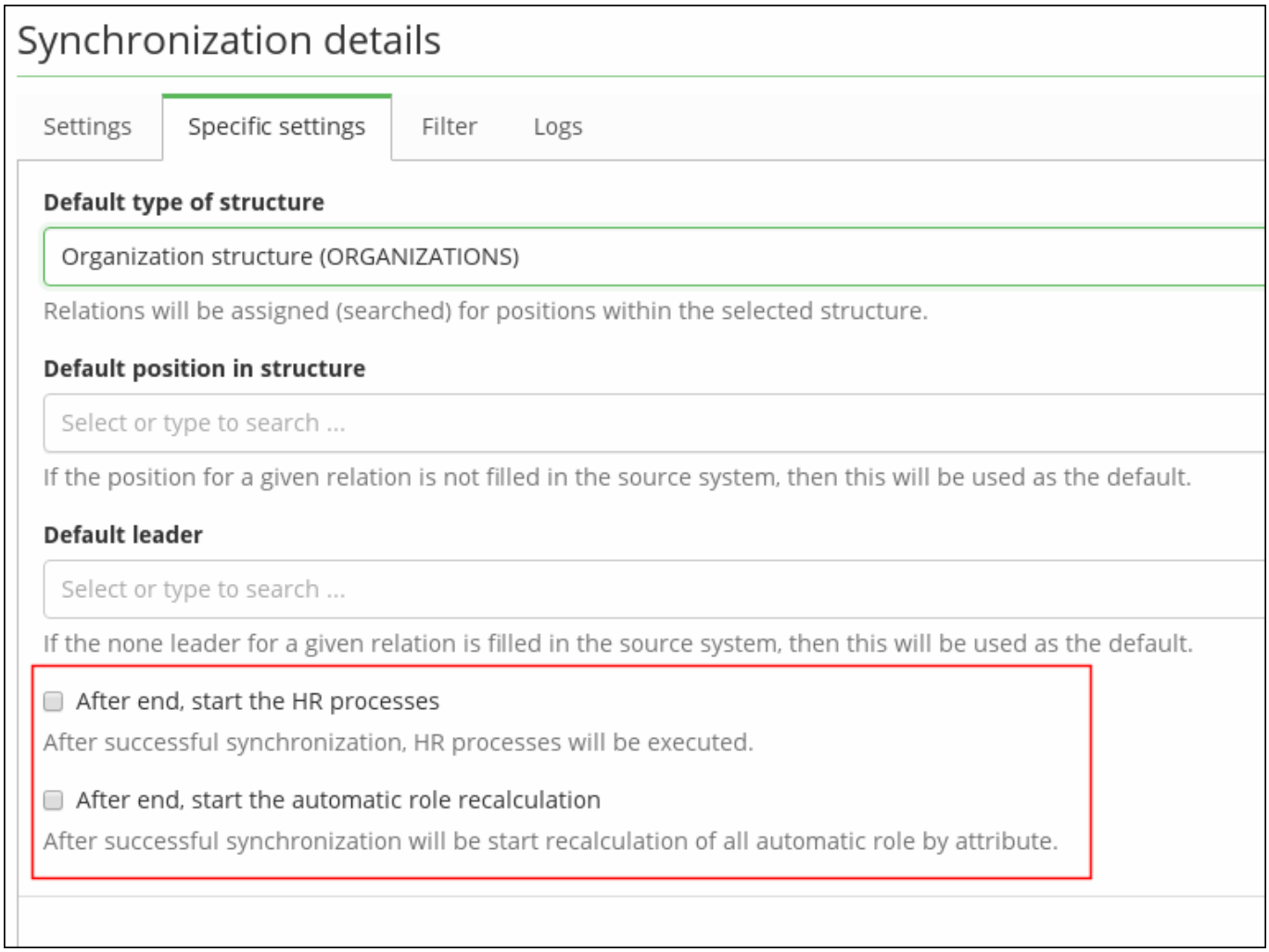
In new IdM installation must **HR processes be running at least once manually or by trigger**. Without this, sync will be not able **find HR tasks** (warning will be logged in sync log)!

HR processes in the base ensure the correct state of identity depending on the state of their contractual relationships. Because we need to evaluate the status of contractual relationships as a whole (to a given identity), it is not possible to trigger HR processes during the synchronization of each contractual relationship. Therefore, no HR processes are executed during this synchronization.

HR processes can be (**should be**) correctly started after the end of the sync. This can be ensured by the property `After end, start the HR processes` on the detail of sync configuration. If is this property ticked, then HR processes '**Enabled contract**', '**End of contract**', '**Contract exclusion**' (in this order) will be automatically started after correctly end of contract relationships sync.

Automatic roles

Recalculation of automatic roles is skipped during sync. Recalculation of automatic roles can be (**should be**) correctly started after the end of the sync. This can be ensured by the property '**After end, start the automatic role recalculation**' on the detail of sync configuration.



Fields for sync contractual relationship mapping

- **Owner** - Relation owner. Must be identity in IdM. This field is required for every relation. Output from attribute transformation can be:
  - ID of IdM identity in String or UUID format.
  - Username of IdM identity in String.
- **Main** - Define if is the contract main (between all contracts for the identity). Output from attribute transformation must be Boolean.
- **State** - State of contract. Output from attribute transformation must be enumeration ContractState or String representation for this enumeration (DISABLED, EXCLUDED) (more details see below).
- **Position** - String representation of contract. Typically name of contract.
- **Guarantees** - List of leaders, directly linked on the contractual relationship (more details see below).

- **Work position** - Define link to some tree node. Generaly define place in organization structure (more details se below).
- **Other positions** - List of other contract positions (more details se below).
- **Valid from** - Validity for the contractual relationship. Relation is 'active', only if is valid and state is null. Output value from attribute transformation must be 'org.joda.time.LocalDate'.
- **Valid till** - Validity for the contractual relationship. Relation is 'active', only if is valid and state is null. Output value from attribute transformation must be 'org.joda.time.LocalDate'.
- **Externe** - If is the contractual relationship for externe identity, then is output value (boolean true) .
- **Description** - String for description the relation.

Guarantees field

List of leaders, directly linked on the contractual relation. Linked leader must exists in IdM. Output from attribute transformation can be:

- Username of leader (String).
- Id of leader (UUID or String).
- List of usernames (List<String>).
- List of Ids (List<String> or List<UUID>).
- Null value. If is value not defined and in sync configuration has set '**Default leader**', then will be this leader set to relation.

If some leader will not found. Then will be synchronization item marked as 'warning' (relation will be created/saved). Detail information will be saved in item log:

```
.........................
Finding guarantee [temslie7].
.........................
Warning! - Identity [temslie7] was not found for [temslie7]!
.........................
```

Work position field

Define link to some tree node. Generaly define place in organization structure. Output from attribute transformation can be:

- Id of tree node (UUID or String).
- Code of tree node. Node by code will be searching in default tree (define in sync configuration '**Default type of structure**').
- Null value. If is value not defined and in sync configuration has set '**Default position in structure**', then will be this node set to relation.

If node will not found. Then will be synchronization item marked as 'warning' (relation will be created/saved). Detail information will be saved in item log:

```
.........................
Work position - try find directly by transformed value [Divanoodle]!
```

```
.......................
Work position - was not not found directly from transformed value
[Divanoodle]!
.......................
Work position - try find in default tree type [DEFAULT_ORG] with code
[Divanoodle]!
.......................
Warning - Work position - none node found for code [Divanoodle]!
```

When there is no work-position attribute defined in the mapping, then **none** default position will be set.

Other positions field

Define link to other contract positions - tree nodes. Generaly define other contract places in organization structure. Output from attribute transformation can be:

- List of Ids of tree nodes (List<String> or List<UUID>).
- Codes of tree nodes (List<String>). Node by code will be searching in default tree (define in sync configuration '**Default type of structure**').
- Null value - contract positions will be empty.

State field

State of contract. Output from attribute transformation must be enumeration ContractState or String representation for this enumeration.

ContractState have this values:

- **DISABLED**
- **EXCLUDED**

In some situations can be informations needed to determine result state in more than once source attributes.

For example we can have attribute '**state**' with one of values (10,20,30) and second attribute '**disabled**' (with value true/false). In this case states '**10**' and '**30**' marks that contractual relation is '**excluded**', but when attribute '**disabled**' will be 'true', then final state of relation must be '**DISABLED**'.

In some situations, we need evaluate values form more source attributes.

In this case you can use attribute '**icAttributes**' (in attribute transformation from the system). This attribute contains all object attributes from the system.

For resolve situation discrabed above was created transformation script '**compileIdentityRelationState**' (included in ACC module):

```
/**
* Compiles identity-relation state. Returns final state for the relation
* (contract). Uses input value as relation state and value from defined
* disabled attribute (from whole IC attributes ... comes from source system)
*
* Result for this script can be one value from [DISABLED, EXCLUDED, null].
*/

Logger log = LoggerFactory.getLogger(
        "compile-identity-relation-state-script");
log.info("Start 'Compile identity-relation state' script.");
/**
 * Name of attribute contains disable information (for identity relation) on
the
 * target system.
 */
final String disableAttributeName = "disabled";
/**
 * In this states is relation excluded;
 */
final String[] excludeStates = [ "10", "30" ];
/**
 * Define if is relation disabled;
 */
boolean disabled = false;

/**
 * Define state of relation comes from source system (assumes String value)
 */
String stateValue = null;

if(attributeValue != null) {
    if(!(attributeValue instanceof String))
    {
        throw new SynchronizationException(MessageFormat.format(
                "Value [{0}] for identity-relation state must be String, but
is [{1}] (System [{2}])", attributeValue,
                value.getClass(), system.getCode()));
    }
    stateValue = (String) attributeValue;
}

if(icAttributes != null){
    for (IcAttribute icAttribute : icAttributes) {
        if (disableAttributeName.equalsIgnoreCase(icAttribute.getName())) {
            Object disableValue = icAttribute.getValue();
            if (disableValue == null) {
```

```
                disabled = false;
            } else {
                if (disableValue instanceof Boolean) {
                    disabled = (boolean) disableValue;
                } else if (disableValue instanceof String) {
                    disabled = Boolean.parseBoolean((String) disableValue);
                }
            }
        }
    }
}

if(disabled){
    // Relation is disabled
    log.info(MessageFormat.format("'Compile identity-relation state' script
- relation is disabled (on system [{0}])", system.getCode()));
    return ContractState.DISABLED.name();
}

for(String excludeState:excludeStates){
    if (excludeState.equals(stateValue)) {
        // Relation is excluded
        return ContractState.EXCLUDED.name();
    }
}

// Relation is maybe active (depends on validity relation attributes too).
return null;
```

Correlation

Synchronization of contracts supports only correlation by simple text attributes. That means, if you already have some existing contracts and you want to pair them with accounts on some new source system, you have to use some e xtended attribute of contracts which will contain the identifier usable for correlation. Specifically, the attribute mapped to the entity attribute **Owner (identity)** , can't be used as a correlation attribute, otherwise the result of the synchronization would be an Unknown state and the following exception:

```
eu.bcvsolutions.idm.core.api.exception.CorrelationPropertyUnsupportedTypeExce
ption: Entity type
[eu.bcvsolutions.idm.core.model.entity.IdmIdentityContract] and property
[identity] has wrong type. Only String or UUID is supported now.
```

If you synchronize only new contracts from a source system, use simply the identifier as a correlation attribute and don't map the identifier to anything.

Tutorials

- Systems - CSV file: users contracts synchronization