Synchronization - tree nodes

sync, tree

An example of organizational structure synchronization can be found in the admin guide.

Basic algorithm

- Root search
- For each root are recursively searched all its child nodes (based on the equality where the **UID** (identifier) of the parent node equals **parent** attribute of the child node).
- Synchronization is started for each tree element.

Situation **The account does not exist** , it is solely based on a comparison of the existence of accounts on the target system against the existence of IDM accounts.

Finding tree roots

The roots of the tree are searched over the set of all accounts obtained from the source system. The reason why roots are not found using the **search** method on the end system is that their definition is in some cases too complex (the search criteria in the IC module are inadequate). Such a case is, for example, a situation where roots are all the elements (accounts) whose **parent** attribute are shown to themselves.

Root search is performed using the Groovy script in the synchronization configuration **Definition of tree roots** . This script runs over all system elements. If it returns **Boolean.TRUE** , then the element is root. If it returns **Boolean.FALSE** , it is not the root. The entry of this script is **account** (IcObject), an object of the element received from the IC module, so the names of the attributes are the names of the scheme attributes.

If the root definition script is not filled, then every element whose **parent** attribute is **null** is considered to be root.

Example of a script addressing the situation described above :

```
// Name of the attribute in the scheme attributes, which contains the
relation to the parent tree node
String PARENT_ATTR_NAME = "parent";
if(account){
 // Get value from parent attribute
 def parentValue = account.getAttributeByName(PARENT_ATTR_NAME).getValue();
 // Get value from ID attribute
 def uidValue = account.getAttributeByName("id").getValue();

 // Root is account, where is parent value equals with ID (externalId)
```

```
value.
 if(parentValue != null && parentValue.equals(uidValue)){
     // We need clear value of parent attribute. In IDM has roots always
parent = null.
     account.getAttributeByName(PARENT_ATTR_NAME).setValues(null);
     return Boolean.TRUE;
 }
}

return Boolean.FALSE;
```

How to synchronize all nodes under one already existing?

Sometimes we need to synchronize all nodes from the source system under one node which already exists in the IdM.

For definition of that '**Super parent**' node we cannot use:

- The transformation from the system (on 'parent' attribute), because we often use null value in the parent attribute as definition the root node.
- Selectbox on UI (configuration of the sync), because we sometimes want to use more 'Super parent' nodes (in case we have more roots and some of them should be under different 'Super parent').

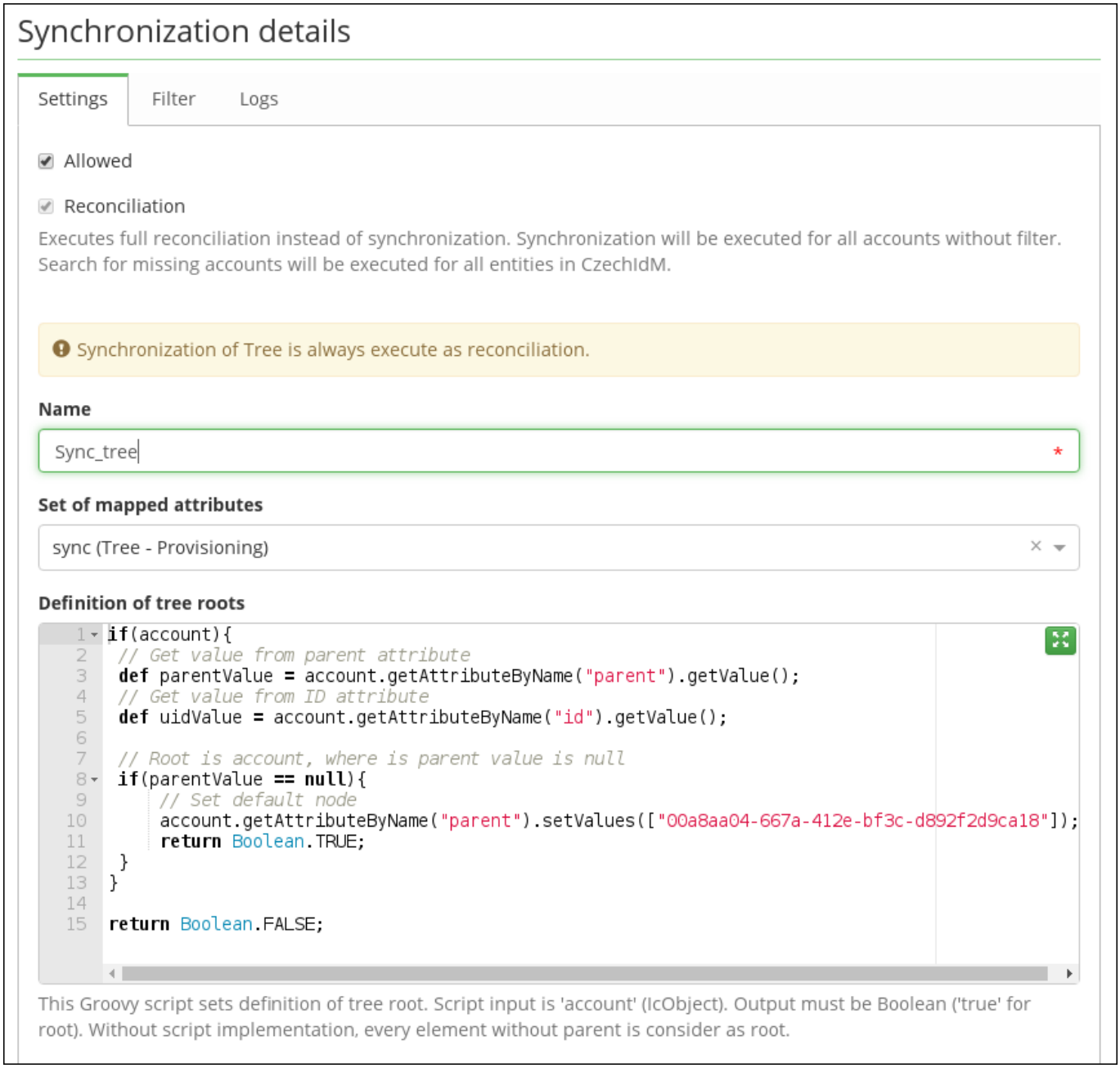
Super parent node can be defined in the script for "Definition of tree roots". This script is defined on the sync configuration and we can set **ID of super parent node** to **parent** attribute.

Using an ID instead of a node's code is an intent for optimization reasons. When searching for a super parent, IdM first checks if it is a UUID value, which is then much faster than searching the node by code.

```
// Name of the attribute in the scheme attributes, which contains the
relation to the parent tree node
String PARENT_ATTR_NAME = "parent";
if(account){
 // Get value from parent attribute
 def parentValue = account.getAttributeByName(PARENT_ATTR_NAME).getValue();

 // Root is account, where is parent value is null
 if(parentValue == null){
     // Set default node
account.getAttributeByName(PARENT_ATTR_NAME).setValues(["00a8aa04-667a-412e-
bf3c-d892f2d9ca18"]);
     return Boolean.TRUE;
 }
}
```

```
return Boolean.FALSE;
```



All roots in IDM must have **parent attribute = null**. If the root is defined in a different way by the source system (for example parent points on itself), then it is important to do the transformation for each root (see the example script above).

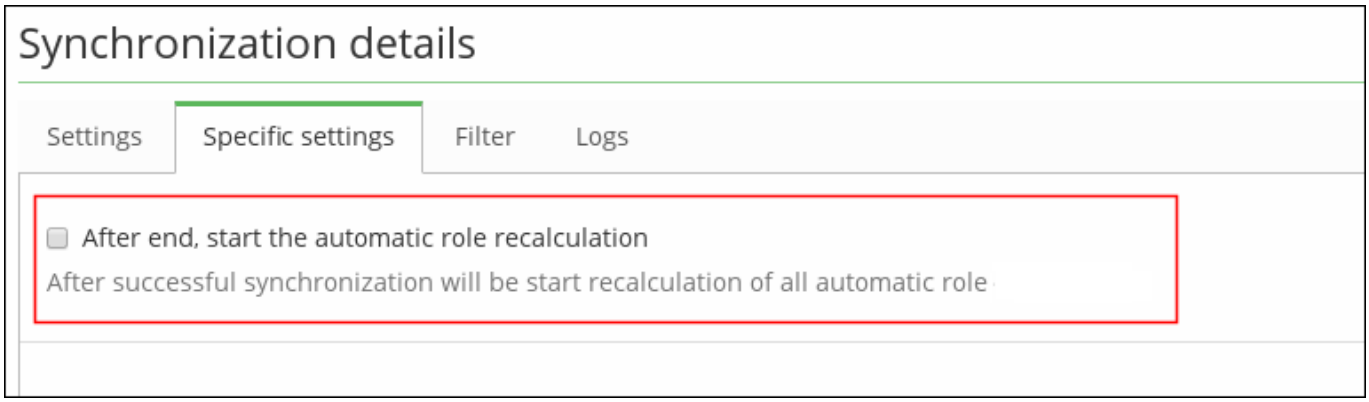
Leaving **uid** attribute and **parent** reference equal makes the synchronization loop infinitely - take care while setting the root computation script.

Actions after end of sync

Actions executes after end of sync are **not executed** when user **canceled 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. Automatic roles are recalculated for synchronized tree nodes, where tree structure (parent node) was changed.



When synchronization of contracts (or slices) is used in the same time on project (both are scheduled), then tree synchronization can be executed without automatic roles are recalculated after synchronization ends. Task `ProcessSkippedAutomaticRoleByTreeTaskExecutor` can be scheduled as dependent task to contract (or slices) synchronization (with automatic roles are recalculated properly). Recalculation of automatic roles for contracts should be recalculated first (new tree structure and automatic roles will be used), then automatic roles for changed tree nodes can be recalculated with `ProcessSkippedAutomaticRoleByTreeTaskExecutor` usage.