Inbound Mapping

Introduction

Inbound mapping is a mapping that transforms data from a projection (e.g. an account) on the resource side to a focal object (e.g. a user) on the midPoint side. Therefore it maps data in the direction which points *into* midPoint. Hence *inbound* mapping. Inbound mappings are usually used to copy and transform data from authoritative sources to midPoint. E.g. inbound mappings are often used to populate new user object with data from HR system.

Please see the Synchronization page for a generic description of the synchronization process and the specific place where inbound mappings are used.

Input

Value constructions are sometimes used in situations where an input of the construction is quite obvious. In such cases the input is placed into a variable named `input` for easier use. This is common practice e.g. in inbound expressions or in expressions describing synchronization of account activation and password.

Expression constructor using input variable

```
<script>
  <code>'The Mighty Pirate ' + input</code>
</script>
```

Association

Available since 3.7

This feature is available since 3.7

Since version 3.7 it is possible to manage group membership directly. There is no more need to define some user's extension attribute and manage the group membership indirectly using this attribute. The inbound mapping for association will do the work. As the example bellow shows, inbound can be defined in resource schema handling in the association part. All midpoint's expression can be used. In the following example the assignmentTargetSearch is used. According to the group in which user's take a membership the assignment will be constructed. The prerequisite is that each known group has it's corresponding role in midPoint. After applying search filter the role is found and it is assigned to the user.

Inbound example for association

```
 <association>
           <ref>ri:group</ref>
        <inbound>
                <strength>strong</strength>
                <expression>
                        <assignmentTargetSearch>
                                <targetType>RoleType</targetType>
                    <filter>
                                    <q:equal>
                                <q:path>name</q:path>
                                                <expression>
                                                    <trace>true</trace>
                                                        <script>
                                                                <code>
                                                                        return 'Auto' + entitlement?.getName()?.
getNorm();
                                                                </code>
                                                        </script>
                                                </expression>
                                   </q:equal>
                        </filter>
                  </assignmentTargetSearch>
                </expression>
                <target>
                        <path>assignment</path>
                        <set>
                    <condition>
                        <script>
                            <code>
                                import com.evolveum.midpoint.schema.constants.*
                                import com.evolveum.midpoint.xml.ns._public.common.common_3.RoleType;

                                if (assignment.target != null) {
                                        return assignment.target.roleType == 'auto'
                                }

                                if (assignment.targetRef != null) {
                                        role = midpoint.getObject(RoleType.class, assignment.targetRef.oid)
                                        return ('auto')?.equals(role.roleType)
                                }
                            </code>
                        </script>
                    </condition>
                        </set>
                </target>
        </inbound>
           <kind>entitlement</kind>
           <intent>group</intent>
           <direction>objectToSubject</direction>
           <associationAttribute>ri:members</associationAttribute>
           <valueAttribute>icfs:name</valueAttribute>
</association>
```

In the example bellow there is a script expression *return 'Auto' + entitlement?.getName()?.getNorm();* which tells midPoint which role should be assigned to the user. The *entitlement* is a special variable pointing to the ShadowType representing group on resource. This variable is available only for script expression in the relativity mode relative. In other cases when there is a need to resolve ShadowType for a group there is a method in midpoint function library which can be used. This method is shown in example bellow.

Resolving ShadowAssociationForGroup

```
groupShadowType = midpoint.resolveEntitlement(input);
```

Range Of Inbound Mappings

Inbound mappings have their range - as all the mappings have. Range is a set of possible values that a mapping can produce. This is an important tool to control which values are to be replaced by the mapping - or better to say, which values should be replaced. See Mappings page for the details.

Inbound mappings in midPoint 3.x were still a bit simplistic. At that time midPoint supported mostly mappings of single-value properties. The situation is quite clear for the single-value case. If the target property is single-value, then it cannot hold more than one value, therefore the value will get replaced all the time. Range definition is not that important here. However, the situation got complicated in late 3.x and 4.0. More and more deployments started to use inbound mappings for multi-value items, especially for assignments. And the situation can get quite complex when assignments are involved. In this case the range definition makes all the difference. However, the default range for midPoint 3.x was set to `all`, which means that all the target values got replaced. This is a bit problem for multi-value items, such as assignments. Therefore since midPoint 4.0 the default behavior was changed.

Since midPoint 4.0 the default range of inbound mappings depend on the target item:

Mapping target is	**Default range**	**Which means ...**
single-value	`all`	The target value will be replaced. This is nice and intuitive behavior for single-value items. This is also compatible with midPoint 3.x. Therefore this behavior was maintained for single-value items. This is also known as *non-tolerant* behavior.
multi-value	`none`	Target values will **not** be replaced. This is safe behavior for multi-value items as the chance to delete something is lower. This is also consistent with other mappings, where the default range is empty. This is mostly intuitive for many multi-valued items, such as assignments - even though it may not be a natural fit everywhere. But it is consistent behavior, therefore it was chosen as a default. This is also known as *tolerant* behavior.

The default behavior can be overridden by explicit definition of mapping range. For example, midPoint 3.x behavior for assignment mappings could be enabled by simply specifying `all` range for the mapping:

```
    <inbound>
        ... sources, expression, etc.
        <target>
            <path>assignment</path>
            <set>
                <predefined>all</predefined>
            </set>
        </target>
    </inbound>
```

See Mapping page for a detailed discussion of mapping ranges.

Motivation

The concept of range is not needed that often in other mappings. But it makes a lot of issues in inbound mappings. The reason for that is that inbound mappings are somehow different. They are not completely relativistic.

Typical midPoint mapping is working with a deltas. The mapping knows what was changed and the mapping takes a full advantage of that. Therefore it will normally not change the things that should not be changed. Therefore the definition of mapping range is not that important in such case. However, in inbound mappings we do not usually have a delta. We just have the state of the resource object (e.g. account) as it is now. We do not know what was changed. Therefore we have to recompute all the values. But the real difficulty here is to know which values to remove. E.g. if mapping target is an assignment, how do we know which assignments are given by this mapping and which are assigned automatically? The mapping should remove those that are given by the mapping, but it should not touch other assignments. This is exactly what range definition does.

For the future, there is an easier and perhaps more intuitive method. But that requires remembering the origin (provenance) of each value that midPoint maintains. Fortunately, such feature is planned: Data Provenance. If you are interested in this please consider purchasing platform subscription.

See Also

- Mapping
- Outbound Mapping
- Synchronization