Auxiliary Object Classes

- Static Use of Auxiliary Object Classes
- Dynamic Use of Auxiliary Object Classes
- Auxiliary Object Classes Inbound Mapping
- Read-only Auxiliary Object Classes
- See Also

Type of every resource objects that midPoint deals is defined by its object class. The object class defines the basic structure of the object: whether it is account, group or organizational unit. But some resources also have additional object classes that can extend the objects with additional attributes. We refer to these additional object classes as **auxiliary object classes**.

Every resource object must have exactly one structural (primary) object class. This object class is fixed. It cannot be changed. And then there may any number of auxiliary object classes. Auxiliary object classes can usually be added to or removed from an object at will.

The normal behaviour is that the objects have zero auxiliary object classes and vast majority of resources do not support auxiliary object classes at all. But there are some resources where auxiliary object classes are really useful, e.g. LDAP Connector and Active Directory Connector (LDAP) resources.

There are two ways to use auxiliary object classes in midPoint: static and dynamic (role-based).

**Static Use of Auxiliary Object Classes**

There are cases when auxiliary object class needs to always be present for a specific object type to work property. In that case simply add auxiliaryObjectClass configuration property to the usual object definition in resource configuration.

For example, there is a standard LDAP schema defined in RFC 2302 that is used to represent UNIX (NIS) accounts in LDAP servers. This schema defines the posixAccount auxiliary object class which can be added to ordinary account object classes. If this auxiliary object class is added then the account has UNIX-specific attributes such as uidNumber and gidNumber. The following configuration example illustrates the case of auxiliary object class that is always added to all the accounts on a resource:

```xml
<resource>
  ...
  <schemaHandling>
    <objectType>
      <kind>account</kind>
      <intent>default</intent>
      <objectClass>ri:inetOrgPerson</objectClass>
      <auxiliaryObjectClass>ri:posixAccount</auxiliaryObjectClass>
    ...
    <attribute>
      <ref>ri:uidNumber</ref>
      ...
    </attribute>
    </objectType>
  ...
  </schemaHandling>
</resource>
```

In this case you can use both the attributes of the structural object class and the attributes of the auxiliary object classes in the resource attribute mappings.

This is a typical example of a resource that is using the NIS schema (RFC2307) for all its accounts.

**Dynamic Use of Auxiliary Object Classes**

The static use of auxiliary object classes is simple and efficient. But having a directory server that is used only for the UNIX accounts somehow defeats the original idea of directory services: the directory service should work for as many systems as possible. There may be some companies where all users in fact have UNIX accounts. But a more realistic case is that some people have UNIX accounts and some people have not. Therefore we need a dynamic way how to assign auxiliary object classes to objects.

The mechanism that usually decides whether a user should have UNIX account or not is RBAC. If the user has a role that gives a UNIX account then the account should have posixAccount auxiliary object class. If the user does not have such a role there should be just the structural object class. MidPoint supports this configuration. In this case the resource configuration contains only the structural object class:

```xml
<resource>
  ...
  <schemaHandling>
    <objectType>
      <kind>account</kind>
      <intent>default</intent>
      <objectClass>ri:inetOrgPerson</objectClass>
      <auxiliaryObjectClass>ri:posixAccount</auxiliaryObjectClass>
    ...
    <attribute>
      <ref>ri:uidNumber</ref>
      ...
    </attribute>
    </objectType>
  ...
  </schemaHandling>
</resource>
```
All the interesting bits are in the role. The role defines that additional auxiliary object class has to be used when the role is assigned. And the role also contains mappings for the attributes that are specific to the auxiliary object class:

```
<role>
  <name>Unix User</name>
  <inducement>
    <construction>
      <resourceRef oid="8a327a26-7b2b-11e6-a2bb-2f9d7406d2c4"/>
      <kind>account</kind>
      <auxiliaryObjectClass>ri:posixAccount</auxiliaryObjectClass>
      <attribute>
        <ref>ri:uidNumber</ref>
        ...
      </attribute>
    </construction>
    ...
  </inducement>
</role>
```

When the Unix User role is assigned to a user, midPoint takes care that the posixAccount auxiliary object classes is added to the existing account and that all the necessary attributes are added as well. And (even more importantly) also the other way around: when the Unix role is unassigned midPoint will remove the posixAccount auxiliary object class and also all the attribute values. It is very important to remove the attribute values together with the auxiliary object class, otherwise the operation will end with an schema violation error. But midPoint takes care to work with auxiliary object classes properly.

**Auxiliary Object Classes Inbound Mapping**

TODO

```
<attribute>
  <ref>ri:uidNumber</ref>
  ...
</attribute>
```

**RFC2307 support in midPoint**

There are even more things to make really good configuration to manage LDAP server with the NIS (RFC2307) schema. Auxiliary object classes are just one part of the story. See the Unix Story Test for a complete configuration example.
Read-only Auxiliary Object Classes

There are situations where it is appropriate to keep the objectClass attribute set because it is defined outside the midPoint control. Such a case may be defining authorizations for external systems that are not and will not be controlled by midPoint.

To preserve them, it is possible to use the tolerant flag set to true within the definition of the basic schema mapping of the user object.

```xml
<schemaHandling>
  <objectType>
    <kind>account</kind>
    <default>true</default>
    <objectClass>ri/inetOrgPerson</objectClass>
    <auxiliaryObjectClassMappings>
      <tolerant>true</tolerant>
    </auxiliaryObjectClassMappings>
  </objectType>
  ...
</schemaHandling>
```

See Also

- Unix Story Test