Object ID

The Object ID (OID) is the universal internal identifier for all persistent objects.

Properties

- **System-wide unique**. The OID must be unique within the IDM system. No two objects may have the same OID. Not even two objects of different types. This property allows to use OID as a system-wide identifier for object references, as a primary key in tables, etc.
- **Probably globally-unique**. The probability that two OIDs in two different IDM systems are the same should be extremely low. The OID generation process should include some randomness to get this property. This provides easier migration of objects between systems (e.g. from test to production), copy&paste of objects, the ability to maintain references while copy&pasting, etc.
- **Human unreadable** (ugly, random-looking form). This property will discourage the practice of creating fixed, easy to remember OIDs. Fixed OIDs are hardcoded constants and should be avoided.
- **Relatively long**. This is an effect of other properties rather than a desired feature. However prepare the code, tools and processes that OIDs will be tens of characters long.
- **Not reassignable**. OID assigned to one object should not be assigned to any other object. This is not really a strict requirement, adding sufficient amount of randomness to OID generation should be just fine. If the OIDs cannot be reassigned we can easily detect broken links. If we ever reassign on OID, the link (reference) that should break will appear to be valid. Broken link is definitely a lesser evil in this case.
- **Internal to the IDM system**. OIDs should not be shared with any other system outside IDM. E.g. OIDs should be used as psoIDs in SPML. We want to keep OIDs internal to be able to regenerate them e.g. in case of upgrades to the version with different OID format.

UUID

The OID can (theoretically) be any string. MidPoint does not place any constraint on OID except for a reasonable length (limited by the underlying database). But the midPoint internal code and also all the examples are using UUIDs. The UUID seems to be almost ideal identifier for this purpose and therefore we recommend using this form of OID in all environments.

Generating OIDs

Ideally OIDs should be generated at a single place in the system to avoid identifier conflicts. It should be the lowest component that works with the database, so it can efficiently check OID uniqueness. And this is exactly what midPoint Repository Subsystem does. If it receives a new object without an OID it will generate a new (unique) one. If it receives an object with existing OID it will check that this OID is unique in the system.

Therefore there are (at least) two ways how to handle OIDs in practice: maintain your own and let midPoint generate them.

Maintain Your Own OIDs

This is usually the preferred way when you maintain midPoint configuration in (XML) files. Maintaining the configuration in files is a good way to repeatable midPoint deployments as the configuration can be kept in version control system, processed by scripts, etc. In this case the best practice seems to be to generate OIDs for each file manually and explicitly specify it in the file:

```
<role oid="e3b6a25c-c63c-11e4-a073-001e8c717e5b">
  <name>Foo</name>
 ...
</role>
```

This will fix the OID for this specific object. Therefore no matter how many times the object is imported it will always get the same OID. This comes very handy if the object has to be referenced from another object:

```
  ...
  <inducement>
    <targetRef oid="e3b6a25c-c63c-11e4-a073-001e8c717e5b" type="RoleType"/>
  </inducement>
  ...
```

In this case just copy & paste the OID.

Properly generate all OIDs. Try to avoid "semantically significant" OIDs such as `00000000-0000-0000-0000-000000000001` . Firstly, midPoint is using such OIDs for a couple of fixed special-purpose objects. Therefore using such OIDs for your own purposes is almost sure to create a conflict. Secondly, these OIDs require centralized allocation of identifiers. This is an error-prone task and maintenance burden. Avoid it. Use proper randomly generated OIDs even if you are generating them yourself. There are many handy utilities to do it, e.g. Linux `uuid` command-line tool.

MidPoint-Generated OIDs

This is the obvious choice when midPoint is configured from user interface wizards. The wizards will maintain the OIDs transparently.

If part of the configuration is still kept in the files it is OK ti omit OID in the file:

```
<role>
  <name>Foo</name>
  ...
</role>
```

If this file is imported midPoint will generate an OID on import. However, please keep in mind that midPoint will generate a different OID each time the object is imported. Therefore this is not good for keeping persistent links.

References and Search Filters

Object References are used in midPoint to refer from one object to another object. E.g. User object refers to linked accounts, roles refer to subroles, etc. The reference is always done by OID, so the reference will not break if objects are renamed. But how to maintain such references if object OIDs are generated by midPoint and they are different all the time? The reference can be "smart" and it may include a search filter:

```
<resource>
  ...
  <connectorRef type="ConnectorType">
    <filter>
      <q:equal>
        <q:path>c:connectorType</q:path>
        <q:value>org.identityconnectors.ldap.LdapConnector</q:value>
      </q:equal>
    </filter>
  </connectorRef>
  ...
</resource>
```

Such search filter will be executed at the time when the object will be imported. It looks for an object, gets its OID and places it in the reference. Therefore the reference will be store like this:

```
  ...
  <connectorRef oid="0885efd2-c63f-11e4-bb0c-001e8c717e5b" type="ConnectorType">
  ...
```

This comes very handy if you plan to combine parts of the configuration created in midPoint GUI with parts of the configuration maintained in files. But this is also needed if all the configuration is maintained in files, as there are objects that are usually maintained by midPoint automatically. Most notable the connectors.

Connector Objects (ConnectorType)

MidPoint automatically discovers local ConnId connectors. It crates objects that represent them. But as these objects are generated by midPoint, they also have randomly-generated OIDs. These are not fixed. Therefore even if you maintain configuration in files, the `connectorRef` reference in the resource usually needs to use a search filter. See Object References page for more details.

OID Maintenance Tips

- Stick to UUID format whenever possible. General string OID should be usable in midPoint, but we are not testing midPoint for this. And even though we do not plan any change in this aspect, there may be unexpected reasons and the OID format might be fixed to UUID in later midPoint versions.

See Also

- Object References
- Relaxed Referential Integrity

External links

- What is midPoint Open Source Identity & Access Management
- Evolveum - Team of IAM professionals who developed midPoint