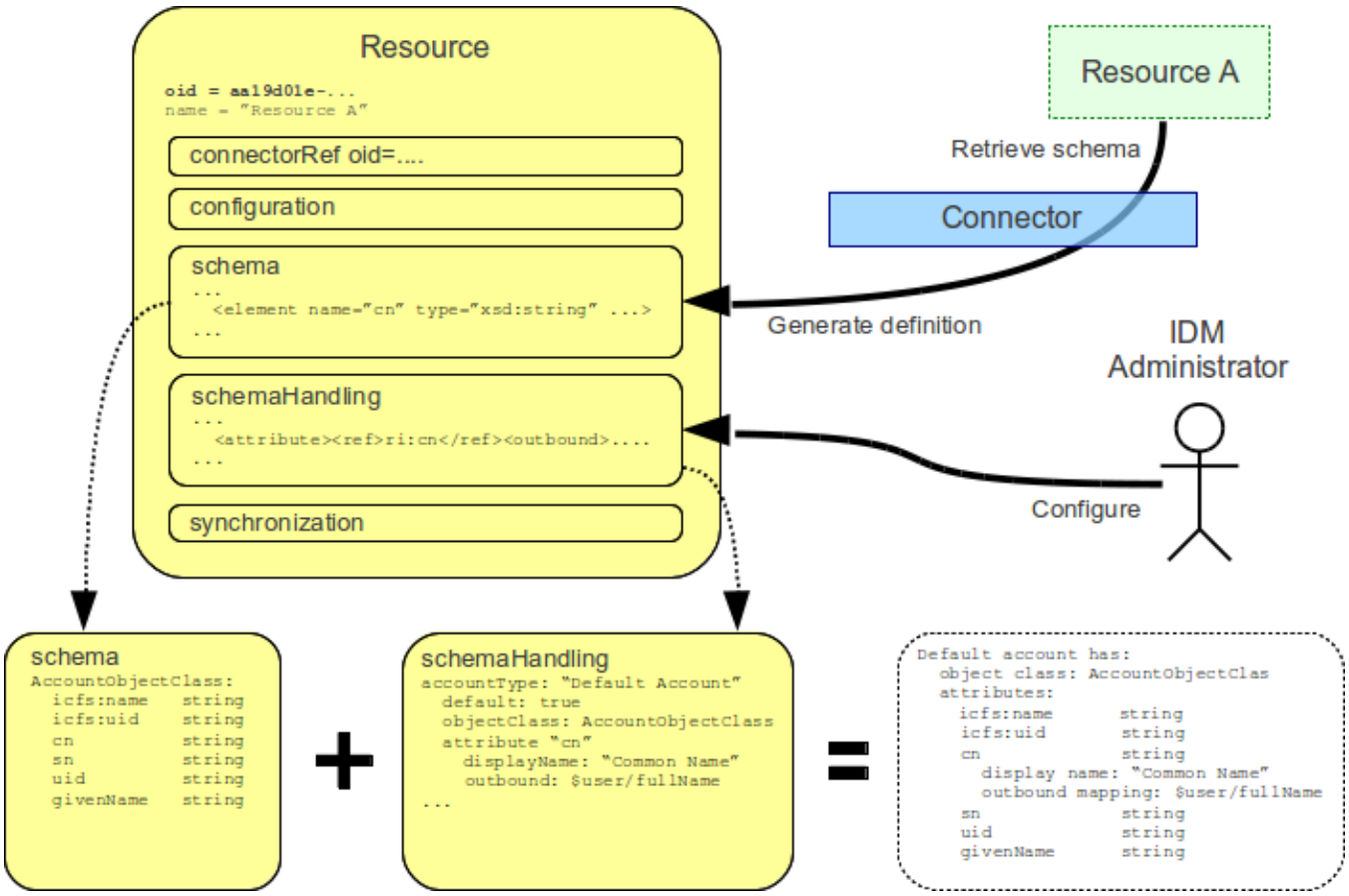
Resource Schema Handling

Introduction

Resource Schema Handling is a definition of how the Resource Schema is used by midPoint. Simply speaking, Resource Schema Handling defines how the individual attributes should be named, whether they are readable or writeable, how to fill the values of such attributes if new account is being created, how to use the attributes if a change is detected on the resource, etc.

The Resource Schema Handling is related to the Resource Schema, but there is a fundamental distinction:

- Resource Schema specifies **capabilities of the Resource and Connector**. If does not define how the schema is used (although it may suggest it).
- Resource Schema Handling specifies the **decisions of IDM administrators**. It defines how the Resource Schema is used by midPoint to implement parts of IDM logic, present the data to the user, etc.



While Resource Schema is usually generated from the Resource by the connector, the Resource Schema Handling must be manually defined by the IDM administrator. Adapting Resource Schema Handling is an substantial part of midPoint customization.

Schema handling defines account types and entitlement types. Accounts are data records representing user on the Resource, entitlements are objects that can be assigned to accounts (such as groups, roles, privileges, team membership, etc.). See Basic Data Model for the introduction to these concepts.

Resource Object Type Definition

By *resource object* we understand any object on the resource that is visible to midPoint. These are usually *account* objects but may also be a wide variety of *group* types, resource-specific low-level roles, privileges, organizational units, configuration objects, etc. Strictly speaking *resource objects* are the objects stored on the resource. When they are replicated to midPoint we call them *resource object shadows* (or just *shadows*). These two terms are often used interchangeably.

It is critical for a successful IDM deployment to correctly understand the meaning and usage patterns of resource objects for each resource connected to midPoint. Identity management is mostly just after manipulating resource objects after all. Therefore it is vital to configure midPoint to correctly understand and handle individual resource object types. Therefore major part of the *schema handling* section is dedicated to this task. The configuration may seem to be a bit complex at the first sight. But it is very flexible and powerful. Having this part of the configuration right significantly simplifies the overall IDM solution.

Kind and Intent

MidPoint identifies each type of resource object by using a **(kind, intent)** tuple. I.e. each resource object has a **kind** and an **intent**. Kind defines what the object *is*, intent specified how the object *behaves*.

See the Kind, Intent and ObjectClass page for more details on kind and intent.

Object Class

Object Class is a type of the object how the *resource* understands it. It is therefore a "technical" type of the resource object. Object classes are presented to midPoint by the connector. But midPoint usually does not know what to do with a specific object class, e.g. an LDAP objectclass `inetOrgPerson`. Therefore midPoint sorts out the object class to *kind* and *intent* as specified above. MidPoint then known quite well what to do with *default account*.

Object class is an term used by midPoint connectors. Therefore object class definition is passed to the connector.

See Kind, Intent and ObjectClass page for more details.

The Definition

Object type definition specifies a specific object type on the Resource should look like and how it should behave. The definition is identified by the (kind, intent) tuple. E.g. there is usually at least one definition for default account object type (kind=account, intent=default). The definition specifies what Object Class to use for such objects, how to handle the attributes, how to determine attribute values, etc.

The resource object definition specified using the `objectType` element in schema handling section. It contains:

- **Kind** of the resource object. If not specified it defaults to `account`.
- **Intent** of the resource object. If not specified it defaults to `default`.
- **Display Name** of the resource object type. This human-readable string is used in GUI and other tools when displaying information about resource objects.
- Indication whether the resource object type is the **default type** for the resource. If no intent is explicitly specified for a particular object kind then the default type will be used. Only a single intent can be marked as default for each object kind in the scope of a resource definition.
- **Object Class** of the resource object. This object class will be used when resource objects of this type are created. It must refer to exiting definition in the Resource Schema. This is the *structural* (primary) object class of the object. It defines all its fundamental characteristics. There must be precisely one (structural) object class for each object.
- **Auxiliary object classes** are additional object classes that define extra characteristics for the object. Auxiliary abject classes can usually be attached to objects of any structural object class. There may be any number of auxiliary object classes. Note: auxiliary object class support is available since midPoint 3.2.
- **Base context** is a point in a hierarchical tree under which objects of this type are stored. This is an optional feature that is only applicable to resources that have hierarchical structure (such as LDAP directories). Note: base context support is available since midPoint 3.4 and 3.3.1.
- **Attribute definitions** are definition that provide more details about handling of attributes that form a particular object type. The administrator can define a specific data and behavior for each attribute:
    - **Display name** of the attribute. This is a human readable name used in GUI and similar tools.
    - **Limitations** of the attribute. E.g. administrator can make the attribute read-only even if the connector can both read and write the attribute value.
    - **Outbound Mapping** defines how the attribute value is created when the account is created or updated. It defines the data flow out of midPont to the resource.
    - **Inbound Mapping** defines how the attribute value is used when a change of the account is detected on the Resource. It defines data flow from the resource into midPoint.
    - Other aspects such as tolerance, ability to ignore the attribute, etc.
- **Credentials** handling defines how credentials are synchronized for this account. Which usually includes only section for:
    - **Password** defines a password synchronization properties, it usually contains Outbound Mapping definition. It may also contain Inbound Mapping but as most resources hash their password this has a very limited use.
- **Activation** handling defines how the activation status of the account is synchronized. It usually contains section for:

    - **Existence mapping** determines when the object should and should not exist.
    - **Administrative status** mapping defines how activation administrative status is synchronized with the resource. It may contain outbound and inbound mappings.
    - **Validity** mappings defines how validity dates (`validFrom`, `validTo`) are synchronized with the resource. It may contain outbound and inbound mappings.
- **Iteration** section defines the parameters of iterative cycles. Such cycles are used e.g. when determining a unique identifier values from non-unique inputs. E.g. a iteration cycle may be used to try account identifiers `jack2`, `jack3} and {{jack4` in case there is already an account with identifier `jack`.
- **Protected** objectss definition. The accounts that are defined in the protected account section will never be touched by midPoint. MidPoint will not synchronize them, will not reconcile them and will not modify them. This is usually used to protect system accounts such as `root` or `administrator`.

The schema handling is additive to a resource schema definitions. This means that there is no need to define all the attributes from the object class in the schema handling section. The attributes that are defined in the object class and are not mentioned in schema handling are taken from the object class definition without any change.

Note: Only one `default` account type is supported by midPoint now. Support for more account types will come in the future.

TODO: example

accountType

The `objectType definition was called accountType in previous midPoint versions. Version 2.2 standardized the definition for all the resource object` *`kinds`*.

Entitlement Types

This is not implemented yet.

TODO: describe current design

Attribute Definitions

Attribute definitions are part of account definition. They provide more details about handling of attributes that form a particular account type. The administrator can define a specific data and behavior for each attribute. The basic and quite straightforward attribute definition elements are:

- **Display name** of the attribute. This is a human readable name used in GUI and similar tools.
- **Description** is a human-readable explanation of the attribute purpose, the purpose of the mappings, etc. It may be quite long.
- **Limitations** of the attribute. See below.
- **Matching Rule** of the attribute. See below.
- **Mappings** that define automated attribute handling. See below.
- **Tolerance** specifies whether the attribute tolerates values that are set outside midPoint. See below.
- **Fetch strategy** influences when midPoint will fetch this attribute. See below.
- **Exclusive strong** See below.
- **Read-Replace mode** See below.
- **Secondary identifier** See below.
- **Volatility Trigger** See below.
- **Modification Priority** See below.

Attribute Tolerance

Tolerance specifies whether the attribute tolerates values that are set outside midPoint. A *tolerant* attribute will tolerate foreign values in the attribute. E.g. if the attribute is a set of account privileges, setting it to tolerant will keep also the values set by native administration tools. On the other hand *non-tolerant* attributes will only allow values set by midPoint. If a foreign value is detected in the attribute then midPoint will remove that value during reconciliation.

All attributes are considered to be tolerant by default. This is in accord with midPoint philosophy to be non-intrusive by default and not to destroy any values unless explicitly said so.

The same principle applies to both single-valued and multi-valued attributes. However, there are subtle differences. MidPoint will almost always overwrite value of a single-value attribute. Even for tolerant attributes. This is quite obvious, as the attribute cannot hold more than one value and therefore the value that is provided by midPoint is probably the correct one. In case of tolerant multi-value attributes, midPoint will not overwrite existing values. The values provided by midPoint will be added to existing values of the attribute. However, midPoint may delete existing value of the attribute even if that attribute is tolerant. MidPoint will do that in case that such value is removed from midPoint (e.g. by unassigning a role) and that such value was given by authoritative mapping. In this case midPoint cannot reliably distinguish whether this particular value was added to the resource by midPoint or whether the value existed in the account even before midPoint discovered it. But the usual case is that midPoint added the value and that is what midPoint will assume in this case. Therefore such value is removed even if the attribute is non-tolerant. If you want avoid removing the value then you can set the mapping to be non-authoritative.

Tolerant single-value attributes

Single value attributes will usually be behave as expected, even if they are non-tolerant (which is the default setting). It means that mappings will overwrite the values and such attribute will behave almost in the same way as non-tolerant. But there is one crucial difference that becomes obvious in case that the mapping produces empty value. Tolerant attribute will **not** delete the attribute value in this case. And that makes sense, even though it is entirely intuitive. In this case midPoint has an option to keep the attribute value untouched. So it will not touch it. In case of non-empty value there is no option to keep the original value untouched because the target attribute can only hold one value. But in this case there is an option. On the other hand, non-tolerant attribute **will** delete the target value and then the mapping will work as expected.

Therefore it is **recommended to set most of the single-value attributes** for which there are mappings **to a non-tolerant mode**.

Even though this behavior may be somehow counter-intuitive, it makes perfect sense from the conceptual point of view. Single-value and multi-value attributes behave in a similar way. And keeping this aligned also allows to keep midPoint algorithms cleaner, handle less exceptions and special cases and it also gives midPoint a slight better flexibility. Therefore please forgive us this little non-intuitive weirdness.

Mappings

Perhaps the most powerful parts of the definition are mappings that take two slightly distinct forms:

- **Outbound Mapping** defines how the attribute value is created when the account is created or updated. It defines the data flow out of midPont to the resource.
- **Inbound Mapping** defines how the attribute value is used when a change of the account is detected on the Resource. It defines data flow from the resource into midPoint.

TODO: expand

Attribute Limitations

The limitations include

- **Ignore** flag, if set to `true` will make the attribute effectively disappear. The attribute will still be passed between midPoint and the resource, but the GUI and other parts of midPoint logic will pretend that it is not there.
- **Multiplicity override** by use of **minOccurs** and **maxOccurs** element. It can be used to adjust multiplicity of the attribute. The multiplicity is usually determined by the schema which is generated by the connector. However, the connector might provide a wrong schema. Or more commonly the schema is used differently as is formally defined. Perhaps the most common case is LDAP. Most LDAP attributes are defined as multi-value while vast majority of systems use them as single-value. The multiplicity override can be used to let midPoint think that these attributes are in fact single-valued.
- **Access** limitations of the attribute. E.g. administrator can make the attribute read-only even if the connector can both read and write the attribute value. The access limitation consists of three boolean switches:
    - **add**
    - **read**
    - **modify**

Attribute limitation example

```
<attribute>
    <ref>ri:cn</ref>
    ...
    <limitations>
        <minOccurs>1</minOccurs>
        <maxOccurs>1</maxOccurs>
        <access>
            <add>true</add>
            <read>true</read>
            <modify>false</modify>
        </access>
    </limitations>
    ...
</attribute>
```

The limitations can be expressed for several layers. Currently there are two layers defined:

- **Presentation** layer is aimed at the GUI and other forms of external data presentation such as an application behind a web service.
- **Model** layer defines midPoint internals. It is applied to mappings, internal schema validations, etc.

Separate set of limitations can be configured for each layer. This is often used to hide some attributes in the GUI while compute them in the model. Therefore such attribute needs to be ignored in the GUI but it has to be read-write in the model. Following example illustrates such configuration. A limitation that does not specify any layer applies to all the layers. The other limitations may modify that.

Attribute limitation with layers

```
<attribute>
    <ref>ri:cn</ref>
    ...
    <limitations>
        <minOccurs>1</minOccurs>
        <maxOccurs>1</maxOccurs>
        <access>
            <add>true</add>
            <read>true</read>
            <modify>true</modify>
        </access>
    </limitations>
    <limitations>
        <layer>presentation</layer>
        <ignore>true</ignore>
    </limitations>
    ...
</attribute>
```

Note for **version 2.1.1 and older**:

Versions prior to 2.2 used older limitation format that is quite limited it its expressive power. This format can still be used but it is considered deprecated and it will be not be supported in later releases. It also does not have the ability to work with layers.

Attribute limitation for version 2.1.1 and older

```
<attribute>
    <ref>ri:cn</ref>
    ...
    <minOccurs>1</minOccurs>
    <maxOccurs>1</maxOccurs>
    ...
    <access>create</access>
    <access>read</access>
    ...
</attribute>
```

Fetch Strategy

The `fetchStrategy` setting affects how and when midPoint retrieves value of this attribute. It is particularly useful in two cases: big attributes and attributes that are not returned by default. The `fetchStrategy` can have one of three values:

- **implicit**: MidPoint expects that the attribute will be implicitly returned by the connector in each fetch request and there is no need to explicitly request the attribute. This is the default.
- **explicit**: MidPoint expects that the attribute will NOT be implicitly returned by the connector. To fetch the attribute midPoint has to explicitly request it. Therefore midPoint will explicitly request this attribute in each fetch request. This setting is ideal for attributes that the connector does not return by default (e.g. operational attributes) but you want to see these attributes in midPoint anyway.
- **minimal**: Fetch the attribute only if absolutely necessary. MidPoint expects that the attribute might be implicitly returned by the connector. Therefore it will try to avoid fetching this value (if possible). This option can be used for values that cause performance overhead (e.g. list of members of large groups, big binary attributes and so on).

These three options can be very handy to tune midPoint performance - and specially the user interface performance. However, please note that proper functioning of these option depends on many things. Firstly the resource and the connector must properly support the "attributes to get" functionality. Smart resource and mature connectors such as LDAP support it. But other do not. It can be partially simulated in the ConnId layer. But this will address the issues only partially. Secondly, this feature depends on proper declaration of resource schema. E.g. if midPoint wants to avoid a fetch of a big attribute then midPoint has to request all the attributes except the one that we do not want. For that midPoint needs to know what other attribute names are. Most resource support schema properly and this works well. But there may be some connectors/resources where schema declaration is not entirely perfect.

Exclusive Strong

When set to false then both strong and normal mapping values are merged to produce the final set of values.When set to true only strong values are used if there is at least one strong mapping. Normal values are used if there is no strong mapping.

Default value is **false**.

Matching Rule

Specification of a matching rule for an attribute. Matching rule is used to compare values of the attribute. The default rule is a literal comparison which is good for most attribute types and for case-sensitive strings. An alternative matching rule may be specified e.g. for case insensitive strings.

Read-Replace Mode

Modifications to this attribute are executed in REPLACE form only. I.e. if ADD or DELETE VALUE is requested, midPoint will fetch the object state, compute the expected result and write it to the resource object via REPLACE VALUE operation. This works around some weird connector behavior. BEWARE: READ+REPLACE is currently supported for attributes only - not for subjectToObject associations.
EXPERIMENTAL. May change in near future.

Secondary Identifier

Indicated if the attribute should be considered as secondary identifier. If set to true, this attribue is stored in repository and user for example by synchronization (correlation rule), consistency mechanism, etc.

Volatility Trigger

If set to true it indicates that change of this attribute may cause changes in other attributes. In that case midPoint re-reads the object after the change of this attributes.

Modification Priority

Modification priority of this item. Items with specified priorities are modified in order that follows these priorities: these with lower numbers are modified first, these with higher numbers next, and items with unspecified priorities are modified last. Each priority level gets its own modify operation (or operations, if required by ICF limitations). Currently this field is supported only for attributes, even it is present on associations as well. (It is envisioned that 'addingPriority' could be created as well in the future; it would concern creating new objects. In that case, attributes with numerically lowest adding priority would be used to create an object, and other attributes would be set via MODIFY operation, again, according to their priorities.)

Resource Schema Annotations

To make the job of IDM administrator easier, well-written connectors will provide reasonable default values for some of the schema handling parameters. Such defaults are specified in the Resource Schema in a form of XSD annotations. See Resource Schema#Resource Schema Annotations for more details.

In the extreme case the entire Resource Schema Handling part is optional. All the details may be default to values defined in the Resource Schema annotations. But that is expected to be a very rare case.

See Also

- Resource Schema
- Shadow Objects
- Resource Schema Mapping
- Inbound Mapping
- Outbound Mapping
- Mapping Evaluation Examples

External links

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