Expression

Introduction

Expression is a part of midPoint configuration that contains logic. It is used in mappings, Correlation and Confirmation Expressions and may be used in other parts later on. Expressions are very flexible. They need to be flexible as expressions are the primary tool of customizing midPoint behavior. Expressions allow simple and efficient constructions that just pass or generate a value but they also allow invocation of complex scripts.

Simple expression example

```
<expression>
    <script>
        <code>
            givenName + ' ' + familyName
        </code>
    </script>
</expression>
```

The expressions are executed by invoking a specific *expression evaluator*. The evaluator is a piece of code that takes expression inputs (variables) and produces a value. There evaluators range from the very simples ("as is" evaluators) to flexible scripting language engines. Individual evaluators are described below.

Each expression take a set of variables as input. Each variable has a name and values. The values may also be "dynamic", not only specifying what the value is but also how it is being changed (delta). The expression than recomputes the variables to the output value, which may also be "dynamic". MidPoint expressions are designed and built especially to support the model of relative changes.

Expression Evaluators

Expression evaluators are individual clauses that create (or transform) a value. Each value construction needs to have at least one constructor to be useful.

Literal

Literal expression evaluator creates an explicit literal value. Single string value may be specified simply by placing it in the `value` tag as illustrated by the following two examples.

Single-value string literal expression

```
<expression>
    <value>Bloody Pirate</value>
</expression>
```

Multi-value literal expression

```
<expression>
    <value>Pirate</value>
    <value>Sailor</value>
</expression>
```

An empty value (combine with tolerant=false and strength=strong in mappings) can be expressed as:

Empty value expression

```
<expression>
  <value/>
</expression>
```

Complex values and multiple values must be wrapped in appropriate element for the literal expression to distinguish the value structure. Following example shows definition of two literal values for structured property `armament`.

Multi-value literal expression (alternative representation)

```
<expression>
    <value>
        <armament>
            <weapon>Cutlass</weapon>
            <placement>right hand</placement>
        </armament>
        <armament>
            <weapon>Pistol</weapon>
            <placement>belt</placement>
        </armament>
    </value>
</expression>
```

As Is

"As Is" expression is used if no value transformation is needed and there is a clear definition of what an *input* value is. The "as is" expression just copies this value to the output. It is usually used in inbound and outbound mappings to synchronize passwords and activation status directly to the resource without any change.

As Is expression

```
<expression>
    <asIs/>
</expression>
```

As is expression works only if the expression has exactly one source (input variable). If there is no source then the expression cannot produce any value. If there is more than one source then the expression cannot determine which one to user. Use the path expression in such cases (see below).

The As Is value is functionally almost identical to passing the value of `input` system variable to the output e.g. by using path expression (see below):

Path expression that simulates As Is expression

```
<expression>
    <path>$input</path>
</expression>
```

The As Is expression evaluator is the default evaluator that will be used if no expression is specified at all (e.g. in mappings).

Path

Path expression is the most reliable and most efficient way to reference a property or a variable. The expression simply takes path to the desired value.

Following constructor is referencing the `name` property in the object stored in `user` variable.

Path constructor referencing user's name property

```
<expression>
    <path>$user/name</path>
</expression>
```

The path expression may seem equivalent to the XPath script expression (see below). However there are several important differences. The XPath expression constructor is much slower as it needs to invoke XPath language interpreter. The path constructor is following the path through prism objects directly which is much faster. This also allows the path constructor to follow the modification mode (delta) of the source property which is very useful in some situations. Therefore as a rule of the thumb the path constructor should always be preferred if it is possible to use it.

Script

Script constructor is the most flexible of all the constructors as it allows to invoke an expression written in an expression or scripting language. There are several script languages to choose from. The script expressions are described in more details in a Script Expression page. This page only provides few examples.

Expression constructor using Groovy script

```
<expression>
<script>
  <language>http://midpoint.evolveum.com/xml/ns/public/expression/language#Groovy</language>
  <code>
    'uid=' + user.getName() + ',ou=people,dc=example,dc=com'
  </code>
</script>
```

Expression constructor using XPath script

```
<expression>
<script>
  <language>http://www.w3.org/TR/xpath/</language>
  <returnType>scalar</returnType>
  <code>
    concat('uid=', $c:user/c:name, ',ou=people,dc=example,dc=com')
  </code>
</script>
```

Expression constructor using ECMAScript (JavaScript) script

```
<expression>
<script>
  <language>http://midpoint.evolveum.com/xml/ns/public/expression/language#ECMAScript</language>
  <code>
    'uid=' + user.getName() + ',ou=people,dc=example,dc=com'
  </code>
</script>
```

See Script Expression page for more details.

Generate

The *generate* constructor is used to generate a random value. The value is generated according to the value policy. If there is a value policy already associated with a target property then it is sufficient to specify just plain `<generate/>` element. The applicable policy will be automatically determined and used. This usually applies to password policies. If no implicit policy is applicable to the target property or if a different policy is desired the policy may be overridden using `valuePolicyRef` element as illustrated below.

Generate constructor

```
<expression>
  <generate>
    <valuePolicyRef oid="d4c010c0-d34d-b3af-fe4d-11241a11101f"/>
  </generate>
</expression>
```

If no value policy is defined and the expression cannot determine the policy automatically it will use a reasonable default setting to generate random value.

Password policies and generate expression

When a generate expression without any parameters (`<generate/>`) is used to generate a password it will choose password policy automatically. When such an expression is used in a mapping it will choose password policy appropriate for the **mapping target**. This makes perfect sense, as the generated value must be a valid value for the target property. Which means that is the generate expression is used in the outbound mapping, it will use resource password policy. But if it is used in the inbound mapping, it will use user password policy. Because in the **inbound** case the target attribute is **user** password, not resource account password. The generate expression cannot use resource password policy because a password generate using that policy may not be a valid user password.

In case that you would like to change this behavior please specify the password policy explicitly using the `valuePolicyRef` parameter.

Assignment Target Search

Mappings and expressions are often used to create assignments. Therefore there is a special-purpose expression evaluator that simplifies the way how assignments are created. The evaluator is using a query to search for an target object in midPoint repository. When such object is found the evaluator creates an assignment for that target. This expression is especially useful in object templates.

Following configuration snippet provides an example of assignment evaluator that looks for an OrgType target:

Assignment expression

```
<expression>
    <assignmentTargetSearch>
        <targetType>c:OrgType</targetType>
        <filter>
            <q:equal>
                <q:path>c:name</q:path>
                <expression>
                    <path>$organizationalUnit</path>
                </expression>
            </q:equal>
        </filter>
    </assignmentTargetSearch>
</expression>
```

This assignment target search expression will look for objects of type OrgType in midPoint repository. It will look up the objects by `name` property. The name of the object should be the same as the value of `organizationalUnit` variable. If such an object is found than an appropriate assignment structure is created, the OID of the org object is placed inside it.

Search expression evaluators and includeNullInputs

Search expression evaluators have changed default for `includeNullInputs`. Null inputs are NOT processed by search expression evaluators by default. The reason is that null inputs are usually insignificant for search expression and skipping them results in fewer search operations. In case that processing of null inputs is needed it has to be explicitly turned on for search expression evaluators. This is usually needed in case that the evaluators should provide "default" values in case that some of the source values is not present. Simply speaking: if the expression is not producing a value that you would expect to be produced, turning on `includeNullInputs` will make midPoint slightly slower, but it may solve your problem.

Relation parameter

If you wish to assign the organization with relation value (such as "manager") to indicate any non-default relation, you need to specify it:

Assignment expression with relation parameter

```
<expression>
    <assignmentTargetSearch>
        <targetType>c:OrgType</targetType>
        <filter>
            <q:equal>
                <q:path>c:name</q:path>
                <expression>
                    <path>$organizationalUnit</path>
                </expression>
            </q:equal>
        </filter>
        <assignmentProperties>
            <relation xmlns:org="http://midpoint.evolveum.com/xml/ns/public/common/org-3">org:manager</relation>
        </assignmentProperties>
    </assignmentTargetSearch>
</expression>
```

After such assignment, GUI will indicate that user with this assignment is a manager of the organization.

Activation parameters

If you need to create assignment for a user with specific activation settings you can do it with following:

Assignment expression with activation parameters

```
<expression>
    <assignmentTargetSearch>
        <targetType>c:RoleType</targetType>
        <oid></oid>
        <populate>
                        <populateItem>
                                <expression>
                                        <script>
                                                <code>
                                                        import com.evolveum.midpoint.xml.ns._public.common.
common_3.ActivationStatusType
                                                        return ActivationStatusType.ENABLED
                                                </code>
                                        </script>
                                </expression>
                                <target>
                                        <path>activation/administrativeStatus</path>
                                </target>
                        </populateItem>
                        <populateItem>
                                <expression>
                                        <script>
                                                <code>
                                                        return basic.parseDateTime("yyyy-mm-dd'T'HH:mm:ss.SSS",
"2016-12-31T23:59:59.000");
                                                </code>
                                        </script>
                                </expression>
                                <target>
                                        <path>activation/validTo</path>
                                </target>
                        </populateItem>
                </populate>
    </assignmentTargetSearch>
</expression>
```

When the example above is user, each role assigned with it has administrativeStatus property set to the ENABLED and validTo date set to the 31.12.2016 EOD. This mechanism provide possibility to create assignment of roles, orgs, services with specific activation settings according to some focus attributes. The same mechanism can be used for defining role parameters and other attributes.

Create on Demand

The evaluator also has additional functionality that allows to create assignment targets on demand. This is a very useful functionality e.g. in case of opportunistic organizational structure synchronization when organizational unit names are only present as account attribute values and midPoint has to create appropriate orgs when it sees a new value. Following configuration sample extends the previous example with an create-on-demand functionality:

Assignment expression with create-on-demand configuration

```
<expression>
    <assignmentTargetSearch>
        <targetType>c:OrgType</targetType>
        <filter>
            <q:equal>
                <q:path>c:name</q:path>
                <expression>
                    <path>$organizationalUnit</path>
                </expression>
            </q:equal>
        </filter>
        <createOnDemand>true</createOnDemand>
        <populateObject>
            <populateItem>
                <expression>
                    <path>$organizationalUnit</path>
                </expression>
                <target>
                    <path>name</path>
                </target>
            </populateItem>
        </populateObject>
    </assignmentTargetSearch>
</expression>
```

New OrgType object will be created if no matching object is found by the query. The new object will be populated by the values specified by inner expressions (in `populateItem` elements).

Expressions inside expressions

Please note that the assignment expressions are part of the expression and it also usually contains inner expressions. So we have expressions inside expressions. This may look confusing at the first moment but in fact it goes very well in line with midPoint approach of reusability. We do not want to reinvent the same mechanism, we rather try to reuse what we already have. And this also creates a very powerful and flexible customization tool.

The assignment expressions can get very post-modern. E.g. one can have assignment expression inside assignment expression. Something like this:

functionality:

Assignment expression with create-od-demand configuration

```
<expression>
    <assignmentTargetSearch>
        <targetType>c:OrgType</targetType>
        <filter>
            <q:equal>
                <q:path>c:name</q:path>
                <expression>
                    <path>$organizationalUnit</path>
                </expression>
            </q:equal>
        </filter>
        <createOnDemand>true</createOnDemand>
        <populateObject>
            <populateItem>
                <expression>
                    <path>$organizationalUnit</path>
                </expression>
                <target>
                    <path>name</path>
                </target>
            </populateItem>
            <populateItem>
                <expression>
                    <assignmentTargetSearch>
                        <targetType>c:OrgType</targetType>
                        <filter>
                            <q:equal>
                                <q:path>c:name</q:path>
                                <expression>
                                    <value>TOP</value>
                                </expression>
                            </q:equal>
                        </filter>
                    </assignmentTargetSearch>
                </expression>
                <target>
                    <path>assignment</path>
                </target>
            </populateItem>
        </populateObject>
    </assignmentTargetSearch>
</expression>
```

This sample creates a new org on demand and such org will be assigned to the user. However the new org itself will have an assignment. In this case it is an assignment to some kind of "TOP" organizational unit. This is usually what is required as we do not want to create new top-level organizational units every time (see Organizational Structure for more details).

Association Target Search

TODO

Association From Link

TODO

Assignment From Association

TODO

Sequential Value

See Using Sequences.

Const

This feature is available in midPoint 3.6 or later.

Expression evaluator used to produce value of a constant.

See Configuration and Use of Constants for more details.

Expression Variables

See: Expression Variables

Extra Variables

Expression may define extra variables in addition to those provided by midPoint:

```
<expression>
    <variable>
            <name>jack</name>
            <objectRef oid="c0c010c0-d34d-b33f-f00d-111111111111" type="UserType"/>
    </variable>
    <path>$jack/givenName</path>
</expression>
```

Root Node

If value construction is used in a case where it is likely that most of the values will originate from a single object or a data structure such structure is assigned to the *root node* of the expression. The root node is kind of a default variable for the expression. Some expression languages can take advantage of the root node but most cannot. Therefore the *root node* mostly applies to XPath and similar languages. In XPath the root node can be addressed without a variable name. Therefore the following two expressions are equivalent (assuming that user is set as a root node).

Expression constructor using explicit variable

```
<expression>
    <script>
        <language>http://www.w3.org/TR/xpath/</language>
        <code>$c:user/c:name</code>
    </script>
</expression>
```

Expression constructor using root node

```
<expression>
    <script>
        <language>http://www.w3.org/TR/xpath/</language>
        <code>c:name</code>
    </script>
</expression>
```

Security

Run As

Expressions are normally evaluated using the security principal of the user that initiated the operation. This is best security practice as the authorizations go deep into the system and close to the data. In this it unlikely that an expression would read data or initiate an operation that the user is not authorized for. Therefore the probability of a security breach is reduced.

However, there are some cases when an expression needs access to data or operations that the use do not usually has. Since midPoint 3.6 the expression can be executed with the identity of a different user:

```
<expression>
    <runAsRef oid="e5e0f2fe-0aea-11e7-b02b-2b6815aa719e"/>
    <script>
        ....
    </script>
</expression>
```

The expression above will be executed with authorizations of the user identified by OID `e5e0f2fe-0aea-11e7-b02b-2b6815aa719e`. If the expression executes any operations that are audited, then this identity will also be used for auditing.

The variable `actor` that is present in most expressions still refers to the identity of the user that initiated the operations. This variable is not affected by the `runAs` configuration.

Security of Script Expressions

Script expressions are a code that runs inside midPoint servers. As such, script expressions are incredibly powerful. But with great powers comes great responsibility. Script expressions can do a lot of useful things, but they can also do a lot of harm. There are just a few simple internal safeguards when it comes to expression evaluation. E.g. midPoint script libraries will properly enforce authorization when executing the functions. However, script languages are powerful and a clever expression can find a way around this safeguards. MidPoint is **not** placing expressions in a sandbox, therefore expressions are free to do almost anything. The sandbox is not enforced from complexity and performance reasons, but it may be applied in future midPoint versions if necessary. For the time being, please be very careful who can define expressions in midPoint. Do not allow any untrusted user to modify the expressions.

See Script Expression Sandboxing for more details.

See Also

- Expression Variables