Bulk actions (midPoint scripting language)

An introduction

There are some situations when administrator needs to carry out actions on a set of objects. Some examples:

1. recompute all users without fullName,
2. delete all users that have no accounts,
3. disable all users with passwords older than 100 days,
4. assign an account on a resource to a people meeting given criteria,
5. assign a role to a people meeting given criteria.

MidPoint provides a specialized language to define such bulk actions. It is based on pipes-and-filters architectural pattern, where execution components (responsible for gathering data and acting on them) are connected into chains, pushing an output of an upstream action to be an input of the action that lies downstream. Just like good old Unix pipelines.

The scripts are currently represented using standard midPoint prism structures, externalized as XML, JSON or YAML documents. A specialized text-based language was once conceived (and experimentally implemented), but it is not currently available. Hoping its time will come it is described here.

Example 1: Recomputing all users without full name

```
<s:search xmlns:s="http://midpoint.evolveum.com/xml/ns/public/model/scripting-3"
          xmlns:c="http://midpoint.evolveum.com/xml/ns/public/common/common-3"
          xmlns:q="http://prism.evolveum.com/xml/ns/public/query-3">
    <s:type>UserType</s:type>
    <s:searchFilter>
        <q:equal>
            <q:path>fullName</q:path>
        </q:equal>
    </s:searchFilter>
    <s:action>
        <s:type>recompute</s:type>
    </s:action>
</s:search>
```

Example 2: Deleting all users that have no accounts

```
<s:search xmlns:s="http://midpoint.evolveum.com/xml/ns/public/model/scripting-3"
          xmlns:c="http://midpoint.evolveum.com/xml/ns/public/common/common-3"
          xmlns:q="http://prism.evolveum.com/xml/ns/public/query-3">
    <s:type>UserType</s:type>
    <s:searchFilter>
        <q:ref>
            <q:path>linkRef</q:path>
        </q:ref>
    </s:searchFilter>
    <s:action>
        <s:type>delete</s:type>
    </s:action>
</s:search>
```

Example 3: Disabling selected users

```
<s:search xmlns:s="http://midpoint.evolveum.com/xml/ns/public/model/scripting-3"
          xmlns:c="http://midpoint.evolveum.com/xml/ns/public/common/common-3"
          xmlns:q="http://prism.evolveum.com/xml/ns/public/query-3">
 <s:type>c:UserType</s:type>
    <s:searchFilter>
       <q:inOid>
          <q:value>5e0735fa-afee-4fd8-96a5-43eaf945adba</q:value>
          <q:value>b87eb285-b4ae-43c0-9e4c-7ba651de81fa</q:value>
          <q:value>469fd663-4492-4c24-8ce3-3907df7ac7ec</q:value>
       </q:inOid>
    </s:searchFilter>
    <s:action>
        <s:type>disable</s:type>
    </s:action>
</s:search>
```

Example 4a: Assigning an account on a resource to a people meeting given criteria

```
<s:search xmlns:s="http://midpoint.evolveum.com/xml/ns/public/model/scripting-3"
          xmlns:c="http://midpoint.evolveum.com/xml/ns/public/common/common-3">
    <s:type>c:UserType</s:type>
    <...>
    <s:action>
        <s:type>assign</s:type>
        <s:parameter>
            <s:name>resource</s:name>
            <c:value>ef2bc95b-76e0-48e2-86d6-3d4f02d3e1a2</c:value>
        </s:parameter>
    </s:action>
</s:search>
```

Example 4b: Assigning an account on a resource (specified by name) to a people meeting given criteria (not yet implemented)

```
<s:search xmlns:s="http://midpoint.evolveum.com/xml/ns/public/model/scripting-3"
          xmlns:c="http://midpoint.evolveum.com/xml/ns/public/common/common-3">
    <s:type>c:UserType</s:type>
    <...>
    <s:action>
        <s:type>assign</s:type>
        <s:parameter>
            <s:name>resource</s:name>
            <c:value>Exchange Server</c:value>  <!-- not implemented yet -->
        </s:parameter>
    </s:action>
</s:search>
```

It is possible to specify the target resource by OID (example 4a). In the future it might be possible to specify it also by name (example 4b), or using arbitrary filter.

Example 5: Assigning a role to a people meeting given criteria

```
<s:search xmlns:s="http://midpoint.evolveum.com/xml/ns/public/model/scripting-3"
          xmlns:c="http://midpoint.evolveum.com/xml/ns/public/common/common-3">
    <s:type>c:UserType</s:type>
    <...>
    <s:action>
        <s:type>assign</s:type>
        <s:parameter>
            <s:name>role</s:name>
            <c:value>00000000-0000-0000-0000-000000000008</c:value>
        </s:parameter>
    </s:action>
</s:search>
```

The language and its execution model

Scripting expressions

The basic building block of the language is **a scripting expression**. The expression is a piece of code that may have an input, does some processing, and (optionally) produces an output. Currently there are the following kinds of expressions:

Name	Meaning
search	Retrieves a set of objects via searchObjects model call.
action	An action that can be carried out on a given piece of data that comes at its input. Typical actions are add, modify, delete, enable, disable, assign, resolve, log, ...
select	Selects a given item (container, reference, property) from the input data and copies its value(s) into output. For example, accepts a list of users and selects only their accounts.
filterContent	Removes selected items from the input data. For example, give a list of users, removes all the data except for names and password values. (Since 3.6.)

pipeline	Chains a set of expression together. They are executed one after another; input sent to the pipeline as a whole is sent to the first expression. Output of the last expression is considered to be the output of the whole pipeline.
sequence	Sequence of command expressions. They are executed one after another; input sent to the sequence as a whole is then sent individually to each expression. Output of the last expression is considered to be the output of the whole sequence.

Other planned expression types: are constant expressions, initialization and use of variables, or filtering input values.

Actions

An action modifies the input data (or acts on it in any other way).

In addition to the input data, an action may have one or more parameters. For example, `assign` action must know the role or resource to be assigned; `modify` action must have the delta that has to be applied.

Currently, there are the following actions:

Action	Description	Parameter	Description
add	Adds an object coming as input to the repository, which must be a PrismObject. (***)	-	-
modify	Modifies an object coming as input, which must be a PrismObject. (*) (***)	delta	Delta to be applied to the object.
delete	Deletes an object coming as input, which must be a PrismObject. (*) (***)	-	-
enable, disable	Enables or disables an object coming as input (must be a FocusType or ShadowType). (*) (***)	-	-
assign	Assigns a role or a resource account to a FocusType. (*) (***)	resource	Resource(s) on which account(s) have to be assigned. **
		role	Role(s) to be assigned. **
recompute	Recomputes a user (must be PrismObject<UserType>). (*) (***)	-	-
execute-script	Executes a script against the input data. (Since midPoint 3.4.1)	script	A value of type ScriptExpressionEvaluatorType.
		outputItem	If the script provides any output that is to be processed further, the item definition has to be given here. It is in the form of URI, pointing to item name (e.g. user) or item type (e.g. UserType). "Unqualified" URIs like the two examples here are allowed. But note that outputting data from scripts is currently only experimental.
		forWholeInput	The script would get the whole pipeline as input (since 3.7, experimental).
resume	Resumes a suspended task.	-	The task must be in a suspended state. Since 3.7.2.
resolve	Resolves a reference, e.g. data coming from a c:linkRef, into a PrismObject.	noFetch	Whether noFetch option has to be applied (default: false).
purge-schema	Removes all schema information from a given resource(s) coming as input (PrismObject<ResourceType>).	-	-
discover-connectors	Discovers all connectors on a given connector host(s), given as PrismObject<ConnectorHostType>.	rebindResources	Searches for all resources using now-outdated versions of newly discovered connectors and re-links them to current connectors.
test-resource	Tests a given resource(s) coming as input (PrismObject<ResourceType>).	-	-
validate	Validates a resource - i.e. provides a set of issues just like in Resource Wizard (since 3.5)	-	-
generate-value	Generates value(s) for object(s) coming as input.	items	Description of what and how to generate.
notify	Sends a notification event for each of objects at input (since 3.5) - i.e. it generates a	subtype	Subtype of the event created.

	custom Event with the content driven by action parameters.	handler	Ad-hoc event handler that should be used to process the event. Normally this parameter should not be needed, because event handling should be driven by the system configuration. However, for ad-hoc events we can specify handler directly within the event.
		forWholeInput	Whole input (i.e. all items in the pipeline) should be sent as event object. The default behavior is to generate one event for each input object.
		status	Status to be put into event (success, failure, inProgress, alsoSuccess, onlyFailure). Default is "success".
		operation	Operation to be put into event (add, modify, delete). Default is "add".
log	Logs debugDump form of the data.	level	info (the default), debug or trace
		message	Custom message that is prepended to the data.

(*) In the future these actions will support also PrismReferences instances as their input.

(**) These are to be specified as PrismObjects, PrismReferences, or PrismProperties encapsulating either ObjectReferenceTypes or Strings (understood as OIDs - in the future, string containing resource/role names could be accepted as well). Since 3.7 it is possible to specify queries or search filters here, so it is possible to assign role/resource by its name (see this sample).

(***) Since 3.5, these actions support `dryRun` parameter that (if set to "true") causes executing "preview changes" instead of real modifications. They also (except for recompute) support `raw` parameter for applying the operation in raw mode. And since 3.7 these actions (again except for recompute) support `skipApprovals` parameter, and `options` parameter, as a generalization of these two (`raw`, `skipApprovals`) that can be used to set arbitrary model execution options (see this sample).

Some simple examples of scripts in XML form can be found in **resources/scripting directory** in **model-intest** module and in **tasks/bulk-actions directory** in the **samples** module.

> Since 3.6, `executeScript` action and `notify` action (that contains custom handler) require superuser authorization, because they allow direct execution of user-supplied scripts (groovy, JavaScript, and so on).

Other features

Embedding in tasks

Scripts can be run within tasks. That is extremely useful for long-running scripts. More information is on this page.

Data being passed

The common data format to be passed between expressions, accepted as script input, or provided as script output is the list of prism values (corresponding to objects, containers, references, or properties). For example an output of a `search` command is the list of PrismObjectValues. Or, the output from `search UserType | select linkRef` command is the list of PrismReferenceValues. Each of these values can be accompanied by an `OperationResult` depicting the state of processing that value. So, for example, after selecting 100 users and attempting to disable them, one can easily determine what users were processed correctly and what were not.

Serialization of the data is described here. (TODO)

Console output

As in other scripting languages, midPoint scripting also provides an easily-understandable text output of individual commands. An example:

TODO

Of course, detailed trace of commands executed along with their results is available in the form of OperationResult objects mentioned above. However, the "console output" feature is meant to be a quick and easy way to convey the administrator the result of the script execution. In current implementation, each action puts there information on actions taken (users enabled, disabled, deleted, modified, ...), along with warnings and errors. For any other information, the operation result should be analyzed and displayed.

Error handling

Currently, the policy is "stop on any exception". For example, when a "modify" or "delete" operation throws an ObjectNotFoundException, the script execution simply stops at that moment. This is for safety reasons.

TODO configuration

Note that actions themselves are also a bit picky. When they get an object they cannot act upon (e.g. a PrismPropertyValue in situations where they expect PrismObjectValue, or a ResourceType when they expect UserType), they treat this like a fatal error and stop the execution of the whole script. Also this behavior could be made configurable in the future.

Tools

TODO (GUI, Eclipse plugin, command-line client)