xoRBAC API Reference

This document describes the application programming interface (API) of xoRBAC **version 0.7.0**. For each method we provide an overview, including the method's name, arguments, return value, and a short functional description.

The `RightsManager` class serves as Facade for xoRBAC (for further information on xoRBAC and the concepts behind xoRBAC see, e.g., [SZ08, ZSN07, SN04, NS03, Str04, NS01, xoR]). Therefore, the methods described below are essentially the API methods offered by the `RightsManager` class (the prefix `rm` refers to an instance of the `RightsManager` class).

Logging, Audit, and Caching

`rm startCaching`

- *Arguments:* -
- *Description:* Start caching the access decisions of the `checkAccess` method. However, access decisions involving conditional permissions (permissions that are associated to one or more context constraints) are not cached (of course). Moreover, the access cache is cleared whenever the current policy rule set is modified (e.g. definition of a new role or permission, or changes concerning the assignment relations).
- *Return:* Boolean value

`rm stopCaching`

- *Arguments:* -
- *Description:* Stop caching access decisions and clear cache.
- *Return:* Boolean value

`rm startLogging`

- *Arguments:* -
- *Description:* Start writing the standard xoRBAC status messages into a log file. The names of standard XORBAC log files have the prefix "xoRBAC_".
- *Return:* Boolean value

`rm stopLogging`

- *Arguments:* -
- *Description:* Stop logging standard status messages.
- *Return:* Boolean value

`rm startAudit`

- *Arguments:* -
- *Description:* Start writing audit messages into a log file (auditing is independent from standard logging - auditing is, however, only rudimentary in this version of xoRBAC). The name of the standard XORBAC audit file is "xoRBAC_Audit_LOG".
- *Return:* Boolean value

`rm stopAudit`

- *Arguments:* -
- *Description:* Stop logging audit messages.
- *Return:* Boolean value

Subjects, Roles, and Permissions

`rm createRole name ?juniorroles? ?seniorroles?`

- *Arguments:*
  - **name**: name of the new role (e.g. Lecturer, Student, Professor)
  - **juniorroles**: list of already existing role objects that should be defined as junior-roles for the role identified by the **name** parameter.
  - **juniorroles**: list of already existing role objects that should be defined as senior-roles for the role identified by the **name** parameter.
- *Description:* generate a new role-object `name` with a list of `juniorroles` and a list of `seniorroles` ; on the implementation level `juniorroles` are superclasses of `name` and `seniorRoles` are subclasses of `name`; therefore `name` is more powerful than its `juniorroles`, and the `seniorroles` are more powerful than `name`.
- *Return:* Boolean value

`rm existRole role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor) or a fully-qualified object name.
- *Description:* check if `role` exists.
- *Return:* Boolean value

`rm getRoleList`

- *Arguments:* -
- *Description:* Get a list of all role-objects aggregated by `rm`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm deleteRole role`

- *Arguments:*
  - **role**: name of the role to be deleted (e.g. Lecturer, Student, Professor)
- *Description:* Delete the role-object identified by the `role` parameter.
- *Return:* Boolean value

`rm createSubject subject`

- *Arguments:*
  - **subject**: name of the new subject (e.g. Jane, Smith, jsmith)
- *Description:* generate a new subject-object `subject`.
- *Return:* Boolean value

`rm existSubject subject`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith) or a fully-qualified object name.
- *Description:* check if `subject` exists.
- *Return:* Boolean value

`rm getSubjectList`

- *Arguments:* -
- *Description:* Get a list of all subject-objects aggregated by `rm`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm deleteSubject subject`

- *Arguments:*
  - **subject**: name of the subject to be deleted (e.g. Jane, Smith, jsmith).
- *Description:* Delete the subject-object `subject`.
- *Return:* Boolean value

`rm createPermission operation object`

- *Arguments:*
  - **operation**: the name/id of an "operation" (e.g.“fetch”, “dispatch”, “store”).
  - **object**: the name/id of an "object" (e.g. “record”, “form”, “entry”, “document1”, “printerXY”).
- *Description:* Create a new permission-object granting the right to perform `operation` on `object`.
- *Return:* Boolean value

`rm existPermission perm`

- *Arguments:*
  - **perm**: an "operation object" pair describing the name of the permission (e.g. “fetch record”, “dispatch form”, “store entry”) or a fully-qualified object name.
- *Description:* check if `perm` exists.
- *Return:* Boolean value

`rm getPermList`

- *Arguments:* -
- *Description:* Get a list of all permission-objects aggregated by `rm`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm deletePermission perm`

- *Arguments:*
  - **perm**: name of the permission to be deleted (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* delete the permission-object `perm` and revoke `perm` from all roles and subjects it is directly assigned to (as per-object mixin).
- *Return:* Boolean value

Role and Permission Assignment

`rm addJuniorRoleRelation role junior`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
  - **junior**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* define `junior` as junior-role of `role`.
- *Return:* Boolean value

`rm removeJuniorRoleRelation role junior`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
  - **junior**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* remove the junior-role relation between `junior` and `role`.
- *Return:* Boolean value

`rm addSeniorRoleRelation role senior`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
  - **senior**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* define `senior` as senior-role of `role`.
- *Return:* Boolean value

`rm removeSeniorRoleRelation role senior`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
  - **senior**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* remove the senior-role relation between `senior` and `role`.
- *Return:* Boolean value

`rm roleSubjectAssign role subject`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
- *Description:* assign `role` to `subject`.
- *Return:* Boolean value

`rm roleSubjectRevoke role subject`

- *Arguments:*
  - **role**:name of a role (e.g. Lecturer, Student, Professor)
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
- *Description:* revoke `role` from `subject`.
- *Return:* Boolean value

`rm replaceRoleOwnerTransaction role old new`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
  - **old**: name of the subject to be replaced (e.g. Jane, Smith, jsmith)
  - **new**: name of the new subject `role` should be assigned to (e.g. Sarah, Jones, sjones)
- *Description:* Safely replace an `old` owner of `role` with a `new` one. Applied if the maximum and minimum cardinalities of `role` are equal.
- *Return:* Boolean value

`rm permRoleAssign perm role`

- *Arguments:*
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
  - **role**: name of a role (e.g. Lecturer, Student, Professor).
- *Description:* assign permission `perm` to `role`.
- *Return:* Boolean value

`rm permRoleRevoke perm role`

- *Arguments:*
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
  - **role**: name of a role (e.g. Lecturer, Student, Professor).
- *Description:* revoke permission `perm` from `role`.
- *Return:* Boolean value

`rm replacePermOwnerTransaction perm old new`

- *Arguments:*
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
  - **old**: name of the role to be replaced (e.g. Lecturer, Student, Professor).
  - **new**: name of the new role `perm` should be assigned to (e.g. Lecturer, Student, Professor).
- *Description:* Safely replace an `old` role owning `perm` with a `new` one. Applied if the maximum and minimum cardinalities of `perm` are equal.
- *Return:* Boolean value

Constraints

Cardinality Constraints

`rm setRoleMinSubjectCardinality role cardinality`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor).
  - **cardinality**: a positive integer ≥ 1.
- *Description:* set the minimal number of subjects that `role` must be (directly) assigned to.
- *Return:* Boolean value

`rm getRoleMinSubjectCardinality role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* get the minimal cardinality of `role`, i.e. the number of subjects that `role` must directly be assigned to (-1 means: no limit).
- *Return:* a positive integer if the cardinality is set, -1 otherwise. An empty string if `role` does not exist.

`rm unsetRoleMinSubjectCardinality role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* unset/delete the minimum cardinality defined for `role`.
- *Return:* Boolean value

`rm setRoleMaxSubjectCardinality role cardinality`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
  - **cardinality**: a positive integer ≥ 0 (note that a maximum cardinality of 0 can be sensible for the definition of "private roles").
- *Description:* set maximum cardinality for `role`, i.e. the maximal number of subjects that `role` can be directly assigned to.
- *Return:* Boolean value

`rm getRoleMaxSubjectCardinality role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* get the maximum cardinality defined for `role`, i.e. the maximal number of subjects that `role` could directly be assigned to (-1 means: no limit).
- *Return:* a non-negative integer if the cardinality is set, -1 otherwise. An empty string if `role` does not exist.

`rm unsetRoleMaxSubjectCardinality role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* unset/delete the maximum cardinality defined for `role`.
- *Return:* Boolean value

`rm setPermMinOwnerCardinality perm cardinality`

- *Arguments:*
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
  - **cardinality**: a positive integer ≥ 1.
- *Description:* set the minimal cardinality for the permission `perm`, i.e. the minimal number of roles that `perm` must be directly assigned to.
- *Return:* Boolean value

`rm getPermMinOwnerCardinality perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* get the minimal cardinality defined for `perm`, i.e. the minimal number of roles that `perm` must be directly assigned to (-1 means: no limit).
- *Return:* a positive integer if the cardinality is set, -1 otherwise. An empty string if `perm` does not exist.

`rm unsetPermMinOwnerCardinality perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* unset the minimal cardinality of `perm`.
- *Return:* Boolean value

`rm setPermMaxOwnerCardinality perm cardinality`

- *Arguments:*
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
  - **cardinality**: a non-negative integer ≥ 0 (note that a maximum cardinality of 0 may be sensible to define permissions that can only (directly) be assigned to subjects).
- *Description:* set the maximum cardinality of `perm`, i.e. the maximal number of roles that `perm` could be directly assigned to.
- *Return:* Boolean value

`rm getPermMaxOwnerCardinality perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* get the maximum cardinality defined for `perm`, i.e. the maximal number of roles that `perm` could be directly assigned to (-1 means: no limit).
- *Return:* a non-negative integer if the cardinality is set, -1 otherwise. An empty string if `perm` does not exist.

`rm unsetPermMaxOwnerCardinality perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* unset the maximum cardinality for `perm`.
- *Return:* Boolean value

Static Separation of Duty Constraints

`rm setSSDRoleConstraint role1 role2`

- *Arguments:*
  - **role1**: name of a role (e.g. Lecturer, Student, Professor)
  - **role2**: name of an other role (e.g. Lecturer, Student, Professor)
- *Description:* define `role1` and `role2` as statically mutual exclusive.
- *Return:* Boolean value

`rm getSSDRoleConstraints role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* get a list of all roles that are defined as statically mutual exclusive to `role` (directly or via a role-hierarchy/inheritance).
- *Return:* Empty string, or list of fully-qualified object names.

`rm unsetSSDRoleConstraint role1 role2`

- *Arguments:*
  - **role1**: name of a role (e.g. Lecturer, Student, Professor)
  - **role2**: name of an other role (e.g. Lecturer, Student, Professor)
- *Description:* unset/delete the mutual exclusion constraint between the roles `role1` and `role2`.
- *Return:* Boolean value

`rm setSSDPermConstraint perm1 perm2`

- *Arguments:*
  - **perm1**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
  - **perm2**: an other "operation object" pair representing an other permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* define the permissions `perm1` and `perm2` as statically mutual exclusive.
- *Return:* Boolean value

`rm getSSDPermConstraints perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* get a list of all permissions that are statically mutual exclusive to `perm`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm unsetSSDPermConstraint perm1 perm2`

- *Arguments:*
  - **perm1**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
  - **perm2**: an other "operation object" pair representing an other permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* unset/delete the mutual exclusion constraint between the permissions `perm1` and `perm2`.
- *Return:* Boolean value

Context Constraints

`rm createCondition condition`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
- *Description:* create a new condition-object `condition`.
- *Return:* Boolean value

`rm existCondition condition`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”) or a fully-qualified object name.
- *Description:* check if `condition` exists.
- *Return:* Boolean value

`rm setConditionLeftOperand cndtn snsrtype ctxfunction ?arguments?`

- *Arguments:*
  - **cndtn**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
  - **snsrtype**: name of an xoRBAC sensor.
  - **ctxfunction**: a function/method offered by `snsrtype`.
  - **arguments**: an optional list of parameters passed to the context-function `ctxfunction`.
- *Description:* set the left operand of `condition`.
- *Return:* Boolean value

`rm getConditionLeftOperand condition`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
- *Description:* get the left operand of `condition`
- *Return:* a list of attribute value pairs (`sensortype <value> contextfunction <value> ?arguments <value>?`), or an empty string if the left operand is not yet specified.

`rm setConditionRightOperandAsFunction cnd snsrtype ctxfunction ?arguments?`

- *Arguments:*
  - **cnd**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
  - **snsrtype**: name of an xoRBAC sensor.
  - **ctxfunction**: a function/method offered by `sensortype`.
  - **arguments**: an optional list of parameters passed to the context-function `ctxfunction`.
- *Description:* set the right operand of condition `cnd` as function.
- *Return:* Boolean value

`rm setConditionRightOperandAsConstant condition value`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
  - **value**: a constant value (e.g. a string or an integer value).
- *Description:* set the right operand of `condition` as constant.
- *Return:* Boolean value

`rm getConditionRightOperand condition`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
- *Description:* get the right operand of `condition`
- *Return:* a list of attribute value pairs (`sensortype <value> contextfunction <value> arguments <value> isconstant <0>` if the right operand is defined as function, and `value <value> isconstant <1>` if it is defined as constant), or an empty string if the right operand of `condition` is not yet specified.

`rm setConditionOperator condition operator`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
  - **operator**: one of the following operators: `>, >=, ==, <=, <, !=`.
- *Description:* set the operator for `condition`.
- *Return:* Boolean value

`rm getConditionOperator condition`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
- *Description:* get the operator defined for `condition`.
- *Return:* one of the following operators: `>`, `>=`, `==`, `<=`, `<`, `!=`, or an empty string if the operator is not yet specified.

`rm buildConditionScript condition`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
- *Description*: build the condition script of `condition` (note that the left-operand, operator, and right-operand must be defined prior to building the condition script).
- *Return:* Boolean value

`rm getConditionList`

- *Arguments:* -
- *Description:* get a list of all condition-objects aggregated by `rm`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm deleteCondition condition`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
- *Description:* delete the condition-object `condition` and unlink `condition` from all context constraints.
- *Return:* Boolean value

`rm createContextConstraint constraint`

- *Arguments:*
  - **constraint**: the name of a context constraint (e.g. “myConstraint”, “after11AMandBefore8PM”, “onlyOnHost66.218.71.86andNotAfter20050101”, “constraint14”).
- *Description:* create a new context constraint-object `constraint`.
- *Return:* Boolean value

`rm existContextConstraint constraint`

- *Arguments:*
  - **constraint**: the name of a context constraint (e.g. “myConstraint”, “after11AMandBefore8PM”, “onlyOnHost66.218.71.86andNotAfter20050101”, “constraint14”) or a fully-qualified object name.
- *Description:* check if `constraint` exists.
- *Return:* Boolean value

`rm linkConditionToContextConstraint condition constraint`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
  - **constraint**: the name of a context constraint (e.g. “myConstraint”, “after11AMandBefore8PM”, “onlyOnHost66.218.71.86andNotAfter20050101”, “constraint14”).
- *Description:* link `condition` to context constraint `constraint`.
- *Return:* Boolean value

`rm unlinkConditionFromContextConstraint condition constraint`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “myCondition”, “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
  - **constraint**: the name of a context constraint (e.g. “myConstraint”, “after11AMandBefore8PM”, “onlyOnHost66.218.71.86andNotAfter20050101”, “constraint14”).
- *Description:* unlink `condition` from context constraint `constraint`.
- *Return:* Boolean value

`rm linkContextConstraintToPerm constraint perm`

- *Arguments:*
  - **constraint**: the name of a context constraint (e.g. “myConstraint”, “after11AMandBefore8PM”, “onlyOnHost66.218.71.86andNotAfter20050101”, “constraint14”).
  - **perm**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* link context constraint `constraint` to permission `perm`.
- *Return:* Boolean value

`rm unlinkContextConstraintFromPerm constraint perm`

- *Arguments:*
  - **constraint**: the name of a context constraint (e.g. “myConstraint”,“after11AMandBefore8PM”, “onlyOnHost66.218.71.86andNotAfter20050101”, “constraint14”).
  - **perm**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* unlink context constraint `constraint` from permission `perm`.
- *Return:* Boolean value

`rm getContextConstraintList`

- *Arguments:* -
- *Description:* get a list of all context constraint-objects aggregated by `rm`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm deleteContextConstraint constraint`

- *Arguments:*
  - **constraint**: the name of a context constraint (e.g. “myConstraint”, “after11AMandBefore8PM”, “onlyOnHost66.218.71.86andNotAfter20050101”, “constraint14”).
- *Description:* delete the context constraint-object `constraint` and unlink `constraint` from all permissions.
- *Return:* Boolean value

Review Functions (Introspection)

`rm getAllRolesDirectlyAssignedToSubject subject`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
- *Description:* get a list of all roles that are directly assigned to `subject`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllRolesTransitivelyAssignedToSubject subject`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
- *Description:* get a list of all roles that transitively (via a role-hierarchy/inheritance) assigned to `subject`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllRolesAssignedToSubject subject`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
- *Description:* get a list of all roles that are directly or transitively via a role-hierarchy assigned to `subject`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllSubjectsDirectlyOwningRole role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* get a list of all subjects that own `role` directly.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllSubjectsTransitivelyOwningRole role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* get a list of all subjects that own `role` transitively via a role-hierarchy.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllSubjectsOwningRole role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* get a list of all subjects that own `role` directly or via a role-hierarchy.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllPermsDirectlyAssignedToRole role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* get a list of all permissions that are directly assigned to `role`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllPermsTransitivelyAssignedToRole role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* get a list of all permissions that are transitively via a role-hierarchy/inheritance assigned to `role`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllPermsAssignedToRole role`

- *Arguments:*
  - **role**: name of a role (e.g. Lecturer, Student, Professor)
- *Description:* get a list of all permissions that are directly or via a role-hierarchy/inheritance assigned to `role`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllRolesDirectlyOwningPerm perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* get a list of all roles that own `perm` directly.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllRolesTransitivelyOwningPerm perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* get a list of all roles that own `perm` transitively via a role-hierarchy.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllRolesOwningPerm perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* get a list of all roles that own `perm` directly or via a role-hierarchy.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllPermsDirectlyAssignedToSubject subject`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
- *Description:* get a list of all permissions that are directly assigned to `subject`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllPermsTransitivelyAssignedToSubject subject`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
- *Description:* get a list of all permissions that are transitively (via roles) assigned to `subject`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllPermsAssignedToSubject subject`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
- *Description:* get a list of all permissions that are directly or via roles assigned to `subject`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllSubjectsDirectlyOwningPerm perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* get a list of all subjects that directly own `perm`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllSubjectsTransitivelyOwningPerm perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* get a list of all subjects that transitively (via roles) own `perm`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllSubjectsOwningPerm perm`

- *Arguments:*
  - **perm**: an "operation object" pair representing a permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* get a list of all subjects that - directly or via roles - own `perm`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllPermsLinkedToContextConstraint constraint`

- *Arguments:*
  - **constraint**: the name of a context constraint (e.g. “after11AMandBefore8PM”, “onlyOnHost66.218.71.86andNotAfter20050101”, “constraint14”).
- *Description:* get a list of all permissions linked to context constraint `constraint`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllContextConstraintsLinkedToCondition condition`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
- *Description:* get a list of all context constraints linked to condition `condition`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllConditionsLinkedToContextConstraint constraint`

- *Arguments:*
  - **constraint**: the name of a context constraint (e.g. “after11AMandBefore8PM”, “onlyOnHost66.218.71.86andNotAfter20050101”, “constraint14”).
- *Description:* get a list of all conditions linked to context constraint `constraint`.
- *Return:* Empty string, or list of fully-qualified object names.

`rm getAllPermsTransitivelyLinkedToCondition condition`

- *Arguments:*
  - **condition**: the name of a condition (e.g. “after11AM”, “onlyOnHost66.218.71.86”, “notBefore20050101”).
- *Description:* get all permissions that are transitively (via a context constraint) linked to `condition`.
- *Return:* Empty string, or list of fully-qualified object names.

Access Control Function

`rm checkAccess subject operation object`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
  - **operation**: name of an operation (e.g. fetch, dispatch, debit, read, write)
  - **object**: name of an object (e.g. account, database, file, document1, my-diary)
- *Description:* decide if `subject` (according to its roles/permissions) is allowed to perform the action `operation` on `object`.
- *Return:* Boolean value

Serialization

`rm exportRDF file`

- *Arguments:*
  - **file**: a valid file name.
- *Description:* export/serialize `rm` into `file` including the information about all roles, permissions, subjects, conditions, context constraints, and assignment relations (using an RDF-XML-serialization syntax).
- *Return:* Empty string

`rm importRDF file`

- *Arguments:*
  - **file**: a valid file name.
- *Description:* import an xoRBAC policy rule set consisting of roles, permissions, subjects, conditions, context constraints, and assignment relations from `file` (using the RDF-XML-serialization syntax produced by `exportRDF`).
- *Return:* Boolean value

GUI

`rm startGUI`

- *Arguments:* -
- *Description:* start the xoRBAC Administration GUI (requires Tk).
- *Return:* Empty string

Additional/Experimental Functions

`rm permSubjectAssign perm subject`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* Assign `perm` directly (without an intermediary role) to `subject`.
- *Return:* Boolean value

`rm permSubjectRevoke perm subject`

- *Arguments:*
  - **subject**: name of a subject (e.g. Jane, Smith, jsmith)
  - **perm**: an "operation object" pair representing the permission (e.g. “fetch record”, “dispatch form”, “store entry”).
- *Description:* Revoke the directly assigned permission `perm` from `subject`.
- *Return:* Boolean value

`rm exportXACML file id`

- *Arguments*:
  - **file**: a valid file name.
  - **id**: a (unique) id for the XACML policy rule set.
- *Description*: export the xoRBAC policy rule set in XACML format. The export includes roles, permissions, permission-to-role assignments, and role-to-subject assignments. Note that in version 0.7.0 this method is preliminary/experimental, and although we believe it to be correct, the output is still to be tested for conformance with the XACML 2.0 RBAC Profile as defined in OASIS document: access_control-xacml-2.0-rbac-profile1-spec-os. Moreover, this method does not yet include an export for context constraints because we would need a customized export method for each actual sensor that is used in a context condition. However, if desired, corresponding export functions for context constraints can be added straightforwardly via XACML's <condition> element.
- *Return:* Empty string.

Bibliography

[NS01] G. Neumann and M. Strembeck. Design and Implementation of a Flexible RBAC-Service in an Object-Oriented Scripting Language. In *Proc. of the 8th ACM Conference on Computer and Communications Security (CCS)*, November 2001.

[NS03] G. Neumann and M. Strembeck. An Approach to Engineer and Enforce Context Constraints in an RBAC Environment. In *Proc. of the 8th ACM Symposium on Access Control Models and Technologies (SACMAT)*, June 2003.

[SN04] M. Strembeck and G. Neumann. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments. *ACM Transactions on Information and System Security (TISSEC)*, 7(3), August 2004.

[Str04] M. Strembeck. Conflict Checking of Separation of Duty Constraints in RBAC - Implementation Experiences. In *Proc. of the Conference on Software Engineering (SE 2004)*, February 2004.

[SZ08] M. Strembeck and U. Zdun. Modeling Interdependent Concern Behavior Using Extended Activity Models. *Journal of Object Technology*, 7(6), July/August 2008.

[xoR] xoRBAC Homepage. http://wi.wu-wien.ac.at/home/mark/xoRBAC/.

[ZSN07] U. Zdun, M. Strembeck, and G. Neumann. Object-based and class-based composition of transitive mixins. *Information and Software Technology*, 49(8), August 2007.