iRODS SRM Module Documentation

Ronny Tschüter

November 30, 2010

Contents

1 Introduction

This is the documentation of the SRM module, developed for the iRODS Data Management System. The SRM module enables users to access files on Storage Resource Manager (SRM) servers, transfer them from/to iRODS, and manage files or directories on SRM servers. GSI proxy certificates or every iRODS user are supported, or the iRODS server can be run using a single certificate. The SRM microservice can be used with the `irule` command or it is possible to automate their usage with iRODS actions.

The software is developed within the German DGrid Integrationsprojekt 2 (DGI2) founded by the German Federal Ministry of Education and Research (BMBF).

2 Installation

1. Copy the SRM module to `<irods_dir>/modules`.
2. Ensure in info.txt `Enabled` is set to `yes`.
3. The Globus GSI Libraries must be available on your system.
4. Adjust the include flags and library paths in Makefile, if necessary.
5. Run `irodssetup`.
6. Add Globus Toolkit libraries (`<globus_dir>/lib`) to your environment variable `LD_LIBRARY_PATH`.

3 Usage

The SRM microservices are build on working certificate validation. Therefore latest GSI certificate information, esp. CRLs in `/etc/grid-security`, must be available on iRODS host.

3.1 Available Microservices

msiSrmAbortFiles(*MAN,*SURLS,*TOKEN,*RES)

Aborts an selective file request.

msiSrmAbortRequest(*MAN,*TOKEN)

Aborts an existing request.

msiSrmBringOnline(*MAN,*SURLS,*TIME,*DESC,*PROTO,*RES)

Transfers data online.

msiSrmGetBestSpaceToken(*MAN,*DESC,*SIZE,*RES)

Returns the spacetoken with the biggest sufficient space.

msiSrmGetPermission(*MAN,*SURLS,*RES)

Gets owner and group permissions for specified file.

msiSrmGetSpaceMD(*MAN,*TOKEN,*RES)

Returns all space meta data for the requested space tokens.

msiSrmGetSpaceTokens(*MAN,*DESC,*RES)

Returns all space tokens of the specified token descriptor.

msiSrmLs(*MAN,*SURLS,*LEV,*RES)

Lists all files in specified directories.

msiSrmMkDir(*MAN,*SURL)

Creates a new directory.

msiSrmPing(*MAN,*RES)

Tests the reachability of a network host.

msiSrmPrepareToGet(*MAN,*SURLS,*DESC,*TIME,*PROTO,*RES)

Prepares get data transfer operation.

msiSrmPrepareToPut(*MAN,*SIZES,*SURLS,*DESC,*TIME,*PROTO,*RES)

Prepares 'put' data transfer operation.

msiSrmPutDone(*MAN,*TOKEN,*SURLS,*RES)

Checks whether a prior put data transfer operation has finished.

msiSrmReleaseFiles(*MAN,*SURLS,*TOKEN,*RES)

Release pins on previously requested copies of the SURL.

msiSrmRm(*MAN,*SURLS,*RES)

Deletes specified files.

msiSrmRmDir(*MAN,*SURL,*REC,*RES)

Deletes specified directory.

msiSrmAddGroupPermission(*MAN,*SURLS,*GRP,*PERM,*RES)

Adds new Group permissions.

msiSrmChangeGroupPermission(*MAN,*SURLS,*GRP,*PERM,*RES)

Replaces existing Group permissions with specified ones.

msiSrmRemoveGroupPermission(*MAN,*SURLS,*GRP,*PERM,*RES)

Removes specified Group Permissions from already existing ones.

msiSrmAddOtherPermission(*MAN,*SURLS,*PERM,*RES)

Adds new Other permissions.

msiSrmChangeOtherPermission(*MAN,*SURLS,*PERM,*RES)

Replaces existing Other permissions with specified ones.

msiSrmRemoveOtherPermission(*MAN,*SURLS,*PERM,*RES)

Removes specified Other permissions from already existing ones.

msiSrmAddOwnerPermission(*MAN,*SURL,*PERM,*RES)

Adds new Owner permissions.

msiSrmChangeOwnerPermission(*MAN,*SURL,*PERM,*RES)

Replaces existing Owner permissions with specified ones.

msiSrmRemoveOwnerPermission(*MAN,*SURL,*PERM,*RES)

Removes specified Owner permissions from already existing ones.

msiSrmAddUserPermission(*MAN,*SURL,*UNAME,*PERM,*RES)

Adds new User permissions.

msiSrmChangeUserPermission(*MAN,*SURL,*UNAME,*PERM,*RES)

Replaces existing User permissions with specified ones.

msiSrmRemoveUserPermission(*MAN,*SURL,*UNAME,*PERM,*RES)

Removes specified User permissions from already existing ones.

3.2 Using Proxy Certificates

The SRM microservices make use of user proxy certificates to authenticate the user on SRM servers. Therefore the user needs to upload a valid proxy certificate to iRODS. As an example the following commands load the proxy certificate of a user with UID 1000 to iRODS.

```
$ grid-proxy-init
Your identity: /C=DE/O=GridGermany/OU=Technische Universitaet Dresden/CN=Ronny
    Tschueter
Enter GRID pass phrase for this identity:
Creating proxy ............................. Done
Your proxy is valid until: Tue Nov 30 02:16:27 2010
$ iput -f /tmp/x509up_u1000 user-proxy
$ ils
/tempZone/home/rods:
  user-proxy
```

It is also possible to create proxy certificate using `voms-proxy-init`. The certificate must reside within the iRODS home directory of the user and be named `"user-proxy"`. It can be renamed or a path could be added in the `proxy_file` variable within `srm.h` header file. See Section 3.5.2 for further details. After changes to the header file the SRM module must be recompiled.

The SRM module is also able to work with a single server certificate. Therefore the iRODS server must be started with the userid of a valid certificate.

3.3 Common Usage Scenarios

This section presents a short usage scenario of SRM microservices in iRODS. The following `irule` command uses the `msiSrmLs` microservice to list all files of the specified SRM location.

```
$ irule -v "msiSrmLs(*MAN,*DIR,*LEV,*RES)" "*MAN=httpg://ophelia.zih.tu-dresden.de
    :8443/srm/managerv2%*DIR=srm://ophelia.zih.tu-dresden.de:8443/srm/managerv2?SFN
    =/pnfs/zih.tu-dresden.de/data/dgtest/testfile1 ,srm://ophelia.zih.tu-dresden.de
    :8443/srm/managerv2?SFN=/pnfs/zih.tu-dresden.de/data/dgtest/new_dir%*LEV=1" "*
    RES"
rcExecMyRule: msiSrmLs(*MAN,*DIR,*LEV,*RES)
outParamDesc: *RES
ExecMyRule completed successfully.     Output

*RES:
SURL:            /pnfs/zih.tu-dresden.de/data/dgtest/testfile1
Type:            File
Size:            2563
Permissions:     rw-r--r--
Locality:        Nearline
Checksum Type:   adler32
Checksum:        3a8a6abd
Space Token:     None
Nr of Subpaths: 0

SURL:            /pnfs/zih.tu-dresden.de/data/dgtest/new_dir
Type:            Directory
Size:            0
Permissions:     rwxrwxr-x
Locality:        Lost
Checksum Type:   No information
Checksum:        No information
Space Token:     None
Nr of Subpaths: 1

  SURL:            /pnfs/zih.tu-dresden.de/data/dgtest/new_dir/1
  Type:            Directory
  Size:            0
  Permissions:     rwxrwxr-x
  Locality:        Lost
  Checksum Type:   No information
  Checksum:        No information
  Space Token:     None
  Nr of Subpaths: 0
```

The `msiSrmLs` microservice has four parameters:

- `MAN` - URL of the SRM manager
- `DIR` - comma-separated list of Storage URLs (e.g. `*DIR=surl1,surl2`)
- `LEV` - sub-directories are processed up to the specified level
- `RES` - output parameter

The `MAN` parameter is used by all SRM microservices. Therefore it can be helpful to define an environment variable, which contains SRM manager address information.

```
$ export MY_SRM_MANAGER=httpg://ophelia.zih.tu-dresden.de:8443/srm/managerv2
$ irule -v "msiSrmLs(*MAN,*DIR,*LEV,*RES)" "*MAN=$MY_SRM_MANAGER%*DIR=...%*LEV=..."
    "*RES"
```

The `*PERM` parameter of SRM microservices to add, change or remove permissions supports following arguments: NONE, R, RW, RX, RWX, W, WX or X. The user can specifiy these arguments both in lower or upper case.

`msiSrmPrepareToGet` and `msiSrmPrepareToGet` have a parameter called `*DESC` to specify a SRM space descriptor. If you don't want to specify any space descriptor set `*DESC` to `NULL`.

3.4 Some remarks to interoperabiltiy of SRM and dCache

There some limitations with respect to interoperabiltiy of SRM and dCache. For example, user permissions are currently not supported by dCache and actions to manipulate group permissions

has to use hyphen as group ID. If you work with the SRM door of dCache, set `*GRP` parameter of `msiSrm*GroupPermission` microservices to `NULL` and correct group ID will be set automatically.

3.5 Additional Tips

3.5.1 Proxy certificate validation

In this section it is shown how to use the `grid-proxy-info` tool to validate user proxy certificates. The `grid-proxy-info` tool, which comes with Globus Toolkit, is usually located in `<globus_dir>/bin/`. First, make a link from `grid-proxy-info` to `<irods_dir>/server/bin/cmd`. In the following example both Globus Toolkit and iRODS are installed to the `opt` directory.

```
$ln -s /opt/globus/bin/grid-proxy-info /opt/iRODS/server/bin/cmd
```

As a result users gain a simple way to validate proxy certificates used in iRODS. To verify a proxy certificate the real path on the file-system to this certificate must be determined (e.g, `<irods_dir>/Vault/home/<username>/user-proxy`).

```
$ iexecmd "grid-proxy-info -f /opt/iRODS/Vault/home/rods/user-proxy"
subject  : /C=DE/O=GridGermany/OU=Technische Universitaet Dresden/CN=Ronny
    Tschueter/CN=1429353664
issuer   : /C=DE/O=GridGermany/OU=Technische Universitaet Dresden/CN=Ronny
    Tschueter
identity : /C=DE/O=GridGermany/OU=Technische Universitaet Dresden/CN=Ronny
    Tschueter
type     : RFC 3820 compliant impersonation proxy
strength : 512 bits
path     : /opt/iRODS/Vault/home/rods/user-proxy
timeleft : 11:17:38
```

Unfortunately the user has to specify the iRODS zone prefix (`/opt/iRODS/Vault`) everytime he wants to check a certificate. To permit paths relative to the zone prefix put the following script in `<irods_dir>/server/bin/cmd`.

```
#!/bin/sh
/opt/globus/bin/grid-proxy-info -f /opt/iRODS/Vault/$1
```

With the assistance of this script (named `test-proxy-certificate` in the following example) users can issue `iexecmd` with the irods path to their certificates, but without the zone prefix.

```
$ iexecmd "test-proxy-certificate /home/rods/user-proxy"
```

3.5.2 Possible adjustments in `srm.h`

Changes to `srm.h` take effect only after recompiling the SRM module.

Enable debugging If the user defines `SRM_DEBUG`, additional information about microservice execution will be written to `<irods_dir>/server/logs/rodsLog*`.

Set user proxy file location The variable `char *proxy_file` points to the user proxy file within iRODS user home directory. This file is used for GSI authentication. The default value is `"/user-proxy"`. The user can change this value or add additional paths. However, don't forget the leading slash.

4 Further Information

4.1 License

This software is released under BSD license.

4.2 Contact

You can contact the author and developer via email: `ronny.tschueter@tu-dresden.de`

4.3 Links

IRODS

www.irods.org/

Globus Toolkit

www.globus.org/

dCache

www.d-cache.org/

The Storage Resource Manager, Interface Specification Version 2.2

https://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html

DGI2 homepage

http://dgi.d-grid.de/index.php?id=456&L=1

Project homepage

http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/grid_computing/iRODS

A Files

```
info.txt                        microservice table header for iRODS
Makefile                        module makefile
README                          information about SRM module

micorservice/include/
    smr.h                       module header file
    smrMS.h                     header file with all microservices
    microservices.header        microservice table header for iRODS
    microservices.table         microservice table for iRODS

micorservice/obj/

micorservice/src/
    srm_abortfiles.c            microservice source files
    srm_abortrequest.c
    srm_bringonline.c
    srm_getbestspacetoken.c
    srm_getpermission.c
    srm_getspacemd.c
    srm_getspacetokens.c
    srm_ls.c
    srm_mkdir.c
    srm_ping.c
    srm_preparetoget.c
    srm_preparetoput.c
    srm_putdone.c
    srm_releasefiles.c
    srm_rm.c
    srm_rmdir.c
    srm_setpermission.c
    srm_tools.c                 module helper functions
```

B API Functions

```
/*
 *  iRODS Microservice function that aborts an selective file request.
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surls         msParam_t containing list of host URLs
 *  @param requestToken msParam_t containing token of prior 'put' operation
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmAbortFiles( msParam_t *endpoint, msParam_t *surls, msParam_t *
    requestToken, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that aborts an existing request.
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param requestToken msParam_t containing token of prior 'put' operation
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmAbortRequest( msParam_t *endpoint, msParam_t *requestToken,
    ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that transfers data online.
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surls         msParam_t containing list of host URLs
 *  @param time          msParam_t containing desired pin time
 *  @param desc          msParam_t containing space descriptor
 *  @param protocols     msParam_t containing list of data transfer protocols
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmBringOnline( msParam_t *endpoint, msParam_t *surls, msParam_t *time,
    msParam_t *desc, msParam_t *protocols, msParam_t *result, ruleExecInfo_t *rei )
    ;

/*
 *  iRODS Microservice function that returns the spacetoken with the biggest
     sufficient space.
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param desc          msParam_t containing space descriptor
 *  @param size          msParam_t containing requested storage size
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmGetBestSpaceToken( msParam_t *endpoint, msParam_t *desc, msParam_t *size,
    msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that gets owner and group permissions for specified
      file.
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surls         msParam_t containing list of host URLs
 *  @param result        msParam_t for result output
```

```
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmGetPermission( msParam_t *endpoint, msParam_t *surls, msParam_t *result,
    ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that returns all space meta data for the requested
    space tokens.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param spacetokens  msParam_t containing a list of space tokens
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmGetSpaceMD( msParam_t *endpoint, msParam_t *spacetokens, msParam_t *
    result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that returns all space tokens of the specified
    token descriptor.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param desc         msParam_t containing space descriptor
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmGetSpaceTokens( msParam_t *endpoint, msParam_t *desc, msParam_t *result,
    ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that lists all files in a directory.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param surls        msParam_t containing list of host URLs
 *  @param subdirlevel  msParam_t containing number of listed directory levels
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmLs( msParam_t *endpoint, msParam_t *surls, msParam_t *subdirlevel,
    msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that creates a new directory.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param surl         msParam_t containing host URL
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmMkDir( msParam_t *endpoint, msParam_t *surl, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that tests the reachability of a network host.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
```

```
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmPing(msParam_t *endpoint, msParam_t *result, ruleExecInfo_t *rei);

/*
 *  iRODS Microservice function that prepares 'get' data transfer operation.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param surls        msParam_t containing list of host URLs
 *  @param desc         msParam_t containing space descriptor
 *  @param time         msParam_t containing desired pin time
 *  @param protocols    msParam_t containing list of data transfer protocols
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmPrepareToGet( msParam_t *endpoint, msParam_t *surls, msParam_t *desc,
    msParam_t *time, msParam_t *protocols, msParam_t *result, ruleExecInfo_t *rei )
    ;

/*
 *  iRODS Microservice function that prepares 'put' data transfer operation.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param filesizes    msParam_t lists size of each file to transfer
 *  @param surls        msParam_t containing list of host URLs
 *  @param desc         msParam_t containing space descriptor
 *  @param time         msParam_t containing desired pin time
 *  @param protocols    msParam_t containing list of data transfer protocols
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmPrepareToPut( msParam_t *endpoint, msParam_t *filesizes, msParam_t *surls
    , msParam_t *desc, msParam_t *time, msParam_t *protocols, msParam_t *result,
    ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that checks whether a prior 'put' data transfer
    operation has finished.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param requestToken msParam_t containing token of prior 'put' operation
 *  @param surls        msParam_t containing list of host URLs
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmPutDone( msParam_t *endpoint, msParam_t *requestToken, msParam_t *surls,
    msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that is used to release pins on previously
    requested "copies" of the SURL.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param surls        msParam_t containing list of host URLs
 *  @param requestToken msParam_t containing token of prior 'put' operation
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
    USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
```

```
int msiSrmReleaseFiles( msParam_t *endpoint, msParam_t *surls, msParam_t *
    requestToken, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that deletes files.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param surls        msParam_t containing list of host URLs
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmRm( msParam_t *endpoint, msParam_t *surl, msParam_t *result,
    ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that deletes an directory.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param surls        msParam_t containing list of host URLs
 *  @param recursive    msParam_t specifing whether sub-directories should be
     deleted (1) or not (0)
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmRmDir( msParam_t *endpoint, msParam_t *surls, msParam_t *recursive,
    msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that adds new Group permissions
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param surl         msParam_t containing host URL
 *  @param groupName    msParam_t containing group name
 *  @param permission   msParam_t containing permission (e.g., r, rw, wx, ...)
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmAddGroupPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    groupName, msParam_t *permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that replaces existing Group permissions with
     specified ones
 *
 *  @param endpoint     msParam_t containing SRM manager URL
 *  @param surl         msParam_t containing host URL
 *  @param groupName    msParam_t containing group name
 *  @param permission   msParam_t containing permission (e.g., r, rw, wx, ...)
 *  @param result       msParam_t for result output
 *  @param rei          iRODS ruleExecutionInfo structure
 *  @return             1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmChangeGroupPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    groupName, msParam_t *permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that removes specified Group Permissions from
     already existing ones.
 *
 *  @param endpoint     msParam_t containing SRM manager URL
```

```
 *  @param surl          msParam_t containing host URL
 *  @param groupName     msParam_t containing group name
 *  @param permission    msParam_t containing permission (e.g., r, rw, wx, ...)
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmRemoveGroupPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    groupName, msParam_t *permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that adds new Other permissions
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surl          msParam_t containing host URL
 *  @param permission    msParam_t containing permission (e.g., r, rw, wx, ...)
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmAddOtherPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that replaces existing Other permissions with
     specified ones
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surl          msParam_t containing host URL
 *  @param permission    msParam_t containing permission (e.g., r, rw, wx, ...)
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmChangeOtherPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that removes specified Other permissions from
     already existing ones
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surl          msParam_t containing host URL
 *  @param permission    msParam_t containing permission (e.g., r, rw, wx, ...)
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmRemoveOtherPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that adds new Owner permissions
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surl          msParam_t containing host URL
 *  @param permission    msParam_t containing permission (e.g., r, rw, wx, ...)
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
```

```
int msiSrmAddOwnerPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that replaces existing Owner permissions with
     specified ones
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surl          msParam_t containing host URL
 *  @param permission    msParam_t containing permission (e.g., r, rw, wx, ...)
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmChangeOwnerPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that removes specified Owner permissions from
     already existing ones
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surl          msParam_t containing host URL
 *  @param permission    msParam_t containing permission (e.g., r, rw, wx, ...)
 *  @param result        msParam_t for result output
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmRemoveOwnerPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that adds new User permissions
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surl          msParam_t containing host URL
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmAddUserPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    userName, msParam_t *permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that replaces existing User permissions with
     specified ones
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surl          msParam_t containing host URL
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
 */
int msiSrmChangeUserPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    userName, msParam_t *permission, msParam_t *result, ruleExecInfo_t *rei );

/*
 *  iRODS Microservice function that removes specified User permissions from
     already existing ones
 *
 *  @param endpoint      msParam_t containing SRM manager URL
 *  @param surl          msParam_t containing host URL
 *  @param rei           iRODS ruleExecutionInfo structure
 *  @return              1 on success, otherwise: SYS_INTERNAL_NULL_INPUT_ERR,
     USER_PARAM_TYPE_ERR or EXEC_CMD_ERROR
```

```
 */
int msiSrmRemoveUserPermission( msParam_t *endpoint, msParam_t *surl, msParam_t *
    userName, msParam_t *permissionString, msParam_t *result, ruleExecInfo_t *rei )
    ;
```