

# The Storage Resource Manager Interface Specification

## Version 2.1

This version prepared by:  
Junmin Gu, Alex Sim, Arie Shoshani  
LBNL

THIS IS A WORK IN PROGRESS DRAFT  
It reflects decisions discussed in  
<http://sdm.lbl.gov/srm-wg/doc/SRM.v2.1.joint.func.design.ver0.doc>

### Introduction

This document contains the interface specification of SRM 2.1. It incorporates the functionality of SRM 2.0 (see “srm.methods.v2.0.rev2.doc” posted at <http://sdm.lbl.gov/srm>), but is much expanded to include additional functionality, especially in the area of dynamic storage space reservation and directory functionality in client-acquired storage spaces.

This document reflects the discussions and conclusions of a 2-day meeting whose purpose was to further define the functionality and standardize the interface of Storage Resource Managers (SRMs) – a Grid middleware component. The meeting took place at CERN on December 4-5, 2002. This document is a follow up to the basic SRM design consideration document that describes the basic functionality of SRM Version 2.0 (see “SRM.v2.0.joint.func.design.rev2.doc” posted at <http://sdm.lbl.gov/srm>). The participants at the meeting are listed below.

### Participants:

EDG-WP2: Peter Kunszt, Heinz Stockinger, Kurt Stockinger, Erwin Laure  
EDG-WP5: Jean-Philippe Baud, Stefano Occhetti, Jens Jensen, Emil Knezo, Owen Syngé  
JLAB: Bryan Hess, Andy Kowalski  
FermiLab: Don Petravick, Timur Perelmutov  
LBNL: Arie Shoshani, Alex Sim  
Other contributors not at the meeting: Chip Watson (Jlab), Rich Wellner (FermiLab), Junmin Gu (LBNL)

The document is organized in four sections. The first, called “Defined Structures” contain all the type definitions used to define the functions (or methods). The next 3 sections contain the specification of “Space Management Functions”, “Directory Functions”, and “Data Transfer Functions”. All the “Space Management Functions”, “Directory Functions” are newly added functions, and “Data Transfer Functions” are slightly modified versions of the SRM V2.0 specification.

It is advisable to read the document SRM.v2.1.joint.func.design.doc posted at <http://sdm.lbl.gov/srm> before reading this specification, since the reasoning for the decisions reflected in this specification are described there in detail.

Namespace SRM:

**Notation: underlined attributes are required.**

<i>Defined Structures:</i>
----------------------------

enum	<b>TSpaceType</b>	{ Volatile, Durable, Permanent }
enum	<b>TFileType</b>	{ Volatile, Durable, Permanent }
enum	<b>TPermissionType</b>	{ None, X, W, WX, R, RX, RW, RWX }
enum	<b>TRequestType</b>	{ GET, PUT, COPY }
typedef	string	<b>TRequestToken</b>
typedef	string	<b>TUserID</b>
typedef	string	<b>TOwner</b>
typedef	string	<b>TChecksumType</b>
typedef	unsigned long	<b>TChecksumValue</b>
typedef	unsigned long	<b>TSizeInMB</b>
typedef	unsigned long	<b>TSizeInBytes</b>
typedef	struct { unsigned short <u>year</u> , unsigned short <u>month</u> , // 1-12 unsigned short <u>day</u> , // 1-31 unsigned short <u>hour</u> , // 0-23 unsigned short <u>minute</u> , // 0-59 unsigned short <u>second</u> // 0-59 }	<b>TGMTTime</b>
typedef	unsigned long	<b>TTimeDurationInSeconds</b>
typedef	struct { string string TSizeInBytes TPermissionType TGMTTime TOwner TFileType TSpaceType TTimeDurationInSeconds TTimeDurationInSeconds TChecksumType TChecksumValue TSURL	dir, <u>name</u> , size, yourPermission, createdAtTime, owner, typeOfThisFile, typeOfSpace, durationAssigned, durationLeft, checksumType, checksumValue, originalSURL // if path is a file

} **TMetaDataPathDetail**

```
typedef struct { TSpaceType          typeOfThisSpace,
                TOwner              owner,
                TSizeInMB           totalSizeOfThisSpace,
                TSizeInMB           sizeOfUnusedSpace,
                TSizeInMB           sizeOfUsedSpace,
                TTimeDurationInSeconds durationAssigned,
                TTimeDurationInSeconds durationLeft } TMetaDataSpace

typedef string TStorageSystemInfo

typedef string TSURL // site URL
typedef string TTURL // transfer URL

typedef struct { TSURL              SURLOrStFN,
                TStorageSystemInfo storageSystemIDandAuth } TAccess

typedef struct { TAccess            SURLInfo
                TAccess            stFNInfo
                string             globalFileName
                TTimeDurationInSeconds lifetime // pin time
                TFileType          fileType
                TSizeInBytes       knownSizeOfThisFile,
                TSizeInMB          maxFileLength } TGetFileRequest

typedef struct { TAccess            stFNInfo
                string             globalFileName
                TTimeDurationInSeconds lifetime // pin time
                TFileType          fileType
                TSizeInBytes       knownSizeOfThisFile,
                TSizeInMB          maxFileLength } TPutFileRequest

typedef struct { TAccess            fromSURLOrStFNInfo
                TAccess            toSURLOrStFNInfo
                string             globalFileName
                TTimeDurationInSeconds lifetime // pin time
                TFileType          fileType
                TSizeInBytes       knownSizeOfThisFile,
                TSizeInMB          maxFileLength } TCopyFileRequest

typedef struct { unsigned int statusCode,
                string        explanation } TReturnStatus
```

// convention of the statusCode: format: 5 digits: x-xx-xx, where x are 0-9:

```
// first digit: 1= SRM common, for other SRM specific codes, use 2 to 9.
// the next two digits are function specific
// the last two digits are reserved for status code
// SRM common codes are defined at the end of this document.
// for example, srmReleaseSpace() has its return codes 10201, 10202, 10203, 10204
```

```
typedef struct {string path,
                 TReturnStatus status } TPathReturnStatus
```

```
typedef struct {TSURL surl,
                 TReturnStatus status } TSURLReturnStatus
```

```
typedef struct {TSURL siteURL,
                 TsizeInBytes fileSize,
                 TReturnStatus status,
                 TTimeDurationInSeconds estimatedWaitingTimeOnQueue,
                 TTimeDurationInSeconds estimatedProcessingTime,
                 TTURL transferURLFromSRM,
                 TTimeDurationInSeconds remainingPinTimeIfAny
                } TFileRequestStatus
```

```
typedef struct {TRequestToken requestToken,
                 TRequestType requestType,
                 int totalFilesInThisRequest,
                 int numOfQueuedRequests,
                 int numOfFinishedRequests,
                 int numOfProgressingRequests,
                 Boolean isSuspended} TRequestSummary
```

```
typedef struct {TSURL surl,
                 TReturnStatus status,
                 TPermissionType userPermission
                } TCheckPermissionReturnStatus
```

```
typedef struct {TRequestToken requestToken,
                 TGMTTime createdAtTime
                } TGetRequestIDReturnStatus
```

**notes:**

- *UserID is not needed when we use gsi.*
- *StorageSystemInfo is a string that contains the login and password required by the storage system. For example, it might have the form of login:pwd@hostname, where “:” is a reserved separator between login and pwd. If hostname is not provided, it is defaulted to what’s in the accompanying site URL or the host of SRM.*

- *TMetaDataSpace* can refer to a single space of each type (i.e. volatile, durable, permanent). It does not include the extra space needed to hold the directory structures.
- Regarding files in Volatile space: Any file in Volatile space is owned by the SRM, but the requester(s) have read permission to it. If another user requests this file, he needs to provide a source siteURL so SRM can check from the source site whether the user has a read/write permission. If permission is granted, then the SRM updates its permission list to include this caller and returns the file in Volatile space instead getting the file from the source site.
- *GlobalFileName* is not a required attribute.
- The type definition *SURL* above is used for both site URL and the “Storage File Name” (*stFN*). This was done in order to simplify the notation. Recall that *stFN* is the file path/name of the intended storage location when a file is put (or copied) into an SRM controlled space. Thus, a *stFN* can be thought of a special case of an *SURL*, where the protocol is assumed to be “srm” and the machine:port is assumed to be local to the SRM. For example, when the request *srmCopy* is made, the source file is specified by a site URL, and the target location can be optionally specified as a *stFN*. By considering the *stFN* a special case of an *SURL*, an *srmCopy* takes *SURLs* as both the source and target parameters.
- The requestToken assigned by SRM is unique and immutable (non-reusable). For example, if the date:time is part of the requestToken it will be immutable.

<b><i>Function specification:</i></b>
---------------------------------------

### Space Management Functions:

**summary:**

**srmReserveSpace**  
**srmReleaseSpace**  
**srmUpdateSpace(includes size and time)**  
**srmCompactSpace:**

**srmGetSpaceMetaData:**

**srmChangeFileType:**

**details:**

**srmReserveSpace:**

In:	TUserID	userID,
	TSpaceType	<u>typeOfSpaceToReserve,</u>
	TSizeInMB	sizeOfSpaceToReserve,
	TTimeDurationInSeconds	lifetimeOfSpaceToReserve,
	TStorageSystemInfo	storageSystemInfo

Out:	TSpaceType	typeOfReservedSpace,
	TSizeInMB	sizeOfReservedSpace,
	TTimeDurationInSeconds	lifetimeOfReservedSpace,
	TReturnStatus	<u>returnStatus</u>

**notes:**

- *lifetimeOfSpaceToReserve is not needed if requesting permanent space.*
- *SRM can provide default size and duration if not supplied.*
- *storageSystemInfo is optional in case storage system requires additional security check.*

**srmReleaseSpace:**

In:	TUserID	userID,
	TSpaceType	<u>typeOfSpace,</u>
	Boolean	forceFileRelease

Out:	TReturnStatus	<u>returnStatus</u>
------	---------------	---------------------

**notes:**

- *forceFileRelease=false is default. This means that the space will not be released if it has files that are still pinned in the space. To release the space regardless of the files it contains and their status forceFileRelease=true must be specified.*
- *To be safe, a request to release a reserved space that has an on-going file transfer will return false, even forceFileRelease= true.*
- *When space is releasable and forceFileRelease=true, all the files in the space are released, even in durable or permanent space.*
- *It is up to each SRM whether a released space will result in removing all its files/directories immediately. One possibility is to keep files/directories in volatile space when the Durable or Permanent spaces are released.*

**srmUpdateSpace(includes size and time)**

In:	TUserID	userID,
	TSpaceType	<u>designatedSpaceType,</u>
	TSizeInMB	newSize,
	TTimeDurationInSeconds	newDurationFromCallingTime

Out:	TSizeInMB	actualSizeGranted,
	TTimeDurationInSeconds	actualDurationGranted,
	TReturnStatus	<u>returnStatus</u>

**notes:**

- *If neither size or duration are supplied in the input, then return will be null.*
- *newSize is the new actual size of the space, so has to be positive.*

- *newDurationFromCallingTime is the new lifetime requested regardless of the previous lifetime, and has to be positive. It might even be shorter than the remaining lifetime at the time of the call.*

**srmCompactSpace:**

In:     TUserID                    userID,  
           TSpaceType                typeOfSpace,  
           Boolean                    doDynamicCompactFromNowOn

Out:    TSizeInMB                newSizeOfThisSpace

**notes:**

- *This function is called to reclaim the space for all released files and update space size in Durable and Permanent spaces. Files not released are not going to be removed (even if lifetime expired.) Directory structure will stay intact.*
- *doDynamicCompactFromNowOn=false by default, which implies that only a one time compactSpace will take place.*
- *If doDynamicCompactFromNowOn=true, then the space of released files will be automatically compacted until the value of doDynamicCompactFromNowOn is set to false.*
- *When space is compacted, the files in that space do not have to be removed by the SRM. For example, the SRM can choose to move them to volatile space. The client will only perceive that the compacted space is now available to them.*
- *To physically force a removal of a file, the client should use srmRm.*

**srmGetSpaceMetaData:**

In:     TUserID                    userID,  
           TSpaceType[]            arrayOfTypeOfSpace

Out:    TMetaDataSpace[]        arrayOfSpaceDetails

**notes:**

- *If no typeOfSpace is given, return ALL caller spaces under each of the types.*

**srmChangeFileType:**

In:     TUserID                    userID,  
           string[]                 arrayOfPath/filename,  
           string[]                 arrayOfGlobalFileName,  
           TSpaceType               typeOfSpace,  
           TFileType                 desiredType

Out:    TPathReturnStatus[]    returnStatus

**notes:**

- *Either path or globalFileName must be supplied.*

- *If a path is pointing to a directory, then the effect is recursive for all the files in this directory.*
- *Changing the file type is bound to the restriction of filetypes in spacetypes, e.g. a Volatile file can not be changed to Permanent if it is not in a Permanent space.*

## Directory Functions:

---

### **summary:**

**srmMkdir:**  
**srmRmdir:** (applies to *dir*)  
**srmRm:** (applies to *file*)  
**srmLs:** (applies to both *dir* and *file*)  
**srmMv:** (applies to both *dir* and *file*)  
**srmCp:** (applies to both *dir* and *file*)  
**srmCd:**  
**srmPwd:**  
**srmReassignToUser:**  
**srmAddPermission:**  
**srmRmPermission:**

### **details:**

#### **srmMkdir:**

In:	TUserID TSpaceType string string	userID, <u>designedSpaceType</u> , topDirectory, <u>newDirectoryPath</u> ,
Out:	TReturnStatus	<u>returnStatus</u>

#### **notes:**

- *The topDirectory refers to the user's top directory. If omitted, the user's top directory is assumed.*
- *Consistent with unix, recursive creation of directories is not supported.*
- *newDiretoryPath can include paths, as long as all sub directories exist.*

#### **srmRmdir: (applies to *dir* )**

In:	TUserID string TSpaceType boolean	userID, <u>dirToBeDeleted</u> , <u>spaceType</u> , doRecursiveRemove
Out:	TReturnStatus	<u>returnStatus</u>

**notes:**

- *doRecursiveRemove* is false by default.
- To distinguish from *srmRm()*, this function is for directories only.
- We use “~” to refer to the top directory of this user in that space.

**srmRm: (applies to files )**

In: TUserID                      userID,  
      string[]                    arrayOfFilePathsToBeDeleted,  
      TSpaceType                spaceType

Out: TPathReturnStatus[] arrayOfDeletedSuccessfully

**notes:**

- To distinguish from *srmRmDir()*, this function applies to files only.

**srmLs: (applies to both *dir* and *file*)**

In: TUserID                      userID,  
      String[]                    pathToBeListed,  
      TSpaceType                spaceType,  
      boolean                    fullDetailedList,  
      boolean                    allLevelRecursive

Out: TMetaDataPathDetail[] details

**notes:**

- *fullDetailedList=false* by default.
- If *fullDetailedList=true* provide full details similar to unix “ls -l”.
- If *allLevelRecursive=true* then file lists of all level below current will be provided as well.

**srmMv: (applies to both *dir* and *file*)**

In: TUserID                      userID,  
      string                      pathToBeMovedFrom,  
      string                      pathToBeMovedTo,  
      TSpaceType                spaceTypeOfFromPath,  
      TSpaceType                spaceTypeOfToPath

Out: TReturnStatus            returnStatus

**notes:**

- Space allocation and de-allocation may be involved if moving from one type of space to another.
- Both paths here are assumed to be owned by the same user.

**srmCp: (applies to both *dir* and *file*)**

In:    TUserID            toUserID,  
       TAccess            toStFNInfo,  
       TAccess            fromStFNInfo,  
       Boolean            copyRecursively // default = false

Out:   TReturnStatus      returnStatus

**notes:**

- *The toUserID must be the ID of the user making the srmCp call.*
- *Space allocation may be involved at the destination side.*
- *Permission checking is required if different users are involved.*

**srmCd:**

In:    TUserID            userID,  
       string              pathToBeChangedTo

Out:   TReturnStatus      returnStatus

**notes:**

- PathToBeChangedTo is assumed to be relative to current directory.

**srmPwd:**

In:    TUserID            userID

Out:   String             currentPath

**srmAddPermission: (applies to both *dir* and *file*)**

In:    TUserID            userID,  
       string              pathTargeted,  
       TSpaceType          spaceTypeOfFromPath,  
       TPermissionType    newPermission,  
       String              anotherUser

Out:   TReturnStatus      returnStatus

**notes:**

- *If anotherUser = “\*”, it means world permission.*
- *AnotherUser depends on the security model of the SRM. For example, If gsi is used, the “distinguished name” may be used.*

**srmRmPermission: (applies to both *dir* and *file*)**

In:    TUserID            userID,  
       string              pathTargeted,  
       TSpaceType          spaceTypeOfFromPath,  
       TPermissionType    permissionToBeRemoved,  
       String              anotherUser

Out: TReturnStatus            returnStatus

**notes:**

- *If anotherUser = “\*”, it means world permission.*
- *AnotherUser depends on the security model of the SRM. For example, If gsi is used, the “distinguished name” may be used.*

**srmReassignToUser:**

In:     TUserID                    userID,  
       string                        assignedUser,  
       TTimeDurationInSeconds    lifeTimeOfThisAssignment,  
       String                        designatedPathFromOwner // file or dir,  
       TSpaceType                    designatedSpaceTypeFromOwner

Out: TReturnStatus            returnStatus

**notes:**

- *This function implies actual lifetime of file/space involved is extended up to the lifeTimeOfThisAssignment.*
- *The caller must be the owner of the files to be reassigned.*
- *permission is omitted because it has to be READ permission.*
- *lifeTimeOfThisAssignment is relative to the calling time. So it must be positive.*
- *After lifeTimeOfThisAssignment time period, or when assignedUser obtained a copy of files through srmCp(), the files involved are released and space is compacted automatically, which ever is first.*
- *If the path here is a directory, then all the files under it are included recursively.*
- *If there are any files involved that are released before this function call, then these files will not be involved in reassignment, even if they are still in the space.*
- *If a compact() is called before this function is complete, then this function has priority over compact(). Compact will be done automatically as soon as files are copies to the assignedUser. Whether to dynamically compact or not is an implementation choice.*

**Data Transfer Functions:**

---

**summary:**

**srmPrepareToGet:**

**srmPrepareToPut:**

**srmCopy:**

**srmReleaseFiles:** (dir is ok. Will release recursively for dirs)

**srmRemoveFiles:**

**srmPutDone:**

**srmAbortRequest:**

**srmAbortFiles:**  
**srmSuspendRequest:**  
**srmResumeRequest:**

**srmGetRequestStatus:**  
**srmGetFilesStatus:**  
**srmGetRequestSummary:**

**srmExtendFileLifeTime:**  
**srmGetRequestID:**

**srmCheckPermission:**

**details:**

**srmPrepareToGet:**

In: TUserID                    userID,  
TGetFileRequest[]         arrayOfFileRequest,  
string[]                    arrayOfTransferProtocols,  
string                      callbackReference,  
string                      userRequestDescription,  
TspaceType                 designatedSpace,  
TTimeDurationInSeconds   retryTime

Out: TRequestToken         requestToken,  
TFileRequestStatus[]     arrayOfFileStatus

**notes:**

- *The userRequestDescription is a user designated name for the request. It can be used in the getRequestID method to get back the system assigned request ID.*
- *If callbackReference is provided then callback will be performed.*
- *Only pull mode is supported.*
- *SRM rejects the file request if stFN (in the TGetFileRequest) is not local.*
- *If stFN is not specified, SRM will generate a name automatically and put it in the specified user space. This will be returned as part of the “transfer URL”.*
- *SRM assigns the requestToken at this time.*
- *Normally this call will be followed by srmRelease().*
- *“retryTime” means: if all the file transfer for this request are complete, then try previously failed transfers for a total time period of “retryTime”.*
- *In case that the retries fail, the return should include an explanation of why the retries failed and when the tries took place.*

**srmPrepareToPut:**

In: TUserID                    userID,  
TPutFileRequest[]         arrayOfFileRequest,  
string[]                    arrayOfTransferProtocols,  
string                      callbackReference,

string userRequestDescription,  
TSpaceType designatedSpace,  
TTimeDurationInSeconds retryTime

Out: TRequestToken requestToken,  
TFileRequestStatus[] arrayOfFileStatus

**notes:**

- *If callbackReference is provided then callback will be performed.*
- *Only push mode is supported for srmPrepareToPut.*
- *StFN (in the TPutFileRequest) has to be local. If stFN is not specified, SRM will name it automatically and put it in the specified user space. This will be returned as part of the “transfer URL”.*
- *srmPutDone() is expected after each file is “put” into the allocated space.*
- *The lifetime of the file starts as soon as SRM get the srmPutDone(). If srmPutDone() is not provided then the files in that space are subject to removal when the space lifetime expires.*
- *“retryTime” is meaningful here only when the file destination is not a local disk, such as tape or MSS.*
- *In case that the retries fail, the return should include an explanation of why the retries failed and when the tries took place.*

**srmCopy:**

In: TUserID userID,  
TCopyFileRequest[] arrayOfFileRequest,  
string callbackReference,  
string userRequestDescription,  
TSpaceType designatedSpace,  
Boolean removeSourceFiles (default = false),  
TTimeDurationInSeconds retryTime

Out: TRequestToken requestToken,  
TFileRequestStatus[] arrayOfFileStatus

**notes:**

- *If callbackReference is provided then callback will be performed.*
- *Pull mode: copy from remote location to SRM. (e.g. from remote to MSS.)*
- *Push mode: copy from SRM to remote location.*
- *Always release files from source after copy is done.*
- *When removeSourceFiles=true, then SRM will remove the source files on behalf of the caller after copy is done.*
- *In pull mode, send srmRelease() to remote location when transfer is done.*
- *If in push mode, then after transfer is done, notify the caller. User can then release the file. If user releases a file being copied to another location before it is done, then refuse to release.*
- *Note there is no protocol negotiation for this request.*

- “*retryTime*” means: if all the file transfer for this request are complete, then try previously failed transfers for a total time period of “*retryTime*”.
- In case that the retries fail, the return should include an explanation of why the retries failed and when the tries took place.

**srmRemoveFiles:**

In: TRequestToken      requestToken,  
      TUserID                userID,  
      TSURL[]                siteURLs

Out: TSURLReturnStatus[]      arrayOfReturnStatus

**notes:**

- If *requestToken* is not provided, then the SRM will do nothing.
- It has the effect of a release before the file is removed.
- If file is not in cache, do nothing

**srmReleaseFiles:**

In: TRequestToken      requestToken,  
      TUserID                userID,  
      TSURL[]                siteURLs

Out: TSURLReturnStatus[]      arrayOfReturnStatus

**notes:**

- If *requestToken* is not provided, then the SRM will release all the files specified by the *siteURLs* owned by this user, regardless of the *requestToken*.
- If *requestToken* is not provided, then *userID* is needed. It may be inferred or provide in the call.
- Releasing files will be followed by compacting space, if *doDynamicCompactFromNowOn* was set to true in a previous *srmCompactSpace* call.

**srmPutDone:**

In: TRequestToken      requestToken,  
      TUserID                userID,  
      TSURL[]                arrayOfSiteURL

Out: TSURLReturnStatus[]      arrayOfReturnStatus

**notes:**

- Called by user after *srmPut()*

**srmAbortRequest:**

In: TRequestToken requestToken,  
TUserID userID

Out: TReturnRequest returnStatus

**notes:**

- *Terminate all file requests in this request regardless of the state. Expired files are released.*

**srmAbortFiles**

In: TRequestToken requestToken,  
TSURL[] arrayOfSiteURLs,  
TUserID userID

Out: TSURLReturnStatus[] arrayOfReturnStatus

**notes://**

**srmSuspendRequest:**

In: TRequestToken requestToken,  
TUserID userID

Out: TReturnStatus returnStatus

**notes://**

**srmResumeRequest:**

In: TRequestToken requestToken,  
TUserID userID

Out: TReturnStatus returnStatus

**notes://**

**srmGetRequestStatus:**

In: TRequestToken requestToken,  
TUserID userID

Out: TFileRequestStatus[] arrayOfFileStatus

**notes:**

- *Returns status for all the file requests in this request.*

**srmGetFilesStatus:**

In: TRequestToken requestToken,  
TSURLOrStFN[] arrayOfSURLOrStFNs,  
TUserID userID

Out: TFileRequestStatus[] arrayOfFileStatus

**notes:**

- *For put requests, the target stFNs are checked, otherwise, source SURLS are checked.*

**srmGetRequestSummary:**

In: TRequestToken[] arrayOfRequestToken,  
TUserID userID

Out: TRequestSummary[] arrayOfRequestSummary

**srmExtendFileLifeTime:**

In: TRequestToken requestToken,  
TSURL siteURL,  
TUserID userID,  
TTimeDurationInSeconds newLifeTimeRequestedFromCallingTime

Out: TReturnStatus returnStatus  
TTimeDurationInSeconds newTimeExtended

**notes:**

- *newLifeTime is relative to the calling time. Lifetime will be set from the calling time for the specified period.*
- *The number of lifetime extensions maybe limited by SRM according to its policies.*
- *IsExtended = false if SRM refuse to do it. (set newTimeExtended = 0 in this case.)*
- *If original lifetime is longer than the requested one, then the requested one will be assigned.*
- *If newLifeTime is not specified, the SRM can use its default to assign the newLifeTime.*

**srmGetRequestID:**

In: string userRequestDescription  
TUserID userID

Out: TGetRequestIDReturnStatus[] arrayOfPossibleRequestToken

**notes:**

- *If userRequestDescription is null, returns all requests this user has.*
- *If the user assigned the same name to multiple requests, he may get back multiple request IDs each with the time the request was made.*

**srmCheckPermission:**

In: TAccess[] arrayOfSiteURL  
TUserID userID,  
Boolean checkInLocalCacheOnly // default: false

Out: TCheckPermissionReturnStatus[] arrayOfResults

**notes:**

- When *checkInLocalCacheOnly=true*, then SRM will only check files in its local cache. Otherwise, if a file is not in its local cache, then SRM will go to the *siteURL* to check the user permission.
- Note that *spaceType* is not specified here. It is up to SRM to decide whether to respond with one or more of Volatile/Durable/Permanent spaces when local cache is concerned.
- If *checkInLocalCacheOnly = false*, SRM can choose to always check the *siteURL* for user permission of each file. It is also ok if SRM choose to check its local cache first, if a file exists and the user has permission, return that permission. Otherwise, check the *siteURL* and return permission.

<b>StatusCode specification:</b>
----------------------------------

**Note:** More status codes will be added as we collect other useful codes that should be in common to all SRMs. For example, we may want to provide more codes for the reasons that the space reservation failed, such as UNAUTHORIZED\_USER, UNPRIVILEGED\_USER, NO\_MORE\_SPACE, etc..., or it may be sufficient to have such reasons given in the “explanation” string as part of **TReturnStatus**.

Function code	Function Name	code
01.	<i>srmReserveSpace</i>	
	#define SPACE_RESERVED	10101
	#define SPACE_PARTIALLY_RESERVED	10102
	#define SPACE_RESERVE_FAILED	10103
02.	<i>srmReleaseSpace</i>	
	#define SPACE_RELEASED	10201
	#define SPACE_ALREADY_RELEASED	10202
	#define SPACE_DOES_NOT_EXIST	10203
	#define SPACE_NOT_RELEASED	10204
03.	<i>srmUpdateSpace</i>	
	#define SPACE_UPDATED	10301
	#define SPACE_PARTIALLY_UPDATED	10302
	#define SPACE_DOES_NOT_EXIST	10303
	#define SPACE_UPDATE_FAILED	10304
04.	<i>srmCompactSpace:</i>	
	// none	
05.	<i>srmGetSpaceMetaData:</i>	
	// none	
06.	<i>srmChangeFileType:</i>	
	#define FILETYPE_CHANGED	10601
	#define FILETYPE_NOT_CHANGED	10602

07.	<i>srmMkdir:</i>	
	#define MKDIR_SUCC	10701
	#define NO_PERMISSION	10702
	#define MKDIR_FAILED	10703
08.	<i>srmRmdir:</i>	
	#define RMDIR_SUCC	10801
	#define NO_PERMISSION	10802
	#define DIR_DOES_NOT_EXIST	10803
	#define RMDIR_FAILED	10804
09.	<i>srmRm:</i>	
	#define FILE_DELETED	10901
	#define FILE_DOES_NOT_EXIST	10902
	#define NO_PERMISSION	10903
	#define FILE_DELETE_FAILED	10904
10.	<i>srmLs:</i>	
	// none	
11.	<i>srmMv:</i>	
	#define MV_SUCC	11101
	#define NO_PERMISSION	11102
	#define PATH_DOES_NOT_EXIST	11103
	#define MV_FAILED	11104
12.	<i>srmCp:</i>	
	#define CP_SUCC	11201
	#define PATH_DOES_NOT_EXIST	11202
	#define NO_PERMISSION	11203
	#define NOT_ENOUGH_SPACE	11204
	#define CP_FAILED	11205
13.	<i>srmCd:</i>	
	#define CD_SUCC	11301
	#define CD_FAILED	11302
14.	<i>srmPwd:</i>	
	// none	
15.	<i>srmReassignToUser:</i>	
	#define REQUEST_ACCEPTED	11501
	#define NO_PERMISSION	11502
	#define USER_DOES_NOT_EXIST	11503
	#define PATH_DOES_NOT_EXIST	11504
	#define REQUEST_FAILED	11505
16.	<i>srmAddPermission:</i>	
	#define ADD_PERMISSION_OK	11601
	#define PATH_DOES_NOT_EXIST	11602
	#define PERMISSION_EXISTS	11603
	#define ADD_PERMISSION_FAILED	11604
17.	<i>srmRmPermission:</i>	
	#define RM_PERMISSION_OK	11701
	#define PATH_DOES_NOT_EXIST	11702

	#define	PERMISSION_DOESNOT_EXIST	11703
	#define	RM_PERMISSION_FAILED	11704
18.		<i>srmPrepareToGet:</i>	
	#define	GET_REQUEST_QUEUED	11801
	#define	GET_REQUEST_PROCESSED	11802
	#define	GET_REQUEST_SUSPENDED	11803
	#define	GET_REQUEST_ABORTED	11804
	#define	GET_REQUEST_DONE	11805
	#define	GET_REQUEST_RELEASED	11806
	#define	GET_REQUEST_PINNED	11807
	#define	GET_REQUEST_PIN_EXPIRED	11808
	#define	GET_REQUEST_FAILED	11809
19.		<i>srmPrepareToPut:</i>	
	#define	PUT_REQUEST_QUEUED	11901
	#define	PUT_REQUEST_PROCESSED	11902
	#define	PUT_REQUEST_SUSPENDED	11903
	#define	PUT_REQUEST_ABORTED	11904
	#define	PUT_REQUEST_DONE	11905
	#define	PUT_REQUEST_RELEASED	11906
	#define	SPACE_ALLOCATED	11907
	#define	PUT_REQUEST_PINNED	11908
	#define	GET_REQUEST_PIN_EXPIRED	11909
	#define	PUT_REQUEST_FAILED	11910
20.		<i>srmCopy:</i>	
	#define	COPY_REQUEST_QUEUED	12001
	#define	COPY_REQUEST_PROCESSED	12002
	#define	COPY_REQUEST_SUSPENDED	12003
	#define	COPY_REQUEST_ABORTED	12004
	#define	COPY_REQUEST_DONE	12005
	#define	COPY_REQUEST_RELEASED	12006
	#define	COPY_REQUEST_FAILED	12007
21.		<i>srmReleaseFiles:</i>	
	#define	FILE_RELEASED	12101
	#define	FILE_DOES_NOT_EXIST	12102
	#define	INVALID_REQUESTTOKEN	12203
	#define	FILE_RELEASE_FAILED	12204
22.		<i>srmPutDone:</i>	
	#define	PUTDONE_OK	12201
	#define	INVALID_REQUESTTOKEN	12202
	#define	FILE_DOES_NOT_EXIST	12203
	#define	PUTDONE_FAILED	12204
23.		<i>srmAbortRequest:</i>	
	#define	ABORTED_REQUEST	12301
	#define	INVALID_REQUESTTOKEN	12302
	#define	REQUEST_ALREADY_DONE	12303
	#define	ABORT_REQUEST_FAILED	12304

```

24.          srmAbortFiles:
      #define          ABORTED_FILE          12401
      #define          INVALID_REQUESTOKEN  12402
      #define          FILE_DOES_NOT_EXIST  12403
      #define          FILE_ALREADY_DONE    12404
      #define          ABORT_FILE_FAILED    12405
25.          srmSuspendRequest:
      #define          SUSPENDED            12501
      #define          ALREADY_SUSPENDED    12502
      #define          INVALID_REQUEST_TOKEN 12503
      #define          REQUEST_ALREADY_FINISHED 12504
      #define          SUSPEND_FAILED       12505
26.          srmResumeRequest:
      #define          RESUMED              12601
      #define          ALREADY_RESUMED      12602
      #define          INVALID_REQUEST_TOKEN 12603
      #define          REQUEST_ALREADY_FINISHED 12604
      #define          RESUME_FAILED        12605
27.          srmGetRequestStatus
      // none
28.          srmGetFilesStatus:
      // none
29.          srmGetRequestSummary:
      // none
30.          srmExtendFileLifeTime:
      #define          EXTENDED            13001
      #define          INVALID_REQUESTOKEN  13002
      #define          FILE_DOES_NOT_EXIST  13003
      #define          LIMIT_REACHED        13004
      #define          EXTEND_FAILED        13005
31.          srmGetRequestID:
      // none
32.          srmCheckPermission:
      #define          FILE_DOES_NOT_EXIST  13201
      #define          FILE_EXISTS_LOCALLY   13202
      #define          FILE_EXISTS_AT_SOURCE 13203
33.          srmRemoveFiles:
      #define          FILE_DOES_NOT_EXIST  13301
      #define          FILE_REMOVED         13302
      #define          NO_PERMISSION        13303
      #define          FILE_REMOVE_FAILED   13304

```