

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/documents/joint.docs/SRM.v2.1.joint.func.design.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 syng

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	TSpaceType	{Volatile, Durable, Permanent}
enum	TFileType	{Volatile, Durable, Permanent}
enum	TPermissionType	{R,W,X}
enum	TRequestType	{GET, PUT, COPY}
enum	TStatusCodeGet	{Queued, Processing, Error, Released, Suspended, Aborted, Pinned, WaitToBePinned}
enum	TStatusCodePut	{Queued, Processing, Error, Suspended, Aborted, SpaceAllocated, PutDone, Pinned, WaitToBePinned, Released}
enum	TStatusCodeCopy	{Queued, Processing, Error Suspended, Aborted, CopyDone, Released}
typedef	string	TRequestToken
typedef	string	TReason
typedef	string	TUserID
typedef	unsigned long	TSizeInMB
typedef	struct {TPermissionType owner, TPermissionType group, TPermissionType world}	TPermission
typedef	struct { <u>int year</u> , <u>int month</u> , // 1-12 <u>int day</u> , // 1-31 <u>int hour</u> , // 0-23 <u>int minute</u> , // 0-59 <u>int second</u> // 0-59}	TGMTTime
typedef	struct {int day, int hour, int minute}	TTimeDuration

```

typedef      struct { boolean      isDir,
                string      name,
                TSizeInMB    size,
                TPermissionType yourPermission,
                TGMTTime     createdAtTime,
                string      owner,
                string      fromURL // if path is a file
        } TMetaDataPathDetail

typedef      struct { TSpaceType    typeOfThisSpace,
                string      owner,
                TSizeInMB    totalSizeOfThisSpace,
                TSizeInMB    sizeOfUnusedSpace,
                TSizeInMB    sizeOfUsedSpace,
                TTimeDuration durationAssigned,
                TTimeDuration durationLeft } TMetaDataSpace

typedef      struct { TFileType     typeOfThisFile,
                TSpaceType    typeOfSpace,
                string      owner,
                TSizeInMB    sizeOfThisFile,
                TTimeDuration durationAssigned,
                TTimeDuration durationLeft } TMetaDataFile

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

typedef      struct { TStorageSystemID storageSystemID,
                TStorageSystemAuth encryptedAuthInfo } TStorageSystemInfo

typedef      struct { TStorageSystemInfo storageSystemIDandAuth,
                TSURL      SURLOrStFN } TAccess

typedef      struct { TAccess      SURLInfo
                TAccess      stFNInfo
                string      globalFileName
                TTimeDuration lifetime // pin time
                TFileType     fileType
                TSizeInMB    knownSizeOfThisFile,
                TSizeInMB    maxFileLength } TFileRequest

```

```

typedef      struct {TAccess          fromSURLInfo
              TAccess          toSURLInfo
              string          globalFileName
              TTimeDuration    lifetime // pin time
              TFileType        fileType
              TSizeInMB        knownSizeOfThisFile,
              TSizeInMB        maxFileLength} TCopyFileRequest

typedef      struct {TStatusCodeGet   getStatus,
              TStatusCodePut         putStatus,
              TStatusCodeCopy        copyStatus} TStatusCode

typedef      struct {TStatusCode   status,
              string              explanation} TStatus

typedef      struct {TURL           siteURL,
              TStatus              status,
              TTimeDuration        estimatedWaitingTimeOnQueue,
              TTimeDuration        estimatedProcessingTime,
              TTURL                transferURLFromSRM
              TTimeDuration        remainingPinTimeIfAny} TFileStatus

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

```

notes:

- *UserID is not needed when we use gsi.*
- *StorageSystemID 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.*
- *StorageSystemAuth is an encrypted string that is required by the storage system.*
- *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.

<i>Function specification:</i>

Space Management Functions:

summary:

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

srmGetFilesMetaData:
srmGetSpaceMetaData:

srmChangeFileType:

details:

srmReserveSpace:

In:	TUserID TSpaceType TSizeInMB TTimeDuration TStorageSystemInfo	userID, <u>typeOfSpaceToReserve,</u> sizeOfSpaceToReserve, lifetimeOfSpaceToReserve, storageSystemInfo
Out:	TSpaceType TSizeInMB TTimeDuration	typeOfReservedSpace, sizeOfReservedSpace, lifetimeOfReservedSpace,

TReason	possibleExplanation,
Boolean	isSpaceReserved

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.*
- *If isSpaceReserved=false, it means SRM refuses the request, and all the other parameters should be null, except possibleExplanation.*

srmReleaseSpace:

In:	TUserID	userID,
	TSpaceType	<u>typeOfSpace,</u>
	Boolean	forceFileRelease
Out:	Boolean	releaseIsSuccessful,
	TReason	possibleExplanation

notes:

- *A request to release a non-reserved space (e.g. non-exist, or already released space) will return true.*
- *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,
	TTimeDuration	newDurationFromCallingTime
Out:	TSizeInMB	actualSizeGranted,
	TTimeDuration	actualDurationGranted,
	TReason	possibleExplanation

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.*

srmGetFilesMetaData:

In: TUserID userID,
 String[] arrayOfPath,
 TSpaceType spaceType,
 String[] arrayOfGlobalFileName

Out: TMetaDataFile[] fileDetails

notes:

- *The path can be specified as ~user/relative_path where ~user can be omitted if caller is referring to self. Like unix, one user needs access permission granted by the owner of the file/space to look into another user's directory.*
- *If the path is null, then return fileDetails on all the files in the space.*
- *spaceType is needed to determine which space to look into, because the file path is relative to each space type.*

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	<u>desiredType</u>

Out: Boolean changeIsSuccessful

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:

details:

srmMkdir:

In:	TUserID	userID,
	TSpaceType	<u>designedSpaceType</u> ,
	string	currentDirectory,
	string	<u>newDirectoryPath</u> ,

Out: Boolean dirCreatedSuccessfully

notes:

- *The topDirectory can be omitted if referring to the user's top directory.*
- *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	userID,
	string	<u>dirToBeDeleted,</u>
	TspaceType	<u>spaceType,</u>
	boolean	doRecursiveRemove

Out:	Boolean	<u>pathDeletedSuccessfully</u>
------	---------	--------------------------------

notes:

- *doRecursiveRemove is false by default.*
- *To distinguish from srmRm(), this function is for directories only.*

srmRm: (applies to files)

In:	TUserID	userID,
	string[]	<u>arrayOfFilePathsToBeDeleted,</u>
	TspaceType	<u>spaceType</u>

Out:	Boolean[]	<u>arrayOfDeletedSuccessfully</u>
------	-----------	-----------------------------------

notes:

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

srmLs: (applies to both dir and file)

In:	TUserID	userID,
	string	<u>pathToBeListed,</u>
	TspaceType	<u>spaceType,</u>
	boolean	fullDetailedList,
	boolean	oneLevelRecursive

Out:	TMetaDataPathDetail[]	details
------	-----------------------	---------

notes:

- *doFullDetailedList=false by default.*
- *If doFullDetailedList=true provide full details similar to unix "ls -l".*
- *If oneLevelRecursive=true then file lists of one level below current will be provided as well.*

srmMv: (applies to both dir and file)

In:	TUserID	userID,
	string	<u>pathToBeMovedFrom,</u>
	string	<u>pathToBeMovedTo,</u>
	TspaceType	<u>spaceTypeOfFromPath,</u>
	TspaceType	<u>spaceTypeOfToPath</u>

Out: Boolean moveIsSuccessful

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,
string pathToBeCopiedTo,
TSpaceType spaceTypeOfToPath,
TUserID fromUserID,
string pathToBeCopiedFrom,
TSpaceType spaceTypeOfFromPath,
Boolean copyRecursively // default = false

Out: Boolean copyIsSuccessful

notes:

- *The toUserID must be the ID of the user making the srmCp call.*
- *The fromUserID can be the ID of either the user making the srmCp call or another user.*
- *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: Boolean cdIsSuccessful

srmPwd:

In: TUserID userID

Out: String currentPath

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

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

Out: Boolean addPermissionIsSuccessful

notes:

- *If anotherUser = "world", 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	<u>assignedUser,</u>
	TTimeDuration	<u>lifeTimeOfThisAssignment,</u>
	String	designatedPathFromOwner // file or dir,
	TSpaceType	designatedSpaceTypeFromOwner
Out:	Boolean	acknowledged

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.*
- *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)

srmPutDone:

srmAbortRequest:

srmAbortFiles:

srmSuspendRequest:

srmResumeRequest:

srmGetRequestStatus:

srmGetFilesStatus:

srmGetRequestSummary:

srmExtendFileLifeTime:
srmGetRequestID:

srmCheckInLocalCache:

details:

srmPrepareToGet:

In:	TUserID	userID,
	TFileRequest[]	<u>arrayOfFileReuquest,</u>
	string[]	arrayOfProtocols,
	string	callbackReference,
	string	userRequestDescription,
	TSpaceType	designatedSpace

Out:	TRequestToken	<u>requestToken,</u>
	TFileRequestStatus[]	arrayOfFileStatus

notes:

- *If callbackReference is provided then callback will be performed.*
- *Only pull mode is supported.*
- *SRM rejects the file request if stFN (in the TFileRequest) 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().*

srmPrepareToPut:

In:	TUserID	userID,
	TFileRequest[]	<u>arrayOfFileRequest,</u>
	string[]	arrayOfProtocols,
	string	callbackReference,
	string	userRequestDescription,
	TSpaceType	designatedSpace

Out:	TRequestToken	<u>requestToken,</u>
	TFileRequestStatus[]	arrayOfFileStatus

notes:

- *If callbackReference is provided then callback will be performed.*
- *Only push mode is supported for srmPrepareToPut.*
- *StFN (in the TfileRequest) 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.*

srmCopy:

In:	TUserID	userID,
	TCopyFileRequest[]	<u>arrayOfFileReuquest</u> ,
	string	callbackReference,
	string	userRequestDescription,
	TSpaceType	designatedSpace,
	Boolean	releaseSourceFiles (default = false)
Out:	TRequestToken	<u>requestToken</u> ,
	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.*
- *When releaseSourceFiles=true, then SRM will release 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.*

srmReleaseFiles:

In:	TRequestToken	requestToken,
	TUserID	userID,
	TSURL[]	<u>siteURLs</u>
Out:	Boolean[]	arrayOfReleaseStatus

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	<u>requestToken</u> ,
	TSURL[]	<u>arrayOfSiteURL</u>

Out: //none

notes:

- *Called by user after srmPut()*

srmAbortRequest:

In: TRequestToken requestToken

Out: Boolean terminated

notes:

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

srmAbortFiles

In: TRequestToken requestToken,
TSURL[] arrayOfSiteURLs

Out: Boolean[] terminated
TReason possibleExplanation

notes:

- *If no siteURLs are given, return terminated=false.*
- *If siteURL does not exist, return terminated=false.*
- *PossibleExplanation should be used for the reason of terminated=false.*

srmSuspendRequest:

In: TRequestToken requestToken

Out: Boolean suspended

notes:

- *Return false if request was completed.*

srmResumeRequest:

In: TRequestToken requestToken

Out: Boolean resumed

notes:

- *Return false if request was completed.*

srmGetRequestStatus:

In: TRequestToken requestToken

Out: TFileRequestStatus[] arrayOfFileStatus

notes:

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

srmGetFilesStatus:

In: TRequestToken requestToken,
TSURLOrStFN[] arrayOfSURLOrStFNs

Out: TFileRequestStatus[] arrayOfFileStatus

notes:

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

srmGetRequestSummary:

In: TRequestToken[] arrayOfRequestToken

Out: TRequestSummary[] arrayOfRequestSummary

srmExtendFileLifeTime:

In: TRequestToken requestToken,
TSURL siteURL,
TTimeDuration newLifeTimeRequestedFromCallingTime

Out: Boolean isExtended.
TTimeDuration 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

Out: TRequestToken[] arrayOfPossibleRequestToken

notes:

- *If userRequestDescription is null, returns null.*

srmCheckInLocalCache:

In: TSURL[] arrayOfSiteURL

Out: Boolean[] arrayOfSiteURLsInCache

notes:

- *spaceType* is not specified here. It is up to SRM to decide whether to respond with one or more of Volatile/Durable/Permanent spaces.
- We assume caller has permission for the files in question.