GWD-R-XXXX

Grid Storage Resource Management
https://forge.gridforum.org/projects/gsm-wg

Editors:
A. Sim
A. Shoshani

9/5/2007

The Storage Resource Manager Interface Specification
Version 2.2

Status of this Memo
This document provides information to the Grid community regarding the specification of the Storage Resource Management. Distribution of this document is unlimited.

Abstract
Storage management is one of the most important enabling technologies for large-scale scientific investigations. Having to deal with multiple heterogeneous storage and file systems is one of the major bottlenecks in managing, replicating, and accessing files in distributed environments. Storage Resource Managers (SRMs), named after their web services protocol, provide the technology needed to manage the rapidly growing distributed data volumes, as a result of faster and larger computational facilities. SRMs are Grid storage services providing interfaces to storage resources, as well as advanced functionality such as dynamic space allocation and file management on shared storage systems. They call on transport services to bring files into their space transparently and provide effective sharing of files. SRMs are based on a common specification that emerged over time and evolved into an international collaboration. This approach of an open specification that can be used by various institutions to adapt to their own storage systems has proven to be a remarkable success – the challenge has been to provide a consistent homogeneous interface to the Grid, while allowing sites to have diverse infrastructures. In particular, one of the main goals to the SRM web service is to support optional features while preserving interoperability.

Table of Contents

Introduction

This document contains the interface specification of SRM 2.2. It incorporates the functionality of SRM 2.0 and SRM 2.1, 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 in May 2006 at Fermilab, which as followed by a 3-day meeting in September 2006 at CERN, as well as email correspondence and conference calls. The purpose of this activity is to agree on the functionality and standardize the interface of Storage Resource Managers (SRMs) – a Grid middleware component.

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 5 sections contain the specification of "Space Management Functions", "Permission Functions", "Directory Functions", "Data Transfer Functions" and "Discovery Functions". All the "Discovery Functions" are newly added functions.

The Appendix describes the main concepts of SRMs as a standard middleware specification for various storage systems. It is intended to support the same interface to simple files systems, as well as sophisticated storage system that include multiple disk caches, robotic tape systems, and parallel file systems. The appendix also lists several implementations of SRM v2.2 around the world, and their deployment in various sites.

For people not familiar with SRM concepts, It is advisable to read the Appendix first. For people familiar with previous versions of SRM specifications, it is advisable to read the document SRM.v2.2.changes.doc posted at http://sdm.lbl.gov/srm-wg before reading this specification.

Meaning of terms
By "https" we mean http protocol with GSI authentication. It may be represented as "httpg". At this time, any implementation of http with GSI authentication could be used. It is advisable that the implementation is compatible with Globus Toolkit 3.2 or later versions.

- Primitive types used below are consistent with XML build-in schema types**:** i.e.
  - *long* is 64bit: **(+/-)** 9223372036854775807
  - *int* is 32 bit**: (+/-)** 2147483647
  - *short* is 16 bit**: (+/-)** 32767
  - *unsignedLong* ranges (inclusive): 0 to18446744073709551615
  - *unsignedInt* ranges (inclusive): 0 to 4294967295
  - *unsignedShort* ranges (inclusive): 0 to 65535

- The definition of the type "anyURI" used below is compliant with the XML standard. See http://www.w3.org/TR/xmlschema-2/#anyURI. It is defined as: "The lexical space of anyURI is finite-length character sequences which, when the algorithm defined in Section 5.4 of [XML Linking Language] is applied to them, result in strings which are legal URIs according to [RFC 2396], as amended by [RFC 2732]".

- In "localSURL", we mean local to the SRM that is processing the request.

- authorizationID : from the SASL RFC 2222
  During the authentication protocol exchange, the mechanism performs authentication, transmits an authorization identity (frequently known as a userid) from the client to server…. The transmitted authorization identity may be different than the identity in the client's authentication credentials. This permits agents such as proxy servers to authenticate using their own credentials, yet request the access privileges of the identity for which they are proxying. With any mechanism, transmitting an authorization

identity of the empty string directs the server to derive an authorization identity from the client's authentication credentials.

- Regarding file sharing by the SRM, it is a local implementation decision. An SRM can choose to share files by proving multiple users access to the same physical file, or by copying a file into another user's space. Either way, if an SRM chooses to share a file (that is, to avoid reading a file over again from the source site) the SRM should check with the source site whether the user has a read/write permission. Only if permission is granted, the file can be shared.

- The word "pinning" is limited to the "copies" or "states" of SURLs and the Transfer URLs (TURLs).

- For each function, status codes are defined with basic meanings for the function. Only those status codes are valid for the function. Specific cases are not stated for each status code. If other status codes need to be defined for a specific function, send an email to the collaboration to discuss the usage.

1. Common Type Definitions

Namespace SRM

Notation: underlined attributes are ***REQUIRED*.**

1.1. File Storage Type

enum **TFileStorageType** {VOLATILE, DURABLE, PERMANENT}

- Volatile file has an expiration time and the storage may delete all traces of the file when it expires.
- Permanent file has no expiration time.
- Durable file has an expiration time, but the storage may not delete the file, and should raise error condition instead.

1.2. File Type

enum **TFileType** {FILE, DIRECTORY, LINK}

1.3. Retention Policy

enum **TRetentionPolicy** { REPLICA , OUTPUT , CUSTODIAL }

- Quality of Retention (Storage Class) is a kind of Quality of Service. It refers to the probability that the storage system lose a file. Numeric probabilities are self-assigned.
  - Replica quality has the highest probability of loss, but is appropriate for data that can be replaced because other copies can be accessed in a timely fashion.
  - Output quality is an intermediate level and refers to the data which can be replaced by lengthy or effort-full processes.
  - Custodial quality provides low probability of loss.
- The type will be used to describe retention policy assigned to the files in the storage system, at the moments when the files are written into the desired destination in the storage system. It will be used as a property of space allocated through the space reservation function. Once the retention policy is assigned to a space, the files put in the reserved space will automatically be assigned the retention policy of the space. The assigned retention policy on the file can be found thought the TMetaDataPathDetail structure returned by the srmLs function.

1.4. Access Latency

enum **TAccessLatency** { ONLINE, NEARLINE }

- Files may be Online, Nearline or Offline. These terms are used to describe how latency to access a file is improvable. Latency is improved by storage systems replicating a file such that its access latency is online.
  - The ONLINE cache of a storage system is the part of the storage system which provides file with online latencies.
  - ONLINE has the lowest latency possible. No further latency improvements are applied to online files.
  - NEARLINE file can have their latency improved to online latency automatically by staging the file to online cache.

- For completeness, we also describe OFFLINE here.
- OFFLINE files need a human to be involved to achieve online latency.
- For the SRM we only keep ONLINE and NEARLINE.

o The type will be used to describe a space property that access latency can be requested at the time of space reservation. The content of the space, files may have the same or "lesser" access latency as the space.

1.5. Permission Mode

enum **TPermissionMode** {NONE, X, W, WX, R, RX, RW, RWX}

1.6. Permission Type

enum **TPermissionType** {ADD, REMOVE, CHANGE}

1.7. Request Type

enum **TRequestType** { PREPARE_TO_GET,
PREPARE_TO_PUT,
COPY,
BRING_ONLINE,
RESERVE_SPACE,
UPDATE_SPACE,
CHANGE_SPACE_FOR_FILES,
LS }

1.8. Overwrite Mode

enum **TOverwriteMode** {NEVER,
ALWAYS,
WHEN_FILES_ARE_DIFFERENT}

o Use case for WHEN_FILES_ARE_DIFFERENT can be that files are different when the declared size for an SURL is different from the actual one, or that the checksum of an SURL is different from the actual one.

o Overwrite mode on a file is considered higher priority than pinning a file. Where applicable, it allows to mark a valid Transfer URL to become invalid when the owner of the SURL issues an overwrite request.

1.9. File Locality

enum **TFileLocality** { ONLINE,
NEARLINE,
ONLINE_AND_NEARLINE,
LOST,
NONE.
UNAVAILABLE }

o Files may be located online, nearline or both. This indicates if the file is online or not, or if the file reached to nearline or not. It also indicates if there are online and nearline copies of the file.

- The ONLINE indicates that there is a file on online cache of a storage system which is the part of the storage system, and the file may be accessed with online latencies.

- The NEARLINE indicates that the file is located on nearline storage system, and the file may be accessed with nearline latencies.
- The ONLINE_AND_NEARLINE indicates that the file is located on online cache of a storage system as well as on nearline storage system.
- The LOST indicates when the file is lost because of the permanent hardware failure.
- The NONE value shall be used if the file is empty (zero size).
- The UNAVAILABLE indicates that the file is unavailable due to the temporary hardware failure.

o The type will be used to describe a file property that indicates the current location or status in the storage system.

1.10. Access Pattern

enum **TAccessPattern** { TRANSFER_MODE, PROCESSING_MODE }

o TAccessPattern will be passed as an input parameter to the srmPrepareToGet and srmBringOnline functions. It will make a hint from the client to SRM how the Transfer URL (TURL) produced by SRM is going to be used. If the parameter value is “ProcessingMode”, the system may expect that client application will perform some processing of the partially read data, followed by more partial reads and a frequent use of the protocol specific “seek” operation. This will allow optimizations by allocating files on disks with small buffer sizes. If the value is “TransferMode” the file will be read at the highest speed allowed by the connection between the server and a client.

1.11. Connection Type

enum **TConnectionType** { WAN, LAN }

o TConnectionType indicates if the client is connected though a local or wide area network. SRM may optimize the access parameters to achieve maximum throughput for the connection type. This will be passed as an input to the srmPrepareToGet, srmPrepareToPut and srmBringOnline functions.

1.12. Status Codes

enum **TStatusCode** {
SRM_SUCCESS,
SRM_FAILURE,
SRM_AUTHENTICATION_FAILURE,
SRM_AUTHORIZATION_FAILURE,
SRM_INVALID_REQUEST,
SRM_INVALID_PATH,
SRM_FILE_LIFETIME_EXPIRED,
SRM_SPACE_LIFETIME_EXPIRED,
SRM_EXCEED_ALLOCATION,
SRM_NO_USER_SPACE,
SRM_NO_FREE_SPACE,
SRM_DUPLICATION_ERROR,
SRM_NON_EMPTY_DIRECTORY,
SRM_TOO_MANY_RESULTS,
SRM_INTERNAL_ERROR,

SRM_FATAL_INTERNAL_ERROR,
SRM_NOT_SUPPORTED,
SRM_REQUEST_QUEUED,
SRM_REQUEST_INPROGRESS,
SRM_REQUEST_SUSPENDED,
SRM_ABORTED,
SRM_RELEASED,
SRM_FILE_PINNED,
SRM_FILE_IN_CACHE,
SRM_SPACE_AVAILABLE,
SRM_LOWER_SPACE_GRANTED,
SRM_DONE,
SRM_PARTIAL_SUCCESS,
SRM_REQUEST_TIMED_OUT,
SRM_LAST_COPY,
SRM_FILE_BUSY,
SRM_FILE_LOST,
SRM_FILE_UNAVAILABLE,
SRM_CUSTOM_STATUS
}

- SRM_NOT_SUPPORTED is used, in general
  - If a server does not support a method
  - If a server does not support particular optional input parameters

1.13. Retention Policy Info

typedef struct { TRetentionPolicy retentionPolicy,
TAccessLatency accessLatency
} **TRetentionPolicyInfo**

- TRetentionPolicyInfo is a combined structure to indicate how the file needs to be stored.
- When both retention policy and access latency are provided, their combination needs to match what SRM supports. Otherwise request will be rejected.

1.14. Request Token

- The Request Token assigned by SRM is unique and immutable (non-reusable). For example, if the date:time is part of the request token it will be immutable.
- Request tokens are case-sensitive.
- Request token is valid until the request is completed. However, SRM server may choose to keep the request tokens for a short period of time after the request is completed, and the time period depends on the SRM servers.

1.15. User Permission

typedef struct { string userID,
TPermissionMode mode
} **TUserPermission**

- userID may represent the associated client's Distinguished Name (DN) instead of unix style login name. VOMS role may be included.

1.16. Group Permission

typedef struct { string groupID,
TPermissionMode mode
} **TGroupPermission**

- groupID may represent the associated client's Distinguished Name (DN) instead of unix style login name. VOMS role may be included.

1.17. Size in Bytes

- Size in bytes is represented in unsigned long.

1.18. UTC Time

- Time is represented in dateTime.
- Formerly TGMTTime in SRM v2.1
- date and time in Coordinated Universal Time (UTC, formerly GMT) with no local time extention.
- Format is same as in XML dateTime type, except no local time extension is allowed. E.g. 1999-05-31T13:20:00 is ok (for 1999 May 31st, 13:20PM, UTC) but 1999-05-31T13:20:00-5:00 is not.

1.19. Time in Seconds (Lifetime and RequestTime)

- Time (lifetime and request time) in seconds is represented in integer.
- "0" (zero) indicates the site defined default time.
- A negative value (-1) indicates "infinite (indefinite)" time.
- Exceptions:
  - Any "remaining" times may have zero (0) second when no time is left.
  - Some special meaning of negative time is defined when needed depending on the operation. E.g. remainingTotalRequestTime in srmStatusOfGetRequest

1.20. SURL

- The type definition SURL is represented as anyURI and 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 as a pulling case, 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, a srmCopy takes SURLs as both the source and target parameters.

1.21. TURL

- TURL is represented in anyURI.

1.22. Return Status

```
typedef     struct {TStatusCode     statusCode,
                    string          explanation
                } TReturnStatus
```

1.23. Return Status for SURL

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

1.24. File MetaData

```
typedef     struct {string                  path,   // absolute dir and file path
                    TReturnStatus           status,
                    unsigned long           size,    // 0 if directory
                    dateTime                createdAtTime,
                    dateTime                lastModificationTime,
                    TFileStorageType        fileStorageType,
                    TRetentionPolicyInfo    retentionPolicyInfo,
                    TFileLocality           fileLocality,
                    string[]                arrayOfSpaceTokens,
                    TFileType               type,     // Directory or File
                    int                     lifetimeAssigned,
                    int                     lifetimeLeft, // on the SURL
                    TUserPermission         ownerPermission,
                    TGroupPermission        groupPermission,
                    TPermissionMode         otherPermission,
                    string                  checkSumType,
                    string                  checkSumValue,
                    TMetaDataPathDetail[]   arrayOfSubPaths
                                                    // optional recursive
                } TMetaDataPathDetail
```

- The *TMetaDataPathDetail* describes the properties of a file. It is used as an output parameter in *srmLs*.
- *retentionPolicyInfo* indicates the assigned retention policy.
- *fileLocality* indicates where the file is located currently in the system.
- *arrayOfSpaceTokens* as an array of *string* indicates where the file is currently located for the client. Only space tokens that the client has authorized to access to read the file must be returned.
- Permissions on the SURL represent unix-like permissions: e.g. rwxr--r--.
- *ownerPermission* describes the owner ID and owner permission on the SURL.
- *groupPermission* describes the group permission with group identifier on the SURL.

- *otherPermission* describes the other permission on the SURL.
- For ACL-like permissions, srmGetPermission must be used.
- *lifetimeAssigned* is the total lifetime that is assigned on the SURL. It includes all SURL lifetime extensions if extended.
- *lifetimeLeft* is the remaining lifetime on the SURL from the current time until expiration.
  - A negative value (-1) indicates "indefinite" lifetime.
  - Zero (0) indicates that the file is expired.

1.25. Space MetaData

```
typedef     struct { string                 spaceToken,
                     TReturnStatus          status,
                     TRetentionPolicyInfo   retentionPolicyInfo,
                     string                 owner,
                     unsigned long          totalSize,        // best effort
                     unsigned long          guaranteedSize,
                     unsigned long          unusedSize,
                     int                    lifetimeAssigned,
                     int                    lifetimeLeft
                   } TMetaDataSpace
```

- *TMetaDataSpace* is used to describe properties of a space, and is used as an output parameter in *srmGetSpaceMetaData*.
- *retentionPolicyInfo* indicates the information about retention policy and access latency that the space is assigned. *retentionPolicyInfo* is requested and assigned at the time of space reservation through *srmReserveSpace* and *srmStatusOfReserveSpaceRequest*.
- *TMetaDataSpace* refers to a single space with retention policy. It does not include the extra space needed to hold the directory structures, if there is any.
- *lifetimeAssigned* is the total lifetime that is assigned to the space. It includes all space lifetime extensions if extended.
- *lifetimeLeft* is the remaining lifetime that is left on the space.

1.26. Directory Option

```
typedef     struct { Boolean    isSourceADirectory,
                     Boolean    allLevelRecursive,    // default = false
                     int        numOfLevels           // default = 1
                   } TDirOption
```

1.27. Extra Info

```
typedef     struct { string     key,
                     string     value
                   } TExtraInfo
```

- TExtraInfo is used where additional information is needed, such as for additional information for transfer protocols of TURLs in srmPing, srmGetTransferProtocols, srmStatusOfGetRequest, and srmStatusOfPutRequest. For example, when it is used for additional information for transfer protocols, the keys may specify access speed, available number of parallelism, and other transfer protocol properties.

- It is also used where additional information to the underlying storage system is needed, such as for additional information, but not limited to, for storage device, storage login ID, storage login authorization. Formerly, it was TStorageSystemInfo.

1.28. Transfer Parameters

```
typedef     struct { TAccessPattern          accessPattern,
                     TConnectionType         connectionType,
                     string[]                arrayOfClientNetworks
                     string[]                arrayOfTransferProtocols
                   } TTransferParameters
```

- TTransferParameters is used where arrayOfTransferProtocols was used previously in SRM v2.1.
- TTransferParameters may be provided optionally in the methods such as srmPrepareToGet, srmBringOnline, srmPrepareToPut and srmReserveSpace. Optional input parameters in TTransferParameters may collide with the characteristics of the space specified. In this case, TTransferParameters as an input parameter must be ignored.
- File transfer protocols are specified in a preferred order on all SRM transfer functions.
- arrayOfClientNetworks is a hint of the client IPs that SRM/dCache can use for optimization of its internal storage systems based on the client's accessible IP addresses.

1.29. File Request for srmPrepareToGet

```
typedef     struct { anyURI              sourceSURL,
                     TDirOption          dirOption,
                   } TGetFileRequest
```

1.30. File Request for srmPrepareToPut

```
typedef     struct { anyURI              targetSURL ,     // local to SRM
                     unsigned long       expectedFileSize
                   } TPutFileRequest
```

1.31. File Request for srmCopy

```
typedef     struct { anyURI              sourceSURL,
                     anyURI              targetSURL,
                     TDirOption          dirOption
                   } TCopyFileRequest
```

1.32. Return File Status for srmPrepareToGet

```
typedef     struct { anyURI              sourceSURL,
                     TReturnStatus       status,
                     unsigned long       fileSize,
                     int                 estimatedWaitTime,
                     int                 remainingPinTime,
                     anyURI              transferURL,
                     TExtraInfo[]              transferProtocolInfo
                   } TGetRequestFileStatus
```

- *transferProtocolInfo* of type *TExtraInfo* is added to the *TGetRequestFileStatus*. This output parameter can be used to provide more information about the transfer protocol so that client can access the TURL efficiently.
- *estimatedWaitTime* to be negative value, -1, for unknown.
- *remainingPinTime* is the lifetime on the TURL, and 0 means it expired. If a TURL has an indefinite lifetime, then negative value, -1, may be used.

1.33. Return File Status for srmBringOnline

```
typedef     struct { anyURI          sourceSURL,
                     TReturnStatus   status,
                     unsigned long   fileSize,
                     int             estimatedWaitTime,
                     int             remainingPinTime,
                   } TBringOnlineRequestFileStatus
```

- *estimatedWaitTime* to be negative value, -1, for unknown.
- *remainingPinTime* is the lifetime on the TURL, and 0 means it expired. If a TURL has an indefinite lifetime, then negative value, -1, may be used.

1.34. Return File Status for srmPrepareToPut

```
typedef     struct { anyURI          SURL,
                     TReturnStatus   status,
                     unsigned long   fileSize,
                     int             estimatedWaitTime,
                     int             remainingPinLifetime  // on TURL
                     int             remainingFileLifetime  // on SURL
                     anyURI          transferURL,
                     TExtraInfo[]    transferProtocolInfo
                   } TPutRequestFileStatus
```

- *transferProtocolInfo* of type *TExtraInfo* is added to the *TPutRequestFileStatus* to give clients more information about the prepared transfer protocol so that client may use the information to make an efficient access to the prepared TURL through the transfer protocol.
- *estimatedWaitTime* to be negative value, -1, for unknown.
- *remainingPinTime* is the lifetime on the TURL, and 0 means it expired. If a TURL has indefinite lifetime, then negative value, -1, may be used.
- *remainingFileLifetime* is the lifetime on the SURL, and 0 means it expired. If SURL has an indefinite lifetime, then negative value, -1, may be used.

1.35. Return File Status for srmCopy

```
typedef     struct { anyURI          sourceSURL,
                     anyURI          targetSURL,
                     TReturnStatus   status,
                     unsigned long   fileSize,
                     int             estimatedWaitTime,
                     int             remainingFileLifetime  // on target SURL
                   } TCopyRequestFileStatus
```

- *estimatedWaitTime* to be negative value, -1, for unknown.
- *remainingFileLifetime* is the lifetime on the SURL, and 0 means it expired. If SURL has an indefinite lifetime, then negative value, -1, may be used.

1.36. Request Summary

```
typedef     struct {string          requestToken,
                    TReturnStatus   status,
                    TRequestType    requestType,
                    int             totalNumFilesInRequest,
                    int             numOfCompletedFiles,
                    int             numOfWaitingFiles,
                    int             numOfFailedFiles
                } TRequestSummary
```

- *numOfWaitingFiles* describes the number of files on the queue.
- *numOfFailedFiles* describes the number of failed files and aborted files.
- *numOfCompletedFiles* describes the number of successfully completed files, number of failed files and number of aborted files.
- *totalNumFilesInRequest* describes the *numOfWaitingFiles, numOfCompletedFiles*, *numOfFailedFiles* and number of files in progress.

1.37. Return Status for SURL

```
typedef     struct { anyURI         surl,
                    TReturnStatus   status
                    int             fileLifetime,
                    int             pinLifetime,
                } TSURLLifetimeReturnStatus
```

- *fileLifetime* describes the file lifetime on SURL.
- *pinLifetime* describes the pin lifetime on TURL, if applicable.

1.38. Return File Permissions

```
typedef     struct {anyURI          surl,
                    TReturnStatus   status,
                    TPermissionMode permission
                } TSURLPermissionReturn
```

1.39. Return Permissions on SURL

```
typedef     struct {anyURI          surl,   // both dir and file
                    TReturnStatus   status,
                    string          owner,
                    TPermissionMode ownerPermission,
                    TUserPermission[]  arrayOfUserPermissions,
                    TGroupPermission[] arrayOfGroupPermissions,
                    TPermissionMode otherPermission
                } TPermissionReturn
```

- The *TPermissionReturn* describes the permission properties of a file. It is used as an output parameter in *srmGetPermission*.

1.40. Return Request Tokens

typedef struct { string requestToken,
dateTime createdAtTime
} **TRequestTokenReturn**

1.41. Supported File Transfer Protocol

typedef struct { string transferProtocol,
TExtraInfo[] attributes
} **TSupportedTransferProtocol**

- *transferProtocol* (required): Supported transfer protocol. For example, gsiftp, http.
- *attributes*: Informational hints for the paired transfer protocol, such how many number of parallel streams can be used, desired buffer size, etc.

2. Space Management Functions

summary:

2.1. srmReserveSpace

This function is used to reserve a space in advance for the upcoming requests to get some guarantee on the file management. Asynchronous space reservation may be necessary for some SRMs to serve many concurrent requests.

2.1.1. Parameters

In:	string	authorizationID,
	String	userSpaceTokenDescription,
	TRetentionPolicyInfo	retentionPolicyInfo,
	unsigned long	desiredSizeOfTotalSpace,
	unsigned long	desiredSizeOfGuaranteedSpace,
	int	desiredLifetimeOfReservedSpace,
	unsigned long []	arrayOfExpectedFileSizes,
	TExtraInfo[]	storageSystemInfo,
	TTransferParameters	transferParameters
Out:	TReturnStatus	returnStatus,
	string	requestToken,
	int	estimatedProcessingTime,
	TRetentionPolicyInfo	retentionPolicyInfo,
	unsigned long	sizeOfTotalReservedSpace, // best effort
	unsigned long	sizeOfGuaranteedReservedSpace,
	int	lifetimeOfReservedSpace,
	string	spaceToken

2.1.2. Notes on the Behavior

a) Input parameter *userSpaceTokenDescription* is case-sensitive. SRM server is expected to keep it as client provides. It can be reused by the client. *srmGetSpaceTokens* will return all the space tokens that have the *userSpaceTokenDescription.*

b) If the input parameter *desiredLifetimeOfReservedSpace* is not provided, the lifetime of the reserved space may be set to “infinite (indefinite)” by default.

c) If particular values of the input parameter *retentionPolicyInfo* cannot be satisfied by the SRM server, SRM_NOT_SUPPORTED or SRM_NO_FREE_SPACE must be returned.
d) Asynchronous space reservation may be necessary for some SRMs to serve many concurrent requests. In such case, request token must be returned, and space token must not be assigned and returned until space reservation is completed, to prevent the usage of the space token in other interfaces before the space reservation is completed. If the space reservation can be done immediately, request token must not be returned.
e) When asynchronous space reservation is necessary, the returned status code should be SRM_REQUEST_QUEUED.
f) Input parameter *arrayOfExpectedFileSize* is a hint that SRM server can use to reserve consecutive storage sizes for the request. At the time of space reservation, if space accounting is done only at the level of the total size, this hint would not help. In such case, the expected file size at the time of srmPrepareToPut will describe how much consecutive storage size is needed for the file. However, some SRMs may get benefits from these hints to make a decision to allocate some blocks in some specific devices.
g) Optional input parameter *storageSystemInfo* is needed in case the underlying storage system requires additional security information.
h) SRM may return its default space size and lifetime if not requested by the client. SRM may return SRM_INVALID_REQUEST if SRM does not support default space sizes.
i) If input parameter *desiredSizeOfTotalSpace* is not specified, the SRM will return its default space size.
j) Output parameter *estimateProcessingTime* is used to indicate the estimation time to complete the space reservation request, when known.
k) Output parameter *sizeOfTotalReservedSpace* is in best effort bases. For guaranteed space size, *sizeOfGuaranteedReservedSpace* should be checked. These two numbers may match, depending on the storage systems.
l) Output parameter *spaceToken* is a reference handle of the reserved space.
m) If an operation is successful (SRM_SUCCESS or SRM_LOWER_SPACE_GRANTED), *sizeOfGuaranteedReservedSpace*, *lifetimeOfReservedSpace* and *spaceToken* are required to return to the client.
n) Optional input parameters in TTransferParameters may collide with the characteristics of the space specified. In this case, TTransferParameters as an input parameter must be ignored.

2.1.3. Return Status Code

SRM_SUCCESS
- successful request completion. Space is reserved successfully as the client requested.

SRM_REQUEST_QUEUED
- successful request submission and acceptance. Request token must be returned, and space token must not be assigned and returned.

SRM_REQUEST_INPROGRESS
- the request is being processed.

SRM_LOWER_SPACE_GRANTED
- successful request completion, but lower space size is allocated than what the client requested

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE

- client is not authorized to reserve space

SRM_INVALID_REQUEST
- If space size or lifetime is not requested by the client, and SRM does not support default values for space size or lifetime.
- input parameters are invalid.

SRM_NO_USER_SPACE
- SRM server does not have enough user space for the client for client to request to reserve.

SRM_NO_FREE_SPACE
- SRM server does not have enough free space for client to request to reserve.
- SRM server does not have enough free space for a particular retentionPolicyInfo

SRM_EXCEED_ALLOCATION
- SRM server does not have enough space for the client to fulfill the request because the client request needs more than the allocated space quota for the client.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server
- specific values of the input parameter *retentionPolicyInfo* is not supported by the SRM
- any input parameter is not supported in the SRM server
- a particular type of an input parameter is not supported in the SRM server

2.2. srmStatusOfReserveSpaceRequest

This function is used to check the status of the previous request to *srmReserveSpace*, when asynchronous space reservation was necessary with the SRM. Request token must have been provided in response to the *srmReserveSpace*.

2.2.1. Parameters

In:	string	authorizationID,
	string	requestToken
Out:	TReturnStatus	returnStatus,
	int	estimatedProcessingTime,
	TRetentionPolicyInfo	retentionPolicyInfo,
	unsigned long	sizeOfTotalReservedSpace,
	unsigned long	sizeOfGuaranteedReservedSpace,
	int	lifetimeOfReservedSpace,
	string	spaceToken

2.2.2. Notes on the Behavior

a) If the space reservation is not completed yet, *estimateProcessingTime* is returned when known. The returned status code in such case should be SRM_REQUEST_QUEUED.
b) See notes for *srmReserveSpace* for descriptions for output parameters.
c) If an operation is successful (SRM_SUCCESS or SRM_LOWER_SPACE_GRANTED), *sizeOfGuaranteedReservedSpace*, *lifetimeOfReservedSpace* and *spaceToken* are required to return to the client.

2.2.3. Return Status Code

SRM_REQUEST_QUEUED
- successful request submission and the request is still on the queue to be served.

SRM_REQUEST_INPROGRESS
- the request is being processed.

SRM_LOWER_SPACE_GRANTED
- successful request completion, but lower space size is allocated than what the client requested

SRM_SUCCESS
- successful request completion. Space is reserved successfully as the client requested.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to reserve space

SRM_INVALID_REQUEST
- *requestToken* does not refer to an existing known request in the SRM server.

SRM_EXCEED_ALLOCATION
- SRM server does not have enough space for the client to fulfill the request because the client request needs more than the allocated space for the client.

SRM_NO_USER_SPACE
- SRM server does not have enough user space for the client for the client for client to request to reserve.

SRM_NO_FREE_SPACE
- SRM server does not have enough free space for the client for client to request to reserve.
- SRM server does not have enough free space for a particular retentionPolicyInfo

SRM_REQUEST_SUSPENDED
- request is suspended.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server
- any input parameter is not supported in the SRM server
- a particular type of an input parameter is not supported in the SRM server

2.3. srmReleaseSpace

srmReleaseSpace() releases an occupied space.

2.3.1. Parameters

In:	string	authorizationID,
	string	spaceToken,
	TExtraInfo[]	storageSystemInfo,
	Boolean	forceFileRelease
Out:	TReturnStatus	returnStatus

2.3.2. Notes on the Behavior

a) *forceFileRelease* is false by 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* must be specified to be true.
b) When space is releasable and *forceFileRelease* is true, all the files in the space are released, even in OUTPUT or CUSTODIAL retention quality space.
c) *srmReleaseSpace* may not complete right away because of the lifetime of existing files in the space. When space is released, the files in that space are treated according to their types: If file storage types are permanent, keep them until further operation such as *srmRm* is issued by the client. If file storage types are durable, perform necessary actions at the end of their lifetime. If file storage types are volatile, release those files at the end of their lifetime.
d) If space is being released with *forceFileRelease* option while SURLs are being created with *srmPrepareToPut* or *srmCopy*, the file is removed and SRM_INVALID_PATH must be returned by the *srmPutDone*, *srmStatusOfPutRequest,* or *srmStatusOfCopyRequest* when the file is volatile. If the file is permanent type, the file is moved to the default space, and the space would be successfully released. The subsequent *srmPutDone*, *srmStatusOfPutRequest*, or *srmStatusOfCopyRequest* would be successful.
e) If space is being released without *forceFileRelease* option while SURLs are being created with *srmPrepareToPut* or *srmCopy*, SRM_FAILURE must be returned in *srmReleaseSpace*.
f) When a "replica" quality space is expired on its lifetime, all files inside must be expired (by definition, file lifetimes are less than and equal to the remaining lifetime of the space). After the space is expired, the space that is associated with the space token no longer exists, along with all files inside - meaning their SURLs disappear from the file system or reflect the expired lifetime.

2.3.3. Return Status Code

SRM_SUCCESS
- successful request completion. Space is successfully released.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to release the space that is associated with the *spaceToken*

SRM_INVALID_REQUEST
- *spaceToken* does not refer to an existing known space in the SRM server.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- *forceFileRelease* is not supported
- *function* is not supported

SRM_FAILURE
- space still contains pinned files.
- space associated with space is already released.
- any other request failure. *Explanation* needs to be filled for details.

2.4. srmUpdateSpace

srmUpdateSpace is to resize the space and/or extend the lifetime of a space. Asynchronous operation may be necessary for some SRMs to serve many concurrent requests.

2.4.1. Parameters

In:	string	authorizationID,
	string	spaceToken,
	unsigned long	newSizeOfTotalSpaceDesired,
	unsigned long	newSizeOfGuaranteedSpaceDesired,
	int	newLifeTime,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	returnStatus,
	string	requestToken,
	unsigned long	sizeOfTotalSpace, // best effort
	unsigned long	sizeOfGuaranteedSpace,
	int	lifetimeGranted

2.4.2. Notes on the Behavior

a) If neither size nor lifetime is provided in the input parameters, then the request will be failed, and SRM_INVALID_REQUEST must be returned. The existing values must not be changed.
b) newSize is the new actual size of the space.
c) *newLifetime* is the new lifetime requested regardless of the previous lifetime. It might even be shorter than the remaining lifetime at the time of the call. It is relative to the calling time. Lifetime will be set from the calling time for the specified period.
d) Output parameter, *lifetimeGranted* is the new lifetime granted regardless of the previous lifetime. It might even be shorter than the previous lifetime. It is relative to the calling time.

2.4.3. Return Status Code

SRM_SUCCESS
- successful request completion. Space is successfully updated as the client requested.

SRM_ REQUEST_QUEUED
- successful request submission and acceptance. Request token must be returned.

SRM_LOWER_SPACE_GRANTED
- successful request completion, but lower space size is allocated than what the client requested

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to update the space that is associated with the *spaceToken*

SRM_SPACE_LIFETIME_EXPIRED
- lifetime of the space that is associated with the *spaceToken* is already expired.

SRM_INVALID_REQUEST
- *spaceToken* does not refer to an existing known space in the SRM server.
- input parameter size or time is not provided.

SRM_EXCEED_ALLOCATION
- SRM server does not have enough space for the client to fulfill the request because the client request has more than the allocated space for the client.

SRM_NO_USER_SPACE
- SRM server does not have enough space for the client to fulfill the request

SRM_NO_FREE_SPACE
- SRM server does not have enough free space to fulfill the request

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- New requested size is less than currently used space.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported

2.5. srmStatusOfUpdateSpaceRequest

This function is used to check the status of the previous request to *srmUpdateSpace*, when asynchronous space update was necessary with the SRM. Request token must have been provided in response to the *srmUpdateSpace*.

2.5.1. Parameters

In:	string	authorizationID,
	string	requestToken
Out:	TReturnStatus	returnStatus,
	unsigned long	sizeOfTotalSpace, // best effort
	unsigned long	sizeOfGuaranteedSpace,
	int	lifetimeGranted

2.5.2. Notes on the Behavior

a) Output parameters for sew sizes are the new actual sizes of the space.
b) Output parameter, *lifetimeGranted* is the new lifetime granted regardless of the previous lifetime. It might even be shorter than the previous lifetime. It is relative to the calling time.

2.5.3. Return Status Code

SRM_REQUEST_QUEUED

- successful request submission and the request is still on the queue to be served.

SRM_REQUEST_INPROGRESS
- the request is being processed.

SRM_SUCCESS
- successful request completion. Space is successfully updated as the client requested.

SRM_LOWER_SPACE_GRANTED
- successful request completion, but lower space size is allocated than what the client requested

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to update the space that is associated with the *spaceToken*

SRM_SPACE_LIFETIME_EXPIRED
- lifetime of the space that is associated with the *spaceToken* is already expired.

SRM_INVALID_REQUEST
- *spaceToken* does not refer to an existing known space in the SRM server.
- input parameter size or time is not provided.

SRM_EXCEED_ALLOCATION
- SRM server does not have enough space for the client to fulfill the request because the client request has more than the allocated space for the client.

SRM_NO_USER_SPACE
- SRM server does not have enough space for the client to fulfill the request

SRM_NO_FREE_SPACE
- SRM server does not have enough free space to fulfill the request

SRM_REQUEST_SUSPENDED
- request is suspended.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- New requested size is less than currently used space.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported

2.6. srmGetSpaceMetaData

This function is used to get information of a space. Space token must be provided, and space tokens are returned upon a completion of a space reservation through *srmReserveSpace* or *srmStatusOfReserveSpaceRequest*.

2.6.1. Parameters

In:	string	authorizationID,
	string[]	arrayOfSpaceTokens
Out:	TReturnStatus	returnStatus,

TMetaDataSpace[] arrayOfSpaceDetails

2.6.2. Notes on the Behavior

a) Output parameters *unusedSize* in *TMetaDataSpace* returns 0 if there is no space left in the allocated space.
b) When clients use more space than allocated, clients get warned to accommodate their files in the spaces or update the space before running out. SRM

2.6.3. Return Status Code

For request level return Status,

SRM_SUCCESS
- successful request completion. Information of all requested spaces are returned successfully.

SRM_PARTIAL_SUCCESS
- Request is completed. Information of some requested spaces are returned successfully, and some are failed to be returned.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request space information

SRM_TOO_MANY_RESULTS
- Request produced too many results that SRM server cannot handle.

SRM_INVALID_REQUEST
- *arrayOfSpaceToken* is empty.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All space requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server

For space level return Status,

SRM_SUCCESS
- successful request completion for the *spaceToken.* Space information is successfully returned.

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request information on the space that is associated with the *spaceToken*

SRM_INVALID_REQUEST
- *spaceToken* does not refer to an existing known space in the SRM server.

SRM_SPACE_LIFETIME_EXPIRED
- The life time on the space that is associated with the *spaceToken* has expired

SRM_EXCEED_ALLOCATION
- Space that is associated with *spaceToken* has no more space left.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

2.7. srmChangeSpaceForFiles

This function is used to change the space property of files to another space property by specifying target space tokens. All files specified by SURLs will have a new space token. SURLs must not be changed. New space token may be acquired from *srmReserveSpace*. Asynchronous operation may be necessary for some SRMs, and in such case, request token is returned for later status inquiry. There is no default behavior when target space token is not provided. In such case, the request will be rejected, and the return status must be SRM_INVALID_REQUEST.

2.7.1. Parameters

In:	string	authorizationID,
	anyURI []	arrayOfSURLs,
	string	targetSpaceToken,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	returnStatus,
	string	requestToken,
	int	estimatedProcessingTime,
	TSURLReturnStatus []	arrayOfFileStatuses

2.7.2. Notes on the Behavior

a) When space transition is completed successfully, SRM_SUCCESS must be returned for each SURL.
b) For any forbidden transition by the SRM implementation, SRM_INVALID_REQUEST must be returned. It includes changing spaces on SURLs that statuses are SRM_FILE_BUSY.
c) Asynchronous operation may be necessary for some SRMs to serve many concurrent requests. In such case, request token must be returned. If the request can be completed immediately, request token must not be returned.
d) When asynchronous operation is necessary, the returned status code should be SRM_REQUEST_QUEUED, and *arrayOfFileStatuses* may not be filled and returned.
e) All files specified in *arrayOfSURLs* will be moved to the space associated with *targetSpaceToken.*
f) When target space token is used, space allocation for a new space token must be done explicitly by the client before using this function.
g) If a directory path is provided, then the effect is recursive for all files in the directory.
h) Space de-allocation may be necessary in some cases, and it must be done by the client explicitly after this operation completes. The status can be checked by srmStatusOfChangeSpaceForFilesRequest.
i) When a space is successfully changed for a file from one space to another, it will either retain its remaining lifetime, or the lifetime will be reduced to that of the target space, whichever is the lesser.
j) If the target space is only large enough to transfer a subset of the files, the request will continue taking place until the target space cannot hold any more files, and the request must be failed. The status of the request must return an error of SRM_EXCEED_ALLOCATION in such case.

2.7.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All file requests are successfully completed. All *SURLs* have new *targetSpaceToken*.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some SURL requests have new *targetSpaceToken*, and some SURL requests are failed to have new *targetSpaceToken*. Details are on the files status.

SRM_ REQUEST_QUEUED
- request is submitted and accepted. *requestToken* must be returned.
- The status can be checked by *srmStatusOfChangeSpaceForFilesRequest*.

SRM_ REQUEST_INPROGRESS
- The request is being processed. Some files are still queued, and some files are completed in space transition.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to change the file types

SRM_INVALID_REQUEST
- *SURL* is empty.
- *targetSpaceToken* is empty.
- *targetSpaceToken* does not refer to an existing space in the SRM server.
- *targetSpaceToken* refers to a forbidden transition by the SRM implementation.

SRM_SPACE_LIFETIME_EXPIRED
- target space that is associated with *targetSpaceToken* has an expired lifetime.

SRM_EXCEED_ALLOCATION
- target space that is associated with *targetSpaceToken* is not enough to hold all *SURLs*.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM
- any input parameter is not supported in the SRM server
- a particular type of an input parameter is not supported in the SRM server

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL.* The *SURL* has a new *targetSpaceToken.*

SRM_ REQUEST_QUEUED
- file request is on the queue.

SRM_ REQUEST_INPROGRESS
- file request is being processed.

SRM_INVALID_PATH
- *SURL* does not refer to an existing file

SRM_AUTHORIZATION_FAILURE
- client is not authorized to change the space for the file that is associated with the *SURL*

SRM_INVALID_REQUEST
- *targetSpaceToken* refers to a forbidden transition for the particular SURL by the SRM implementation.
- The status of SURL is SRM_FILE_BUSY.

SRM_EXCEED_ALLOCATION
- target space that is associated with *targetSpaceToken* is not enough to hold *SURL*.

SRM_FILE_LOST
- the requested file with the SURL is permanently lost.

SRM_FILE_BUSY
- client requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.
- The requested file with the SURL is being used by other clients.

SRM_FILE_UNAVAILABLE
- the requested file with the SURL is temporarily unavailable.

SRM_FAILURE
- All file requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

2.8. srmStatusOfChangeSpaceForFilesRequest

This function is used to check the status of the previous request to *srmChangeSpaceForFiles,* when asynchronous operation was necessary in the SRM. Request token must have been provided in response to the *srmChangeSpaceForFiles*.

2.8.1. Parameters

In:	string	authorizationID,
	string	requestToken
Out:	TReturnStatus	returnStatus
	int	estimatedProcessingTime,
	TSURLReturnStatus []	arrayOfFileStatuses

2.8.2. Notes on the Behavior

a) When space transition is completed successfully, SRM_SUCCESS must be returned for each SURL.
b) If changing space is not completed, *estimateProcessingTime* is returned when known.
c) If all files are still in the queue and none of the files are completed in changing space, the returned status code should be SRM_REQUEST_QUEUED.
d) If some files are queued, and some files are completed in changing space, SRM_REQUEST_INPROGRESS must be returned as the return status code. Each file should have its own status code.
e) If the target space is only large enough to transfer a subset of the files, the request will continue taking place until the target space cannot hold any more files, and the request must be failed. The status of the request must return an error of SRM_EXCEED_ALLOCATION in such case.

2.8.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All file requests are successfully completed. All *SURLs* have new *targetSpaceToken*.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some SURL requests have new *targetSpaceToken*, and some SURL requests are failed to have new *targetSpaceToken*. Details are on the files status.

SRM_ REQUEST_QUEUED
- Request submission was successful and the entire request is still on the queue.

SRM_ REQUEST_INPROGRESS
- Some files are still queued, and some files are completed in space transition.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to change the file types

SRM_INVALID_REQUEST
- *requestToken* does not refer to an existing known request in the SRM server.
- *targetSpaceToken* refers to a forbidden transition by the SRM implementation.

SRM_SPACE_LIFETIME_EXPIRED
- target space that is associated with *targetSpaceToken* has an expired lifetime.

SRM_EXCEED_ALLOCATION
- target space that is associated with *targetSpaceToken* is not enough to hold *SURLs*.

SRM_REQUEST_SUSPENDED
- request is suspended.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All file requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM
- any input parameter is not supported in the SRM server
- a particular type of an input parameter is not supported in the SRM server

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL.* The *SURL* has a new *targetSpaceToken.*

SRM_ REQUEST_QUEUED
- file request is on the queue.

SRM_ REQUEST_INPROGRESS
- file request is being processed.

SRM_INVALID_PATH
- *SURL* does not refer to an existing file request

SRM_AUTHORIZATION_FAILURE
- client is not authorized to change the space for the file that is associated with the *SURL*

SRM_INVALID_REQUEST

- *targetSpaceToken* refers to a forbidden transition for the particular SURL by the SRM implementation.
- The status of SURL is SRM_FILE_BUSY.

SRM_EXCEED_ALLOCATION
- target space that is associated with *targetSpaceToken* is not enough to hold *SURL*.

SRM_REQUEST_SUSPENDED
- file request is suspended.

SRM_FILE_LOST
- the requested file with the SURL is permanently lost.

SRM_FILE_BUSY
- client requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.
- The requested file with the SURL is being used by other clients.

SRM_FILE_UNAVAILABLE
- the requested file with the SURL is temporarily unavailable.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

2.9. srmExtendFileLifeTimeInSpace

This function is used to extend lifetime of the files (SURLs) in a space.

2.9.1. Parameters

In:	string	authorizationID,
	string	spaceToken,
	anyURI []	arrayOfSURLs,
	int	newLifeTime
Out:	TReturnStatus	returnStatus,
	TSURLLifetimeReturnStatus []	arrayOfFileStatuses

2.9.2. Notes on the Behavior

a) *arrayOfSURLs* are optional. When SURLs are not provided, all files in the space must have the new extended lifetimes.
b) *newLifeTime* is relative to the calling time. Lifetime will be set from the calling time for the specified period.
c) The new file lifetime, *newLifeTime* must not exceed the remaining lifetime of the space.
d) The number of lifetime extensions may be limited by SRM according to its policies.
e) If original lifetime is longer than the requested one, then the new requested one will be assigned.
f) If *newLifeTime* is not specified, the SRM does not change the lifetime.
g) If input parameters *newLifeTime* request exceed the remaining lifetime of the space, then SRM_SUCCESS is returned at the request and file level, and *TSURLLifetimeReturnStatus* contains the remaining lifetime.
h) Lifetime extension must fail on SURLs when their status is SRM_FILE_BUSY.

i) This method applied only to SURLs, and output parameter *pinLifetime* in *TSURLLifetimeReturnStatus* must be null.

2.9.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. All *SURLs* have a new extended lifetime.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some *SURLs* have a new extended lifetime, and some *SURLS* have failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to extend lifetime of files in the space specified by the space token.

SRM_INVALID_REQUEST
- *spaceToken* is empty.
- *spaceToken* does not refer to an existing known space in the SRM server.

SRM_SPACE_LIFETIME_EXPIRED
- lifetime of the space that is associated with the *spaceToken* is already expired.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All file requests updating lifetimes in a space are failed.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server
- any input parameter is not supported in the SRM server
- a particular type of an input parameter is not supported in the SRM server

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL.* The *SURL* has a new extended lifetime.

SRM_INVALID_PATH
- *SURL* does not refer to an existing file request
- *SURL* does not refer to an existing file request that is associated with the space token

SRM_AUTHORIZATION_FAILURE
- client is not authorized to extend the lifetime for the file that is associated with the *SURL*

SRM_FILE_LOST
- the requested file is permanently lost.

SRM_FILE_UNAVAILABLE
- the requested file is temporarily unavailable.

SRM_FILE_LIFETIME_EXPIRED
- the requested file is expired already.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

2.10. srmPurgeFromSpace

This function is used when removing files from the given space is needed. Difference from *srmReleaseFiles* and *srmAbortFiles* is that *srmPurgeFromSpace* is not associated with a request. This function must not remove the SURLs, but only the "copies" or "states" of the SURLs. *srmRm* must be used to remove SURLs.

2.10.1. Parameters

In:	string	authorizationID
	anyURI []	arrayOfSURLs
	string	spaceToken,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	returnStatus,
	TSURLReturnStatus[]	arrayOfFileStatuses

2.10.2. Notes on the Behavior

a) If the specified *SURL* is the only remaining copy of the file in the storage system, SRM_LAST_COPY must be returned. To remove the last copy of the *SURL*, *srmRm* may be used.
b) If the client has an administers role that SRM server can accept in an understandable form, this request will forcefully release the pins owned by the group, and remove the “copy” (or “state”) of the file.
c) In most cases, all pins on files that are associated with the client will be released. In such cases, files may still be pinned by others and SRM_FILE_BUSY will be returned.
d) SRM will remove only the “copies” (or “state”) of the *SURLs* associated with the space token.

2.10.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. All *SURLs* are purged from the space specified by the *spaceToken*.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some *SURLs* are successfully purged from the space specified by the *spaceToken*, and some *SURLs* are failed to be purged from the space specified by the *spaceToken*. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to clean up the space that is associated with *spaceToken*

SRM_INVALID_REQUEST
- *arrayOfSURLs* is empty.
- *spaceToken* is empty.
- *spaceToken* does not refer to an existing known space in the SRM server.

SRM_INTERNAL_ERROR

- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All file requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server

For file level return Status,

SRM_SUCCESS
- successful request completion for the *SURL. SURL* is purged from the space specified by the *spaceToken*.

SRM_INVALID_PATH
- *SURL* does not refer to an existing file
- *SURL* does not refer to an existing file that is associated with the space token

SRM_AUTHORIZATION_FAILURE
- Client is not authorized to purge *SURL* in the space that is associated with *spaceToken*

SRM_FILE_LOST
- the request file is permanently lost.

SRM_FILE_BUSY
- client requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.
- The requested file is used by other clients.

SRM_FILE_UNAVAILABLE
- the requested file is temporarily unavailable.

SRM_LAST_COPY
- the requested file is the last copy and will not be purged from the space. *srmRm* must be used to remove the last copy.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

2.11. srmGetSpaceTokens

srmGetSpaceTokens() returns space tokens for currently allocated spaces.

2.11.1. Parameters

In:	string	userSpaceTokenDescription,
	string	authorizationID
Out:	TReturnStatus	returnStatus
	string[]	arrayOfSpaceTokens

2.11.2. Notes on the Behavior

a) If *userSpaceTokenDescription* is null, returns all space tokens this user owns.

b) Input parameter *userSpaceTokenDescription* is case-sensitive. SRM server is expected to keep it as client provides. It can be reused by the client. *srmGetSpaceTokens* will return all the space tokens that have the *userSpaceTokenDescription.*
c) If the user assigned the same name to multiple space reservations, he may get back multiple space tokens.

2.11.3. Return Status Code

SRM_SUCCESS
- All requests are successfully completed. Space tokens are returned successfully.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request *spaceTokens* associated with the *userSpaceTokenDescription*

SRM_INVALID_REQUEST
- *userSpaceTokenDescription* does not refer to an existing space description.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server

3. Permission Functions

summary:

3.1. srmSetPermission

srmSetPermission is to set permission on local SURL.

3.1.1. Parameters

In:	string	authorizationID,
	anyURI	SURL,
	TPermissionType	permissionType,
	TPermissionMode	ownerPermission,
	TUserPermission[]	arrayOfUserPermissions,
	TGroupPermission[]	arrayOfGroupPermissions,
	TPermissionMode	otherPermission,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	returnStatus

3.1.2. Notes on the Behavior

a) Applies to both dir and file.
b) Support for *srmSetPermission* is optional.
c) User permissions are provided in order to support dynamic user-level permission assignment similar to Access Control Lists (ACLs).
d) Permissions can be assigned to set of users and sets of groups, but only a single owner.
e) In this version, SRMs do not provide any group operations (setup, modify, remove, etc.)
f) Groups are assumed to be set up before *srmSetPermission* is used.
g) If *TPermissionType* is ADD or CHANGE, and *TPermissionMode* is null, then it is assumed that *TPermissionMode* is READ only.
h) If *TPermissionType* is REMOVE, then the *TPermissionMode* is ignored.
i) if *TPermissionType* is CHANGE, but it is being applied to a [user|group] which currently does not have permissions set up for it, then the request works as ADD. It follows the setfacl: Adds one or more new ACL entries to the file, and/or modifies one or more existing ACL entries on the file. If an entry already exists for a specified uid or gid, the specified permissions will replace the current permissions. If an entry does not exist for the specified uid or gid, an entry will be created.
j) *srmSetPermission* will modify permissions on SURLs even if the statuses of the SURLs are SRM_FILE_BUSY.

3.1.3. Return Status Code

SRM_SUCCESS

- successful request completion. *SURL* has a new permission.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to set permissions
- client is not authorized to set permissions on the *SURL*

SRM_INVALID_PATH
- *SURL* does not refer to an existing known path

SRM_INVALID_REQUEST
- Permissions are provided incorrectly

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server
- any input parameter is not supported in the SRM server
- a particular type of an input parameter is not supported in the SRM server

3.2. srmCheckPermission

srmCheckPermission is used to check the client permissions on the SURLs. It only checks for the client for authorization on the *SURLs* in the local storage.

3.2.1. Parameters

In:	anyURI []	arrayOfSURLs,
	string	authorizationID,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	returnStatus,
	TSURLPermissionReturn[]	arrayOfPermissions

3.2.2. Notes on the Behavior

a) SRM will check files in its local online and nearline storage.

3.2.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. Permissions on *SURLs* are checked and returned.

SRM_PARTIAL_SUCCESS
- All requests are completed. Permissions of some SURLs are successfully checked and returned, but some permission of some *SURLs* are failed to be checked. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request permission information

SRM_INVALID_REQUEST
- *arrayOfSURL* is empty.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL.* Permissions on *SURL* are checked and returned.

SRM_INVALID_PATH
- *SURL* does not refer to an existing known path

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request permission information on the *SURL*

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

3.3. srmGetPermission

srmGetPermission is used to get the permissions on the SURLs. It only checks for the client for authorization on the *SURLs* in the local storage.

3.3.1. Parameters

In:	anyURI []	arrayOfSURLs,
	string	authorizationID,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	returnStatus,
	TPermissionReturn[]	arrayOfPermissionReturns

3.3.2. Notes on the Behavior

b) SRM will check files in its local online and nearline storage.

3.3.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. Permissions on *SURLs* are returned.

SRM_PARTIAL_SUCCESS

- All requests are completed. Permissions of some SURLs are successfully returned, but some permission of some *SURLs* are failed to be returned. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request permission information

SRM_INVALID_REQUEST
- *arrayOfSURL* is empty.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL.* Permissions on *SURL* are returned.

SRM_INVALID_PATH
- *SURL* does not refer to an existing known path

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request permission information on the *SURL*

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

4. Directory Functions

summary:

4.1. srmMkdir

srmMkdir create a directory in a local SRM space.

4.1.1. Parameters

In:	string	authorizationID,
	anyURI	SURL,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	returnStatus

4.1.2. Notes on the Behavior

a) Consistent with unix, recursive creation of directories is not supported.
b) *SURL* is a directory path and can include paths, as long as all directory hierarchy exists.

4.1.3. Return Status Code

SRM_SUCCESS
- All requests are successfully completed. *SURL* is created.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to create a directory
- client is not authorized to create a directory as *SURL*

SRM_INVALID_PATH
- *SURL* does not refer to a valid path
- component of *SURL* does not refer to an existing path

SRM_DUPLICATION_ERROR
- *SURL* exists already

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server

4.2. srmRmdir

srmRmdir removes an empty directory in a local SRM space.

4.2.1. Parameters

In:	string	authorizationID,	
	anyURI	SURL,	
	TExtraInfo[]	storageSystemInfo,	
	boolean	recursive	// false by default
Out:	TReturnStatus	returnStatus	

4.2.2. Notes on the Behavior

a) It applies to directory only.
b) *recursive* is false by default.
c) To distinguish from *srmRm*(), this function is for directories only
d) When only expired volatile files are in the requested directory, srmRmdir must allow the removal of the requested directory regardless of the expired files. The SURL of the expired volatile files must no longer exist in the file system, and may or may not be removed right away physically depending on the internal server policy.

4.2.3. Return Status Code

SRM_SUCCESS
- All requests are successfully completed. *SURL* is removed.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to remove a directory
- client is not authorized to remove a directory as *SURL*

SRM_INVALID_PATH
- *SURL* does not refer to a valid path

SRM_NON_EMPTY_DIRECTORY
- *SURL* is not empty

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server
- input parameter *recursive* is not supported in the SRM server

4.3. srmRm

This function will remove SURLs (the name space entries) in the storage system. Difference from *srmPurgeFromSpace* is that *srmPurgeFromSpace* removes only previously requested “copies” (or

"state") of the SURL in a particular space, and *srmPurgeFromSpace* shall not remove SURLs or the name space entries.

4.3.1. Parameters

In:	string	authorizationID,
	anyURI[]	arrayOfSURLs,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	returnStatus,
	TSURLReturnStatus[]	arrayOfFileStatuses

4.3.2. Notes on the Behavior

a) To distinguish from *srmRmdir*(), this function applies to files only
b) *srmRm* removes all copies or states on the storage, and removes the entry from the name space.
c) When an SURL is removed, all associated pinned TURLs are all released and removed as well.
d) *srmLs, srmPrepareToGet* or *srmBringOnline* will not find these removed files any more. It must set file requests on SURL from srmPrepareToGet as SRM_ABORTED.
e) *srmRm* aborts the SURLs from *srmPrepareToPut* requests not yet in SRM_PUT_DONE state, and must set its file status as SRM_ABORTED.
f) *srmRm* will remove SURLs even if the statuses of the SURLs are SRM_FILE_BUSY. In this case, operations such as srmPrepareToPut or srmCopy that holds the SURL status as SRM_FILE_BUSY must return SRM_INVALID_PATH upon status request or srmPutDone.

4.3.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. All *SURLs* are removed.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some *SURLs* are successfully removed, and some *SURLs* are failed to be removed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to remove any files

SRM_INVALID_REQUEST
- *arrayOfSURLs* is empty.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_SUCCESS

- successful request completion for the *SURL*. *SURL* is removed.

SRM_INVALID_PATH
- *SURL* does not refer to an existing known file path

SRM_AUTHORIZATION_FAILURE
- client is not authorized to remove *SURL*

SRM_FILE_LOST
- the request file is permanently lost.

SRM_FILE_UNAVAILABLE
- the request file is temporarily unavailable.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

4.4. srmLs

srmLs() returns a list of files with a basic information. This operation may be asynchronous, and in such case, requestToken must be returned.

4.4.1. Parameters

In:	string	authorizationID,
	anyURI []	arrayOfSURLs,
	TExtraInfo[]	storageSystemInfo,
	TFileStorageType	fileStorageType,
	boolean	fullDetailedList,
	boolean	allLevelRecursive,
	int	numOfLevels,
	int	offset,
	int	count
Out:	TReturnStatus	returnStatus
	string	requestToken
	TMetaDataPathDetail[]	details

4.4.2. Notes on the Behavior

a) Applies to both directory and file
b) *fullDetailedList* is false by default.
   - For directories, only path is required to be returned.
   - For files, path and size are required to be returned.
c) If *fullDetailedList* is true, the full details are returned.
   - For directories (numOfLevels=0) or a single file *, path, size, userPermission, lastModificationTime, type, fileLocality,* and *lifetimeLeft* are required to be returned, similar to unix command *ls –l*.
   - For directories (numOfLevels=1) *, path, size, userPermission, lastModificationTime,* and *type* are required to be returned.
d) If *allLevelRecursive* is true then file lists of all level below current will be provided as well.

e) If *allLevelRecursive* is "true" it dominates, i.e. ignore *numOfLevels*. If *allLevelRecursive* is "false" or missing, then do *numOfLevels*. If *numOfLevels* is "0" (zero) or missing, assume a single level. If both *allLevelRecursive* and *numOfLevels* are missing, assume a single level.
f) Default value of *numOfLevels* is 1 when not provided.
g) If *numOfLevels* is 0, then information about directory itself is returned. Negative value is invalid.
h) If *numOfLevels* is 1, then information about files in the directory is returned. Negative value is invalid.
i) For directory path, appending a slash (/) at the end of the path is recommended.
j) When listing for a particular type specified by "*fileStorageType*", only the files with that type will be in the output.
k) Empty directories will be returned.
l) For non-existing or system-prohibited file or directory browsing, SRM_INVALID_PATH must be returned. For non-supported file or directory browsing, SRM_NOT_SUPPORTED must be returned. *Explanation* needs to be filled for details.
m) When browsing the top directory is not supported by the SRM, SRM_NOT_SUPPORTED must be returned at the file level.

4.4.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. All *SURLs* are checked and the information for all *SURLs* is returned successfully.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some *SURL* request is successfully completed, and some *SURL* request is failed. Details are on the files status.

SRM_REQUEST_QUEUED
- successful request submission and acceptance. Request token must be returned.

SRM_REQUEST_INPROGRESS
- Some files are completed, and some files are still on the queue. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request information

SRM_TOO_MANY_RESULTS
- srmLs request has generated too many results that SRM cannot handle. In most cases, it needs to be narrowed down with offset and count by the client.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_INVALID_REQUEST
- Negative values for *numOfLevels*, *offset* and *count* are provided.
- Operation on the path such as browsing the top directory may be prohibited. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- Requested *fileStorageType* is not supported in SRM
- Filtering *fileStorageType* is not supported in SRM
- Directory operation (directory SURL, allLevelRecursive and numOfLevels) is not supported in SRM

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL.* The information for the *SURL* is checked and returned successfully.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_REQUEST_QUEUED
- file request is still on the queue.

SRM_INVALID_PATH
- *SURL* does not refer to an existing known file path.

SRM_AUTHORIZATION_FAILURE
- client is not authorized to receive the information of the *SURL* or to access the directory or sub-directories

SRM_FILE_BUSY
- client requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.

SRM_FILE_LIFETIME_EXPIRED
- lifetime on *SURL* is expired. There is no guarantee of the file still in the cache.

SRM_FILE_IN_CACHE
- lifetime on *SURL* has expired, but the file is still in the cache.

SRM_NOT_SUPPORTED
- Operation on the path such as browsing the top directory may be not supported. *Explanation* needs to be filled for details.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

4.5. srmStatusOfLsRequest

srmStatusOfLsRequest() returns a list of files with a basic information. This is an asynchronous operation of srmLs.

4.5.1. Parameters

In:	string	authorizationID,
	string	requestToken
	int	offset,
	int	count
Out:	TReturnStatus	returnStatus
	TMetaDataPathDetail[]	details

4.5.2. Notes on the Behavior

a) Empty directories will be returned.
b) For non-existing file or directory, SRM_INVALID_PATH must be returned.

4.5.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. All *SURLs* are checked and the information for all *SURLs* is returned successfully.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some *SURL* request is successfully completed, and some *SURL* request is failed. Details are on the files status.

SRM_REQUEST_QUEUED
- successful request submission and all files request is still on the queue.

SRM_REQUEST_INPROGRESS
- Some files are completed, and some files are still on the queue. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request information

SRM_TOO_MANY_RESULTS
- srmLs request has generated too many results that SRM cannot handle. In most cases, it needs to be narrowed down with offset and count by the client.

SRM_INVALID_REQUEST
- Negative values for *offset* and *count* are provided.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- Requested *fileStorageType* is not supported in SRM
- Filtering *fileStorageType* is not supported in SRM
- Directory operation (directory SURL, allLevelRecursive and numOfLevels) is not supported in SRM

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL.* The information for the *SURL* is checked and returned successfully.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_REQUEST_QUEUED
- file request is still on the queue.

SRM_INVALID_PATH
- *SURL* does not refer to an existing known file path

SRM_AUTHORIZATION_FAILURE

- client is not authorized to receive the information of the *SURL* or to access the directory or sub-directories

SRM_FILE_BUSY
- client requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.

SRM_FILE_LIFETIME_EXPIRED
- lifetime on *SURL* is expired. There is no guarantee of the file still in the cache.

SRM_FILE_IN_CACHE
- lifetime on *SURL* has expired, but the file is still in the cache.

SRM_NOT_SUPPORTED
- Operation on the path such as browsing the top directory may be not supported. *Explanation* needs to be filled for details.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

4.6. srmMv

srmMv is to move a file or a directory to destination.

4.6.1. Parameters

In:	string	authorizationID,
	anyURI	fromSURL,
	anyURI	toSURL,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	returnStatus

4.6.2. Notes on the Behavior

a) Applies to both directory and file, and works like unix *mv*.
b) Authorization checks need to be performed on both *fromSURL* and *toSURL*.
c) *srmMv* must fail on SURL that its status is SRM_FILE_BUSY, and SRM_INVALID_REQUEST must be returned.
d) Moving an SURL to itself results in no operation and SRM_SUCCESS will be returned for no operation.
e) When moving an SURL to already existing SURL, SRM_DUPLICATION_ERROR must be returned.

4.6.3. Return Status Code

SRM_SUCCESS
- All requests are successfully completed. *SURL* is moved successfully from one local path to another local path.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to move *fromSURL*.
- Client is not authorized to move a file into *toSURL*

SRM_INVALID_PATH
- *fromSURL* does not refer to an existing known path
- *toSURL* does not refer to a valid path
- status of *fromSURL* is SRM_FILE_BUSY.

SRM_DUPLICATION_ERROR
- *toSURL* exists already.

SRM_FILE_LOST
- the requested file is permanently lost.

SRM_FILE_BUSY
- client requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.
- The requested file is being used by other clients.

SRM_FILE_UNAVAILABLE
- the requested file is temporarily unavailable.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server

5. Data Transfer Functions

summary:

5.1. srmPrepareToGet

This function is used to bring files online upon the client's request and assign TURL so that client can access the file. Lifetime (pinning expiration time) is assigned on the TURL. When specified target space token which must be referred to an online space, the files will be prepared using the space associated with the space token. It is an asynchronous operation, and request token must be returned if request is valid and accepted. The status must be checked through srmStatusOfGetRequest with the returned request token.

5.1.1. Parameters

In:	string	authorizationID,
	TGetFileRequest[]	arrayOfFileRequests,
	string	userRequestDescription,
	TExtraInfo[]	storageSystemInfo,
	TFileStorageType	desiredFileStorageType
	int	desiredTotalRequestTime
	int	desiredPinLifetime,
	string	targetSpaceToken
	TRetentionPolicyInfo	targetFileRetentionPolicyInfo
	TTransferParameters	transferParameters

Out:	TReturnStatus	returnStatus
	string	requestToken,
	TGetRequestFileStatus[]	arrayOfFileStatuses
	int	remainingTotalRequestTime

5.1.2. Notes on the Behavior

a) The default value of "lifetime" for Volatile or Durable files will be the lifetime left in the space of the corresponding file type. The default value of "fileStorageType" is Volatile.
b) If input parameter *targetSpaceToken* is provided, then the target space token must refer to online space. All requested files will be prepared into the target space.
c) Input parameter *targetFileRetentionPolicyInfo* of *TRetentionPolicyInfo* is to specify the desired retention policy information on the file when the file is prepared online.
d) If both input parameters *targetSpaceToken* and *TRetentionPolicyInfo* are provided, then their types must match exactly. Otherwise, the request may be rejected with SRM_INVALID_REQUEST.
e) Access latency must be ONLINE always.
f) Input parameter *TAccessPattern* is provided at the request-level, and all files will have the same access pattern.
g) Optional input parameters in *TTransferParameters* may collide with the characteristics of the space specified. In this case, *TTransferParameters* as an input parameter must be ignored.
o) The *userRequestDescription* is a user designated name for the request. It is case-sensitive. SRM server is expected to keep it as client provides. It can be reused by the client. It can be used in the *srmGetRequestTokens* function to get back the system assigned request tokens. *srmGetRequestTokens* will return all the request tokens that have the *userRequestDescription.*
h) Only pull mode is supported for file transfers that client must pull the files from the TURL within the expiration time (*remainingPinTime*).
i) Input parameter *desiredPinLifetime* is for a client preferred lifetime (expiration time) on the prepared TURL.
j) If request is accepted, SRM assigns the *requestToken* for asynchronous status checking. In such case, the returned status code should be SRM_REQUEST_QUEUED.
k) *totalRequestTime* means: All the file transfer for this request must be complete within this *totalRequestTime*. Otherwise, SRM_REQUEST_TIMED_OUT must be returned as the request status code with individual file status of SRM_FAILURE with an appropriate explanation.
l) If *desiredTotalRequestTime* is unspecified as NULL, the request will be retried for a duration which is dependent on the SRM implementation.
m) If input parameter *desiredTotalRequestTime* is 0 (zero), each file request will be tried at least once. Negative value is invalid.
n) Output parameter *remainingTotalRequestTime* indicates how long the *desiredTotalRequestTime* is left. If *remainingTotalRequestTime* is 0 (zero), the request has been timed out. If *remainingTotalRequestTime* is a negative value (-1), it would mean that each file request will be tried at least once.
o) The invocation of *srmReleaseFile*() is expected for finished files later on.
p) The returned request token should be valid until all files in the request are released or removed.
q) Streaming mode is allowed. If streaming mode is supported and there is not enough space to hold the request or partially hold the request, the SRM server returns SRM_REQUEST_QUEUED and keeps trying the request for the duration of *desiredTotalRequestTime*. In the output parameter of explanation in *returnStatus*, the server may make explicit that the retry is being done. If streaming mode is not supported, the server returns SRM_NO_USER_SPACE or

SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests were successful) or SRM_FAILURE at the request level.

r) Zero length files must not fail on srmPrepareToGet.

5.1.3. Return Status Code

For request level return status,

SRM_REQUEST_QUEUED
- successful request submission and acceptance. All file requests are on the queue. Request token must be returned.

SRM_REQUEST_INPROGRESS
- some files are completed, and some files are still on the queue. Request token must be returned.

SRM_SUCCESS
- all file requests are successfully completed. All *SURLs* are successfully pinned. For *TURLs*, file level status needs to be checked.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some file request is successfully pinned, and some file request is failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to submit the request

SRM_INVALID_REQUEST
- *arrayOfFileRequest* is empty
- If both input parameters *targetSpaceToken* and *TRetentionPolicyInfo* are provided, then their types must match exactly.
- Access latency is something other than ONLINE.
- *targetSpaceToken* does not refer to an existing known space in the SRM server.

SRM_SPACE_LIFETIME_EXPIRED
- space associated with the *targetSpaceToken* is expired.

SRM_EXCEED_ALLOCATION
- space associated with the *targetSpaceToken* is not enough to hold all requested *SURLs*.

SRM_NO_USER_SPACE
- user space is not enough to hold all requested *SURLs*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold all requested *SURLs* for free. When client does not specify the *targetSpaceToken*, SRM uses a default space. The default space is not sufficient to accommodate the request.

SRM_NOT_SUPPORTED
- SRM server does not support the given input parameters. For example, client requested bbftp for the only transfer protocol, but SRM cannot support that. Client requested *desiredFileStorageType* that is not supported by the SRM server.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM server.
- Directory operation is not supported in the SRM server.
- Recursive directory operation is not supported in the SRM server.
- None of the file transfer protocols are supported in the SRM server.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_FILE_PINNED
- successful request completion for the *SURL. SURL* is successfully pinned, and *TURL* is available for access.

SRM_REQUEST_QUEUED
- file request is on the queue.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_ABORTED
- The requested file has been aborted.

SRM_RELEASED
- The requested file has been released.

SRM_FILE_LOST
- the requested file is permanently lost.

SRM_FILE_BUSY
- client requests for files which there is an active *srmPrepareToPut* (no *srmPutDone* is not yet called) for.

SRM_FILE_UNAVAILABLE
- the requested file is temporarily unavailable.

SRM_INVALID_PATH
- *SURL* does not refer to an existing known file request that is associated with the request token

SRM_AUTHORIZATION_FAILURE
- client is not authorized to retrieve the file that is associated with the *SURL*

SRM_FILE_LIFETIME_EXPIRED
- *SURL* is expired
- *TUR*L is expired
- pin lifetime on TURL has expired, but the file is still in the cache.

SRM_NO_USER_SPACE
- user space is not enough to hold requested *SURL*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold requested *SURL* for free. When client does not specify the *targetSpaceToken*, SRM uses a default space. The default space is not sufficient to accommodate the request.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.
- The file request would not be able to be completed within the *totalRequestTime*.
- The requested file has been suspended because the request has timed out.

5.2. srmStatusOfGetRequest

This function is used to check the status of the previously requested srmPrepareToGet. Request token from srmPrepareToGet must be provided.

5.2.1. Parameters

In:	string	requestToken,
	string	authorizationID
	anyURI []	arrayOfSourceSURLs,
Out:	TReturnStatus	returnStatus,
	TGetRequestFileStatus[]	arrayOfFileStatuses
	int	remainingTotalRequestTime

5.2.2. Notes on the Behavior

a) The default value of "lifetime" for Volatile or Durable files will be the lifetime left in the space of the corresponding file type. The default value of "fileStorageType" is Volatile.
b) If *arrayOfSourceSURLs* is not provided, SRM must return status for all file requests in the request that is associated with the request token.
c) When the file is ready and TURL is prepared, the return status code should be SRM_FILE_PINNED.
d) When the file is ready for the client, the file is implicitly pinned in the cache and lifetime will be enforced, subject to the policies associated with the underlying storage.
e) If any of the request files is temporarily unavailable, SRM_FILE_UNAVAILABLE must be returned for the file.
f) If any of the request files is permanently lost, SRM_FILE_LOST must be returned for the file.
g) The file request must fail with an error SRM_FILE_BUSY if srmPrepareToGet requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.
h) SRM must fail (SRM_FAILURE) only if all files in the request failed.
i) *totalRequestTime* means: All the file transfer for this request must be complete within this *totalRequestTime*. Otherwise, SRM_REQUEST_TIMED_OUT must be returned as the request status code with individual file status of SRM_FAILURE with an appropriate explanation.
j) Output parameter *remainingTotalRequestTime* indicates how long the *desiredTotalRequestTime* is left. If *remainingTotalRequestTime* is 0 (zero), the request has been timed out. If *remainingTotalRequestTime* is a negative value (-1), it would mean that each file request will be tried at least once.
k) Streaming mode is allowed. If streaming mode is supported and there is not enough space to hold the request or partially hold the request, the SRM server returns SRM_REQUEST_QUEUED and keeps trying the request for the duration of *desiredTotalRequestTime* from the request. *remainingTotalRequestTime* is being returned. In the output parameter of explanation in *returnStatus*, the server may make explicit that the retry is being done. If streaming mode is not supported, the server returns SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests were successful) or SRM_FAILURE at the request level. Clients may need to release files or clean up the target space when target space token was provided.
l) Output parameter *returnStatus* must always refer to the request status of the whole request, even if a subset of the whole request was specified in the input for specific file statuses.

5.2.3. Return Status Code

For request level return status,

SRM_SUCCESS
- all file requests are successfully completed. All *SURLs* are successfully pinned. For *TURLs*, file level status needs to be checked.

SRM_REQUEST_QUEUED
- successful request submission and all files request is still on the queue

SRM_REQUEST_INPROGRESS
- some files are completed, and some files are still on the queue

SRM_PARTIAL_SUCCESS
- All requests are completed. Some file request is successfully pinned, and some file request is failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to submit the request

SRM_INVALID_REQUEST
- *requestToken* does not refer to an existing known request in the SRM server.

SRM_SPACE_LIFETIME_EXPIRED
- space associated with the *targetSpaceToken* is expired.

SRM_EXCEED_ALLOCATION
- space associated with the *targetSpaceToken* is not enough to hold all requested SURLs.

SRM_NO_USER_SPACE
- user space is not enough to hold all requested *SURLs*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold all requested *SURLs* for free.

SRM_NOT_SUPPORTED
- SRM server does not support the given input parameters. For example, client requested bbftp for the only transfer protocol, but SRM cannot support that. Client requested *desiredFileStorageType* that is not supported by the SRM server.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM server.
- Directory operation is not supported in the SRM server.
- Recursive directory operation is not supported in the SRM server.
- None of the file transfer protocols are supported in the SRM server.

SRM_ABORTED
- The request has been aborted.

SRM_REQUEST_TIMED_OUT
- Total request time is over and the rest of the request is failed.

SRM_REQUEST_SUSPENDED
- request is suspended.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_FILE_PINNED
- successful request completion for the *SURL. SURL* is successfully pinned, and *TURL* is available for access.

SRM_REQUEST_QUEUED
- file request is on the queue.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_ABORTED
- The requested file has been aborted.

SRM_RELEASED
- The requested file has been released.

SRM_REQUEST_SUSPENDED
- File request is suspended.

SRM_FILE_LOST
- the requested file is permanently lost.

SRM_FILE_BUSY
- client requests for files which there is an active *srmPrepareToPut* (no *srmPutDone* is not yet called) for.

SRM_FILE_UNAVAILABLE
- the requested file is temporarily unavailable.

SRM_INVALID_PATH
- *SURL* does not refer to an existing known file request that is associated with the request token

SRM_AUTHORIZATION_FAILURE
- client is not authorized to retrieve the file that is associated with the *SURL*

SRM_FILE_LIFETIME_EXPIRED
- *SURL* is expired
- *TUR*L is expired
- pin lifetime on TURL has expired, but the file is still in the cache

SRM_NO_USER_SPACE
- user space is not enough to hold requested *SURL*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold requested *SURL* for free. When client does not specify the *targetSpaceToken*, SRM uses a default space. The default space is not sufficient to accommodate the request.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.
- The file request would not be able to be completed within the *totalRequestTime*.
- The requested file has been suspended because the request has timed out.

5.3. srmBringOnline

This function is used to bring files online upon the client's request so that client can make certain data readily available for future access. In hierarchical storage systems, it is expected to "stage" files to the top hierarchy and make sure that the files stay online for a certain period of time. When client specifies target space token which must be referred to an online space, the files will be brought online using the space associated with the space token. It is an asynchronous operation, and request token must be

returned if asynchronous operation is necessary in SRM. The status may be checked through *srmStatusOfBringOnlineRequest* with the returned request token.
This function is similar to srmPrepareToGet, but it does not return Transfer URL (TURL).

5.3.1. Parameters

In:	string	authorizationID,
	TGetFileRequest[]	arrayOfFileRequests,
	string	userRequestDescription,
	TExtraInfo[]	storageSystemInfo,
	TFileStorageType	desiredFileStorageType
	int	desiredTotalRequestTime
	int	desiredLifetime, // life time on online
	string	targetSpaceToken,
	TRetentionPolicyInfo	targetFileRetentionPolicyInfo,
	TTransferParameters	transferParameters,
	int	deferredStartTime
Out:	TReturnStatus	returnStatus
	string	requestToken
	TBringOnlineRequestFileStatus[]	arrayOfFileStatuses
	int	remainingTotalRequestTime
	int	remainingDeferredStartTime

5.3.2. Notes on the Behavior

a) Input parameter *deferredStartTime* is to support CE-SE resource co-allocation and tape mounting efficiency. It means that client does not intent to use the files before that time. If SRM decides not to bring any files until *deferredStartTime* is reached, SRM_REQUEST_QUEUED must be returned. By default *deferredStartTime* is 0 (zero) and the request gets queued or processed upon submission. Negative value is invalid.
b) Output parameter *remainingDeferredStartTime* indicates how long the deferredStartTime is left, if supported. Negative value is not valid.
c) Input parameter *targetFileRetentionPolicyInfo* of *TRetentionPolicyInfo* is to specify the desired retention policy information on the file when the file is brought online.
d) If both input parameters *targetSpaceToken* and *TRetentionPolicyInfo* are provided, then their types must match exactly. Otherwise, the request may be rejected, and SRM_INVALID_REQUEST will be returned.
e) Optional input parameters in *TTransferParameters* may collide with the characteristics of the space specified. In this case, *TTransferParameters* as an input parameter must be ignored.
f) If the transfer protocol hints are not specified, default is assumed to be processing mode and LAN access for the site.
g) Access latency must be ONLINE always.
h) It is up to the SRM implementation to decide TConnectionType if not provided.
i) The *userRequestDescription* is a user designated name for the request. It is case-sensitive. SRM server is expected to keep it as client provides. It can be reused by the client. It can be used in the *srmGetRequestTokens* function to get back the system assigned request tokens. *srmGetRequestTokens* will return all the request tokens that have the *userRequestDescription.*

j) Input parameter *desiredLifetime* is for a client preferred lifetime (expiration time) on the file "copies (or "states") of the SURLs that will be "brought online" into the target space that is associated with the *targetSpaceToken*.
k) This call may be an asynchronous (non-blocking) call, and SRM assigns the *requestToken* when the request is valid and accepted. The returned status code should be SRM_REQUEST_QUEUED. To get subsequent status and results, separate calls should be made through srmStatusOfBringOnline.
l) The returned request token should be valid until all files in the request are released, removed or aborted.
m) *totalRequestTime* means: All the file transfer for this request must be complete within this *desiredTotalRequestTime*. Otherwise, SRM_REQUEST_TIMED_OUT must be returned as the request status code with individual file status of SRM_FAILURE with an appropriate explanation.
n) If input parameter *desiredTotalRequestTime* is unspecified as NULL, the request will be retried for a duration which is dependent on the SRM implementation.
o) If input parameter *desiredTotalRequestTime* is 0 (zero), each file request will be tried at least once. Negative value is not valid.
p) Output parameter *remainingTotalRequestTime* indicates how long the *desiredTotalRequestTime* is left. If *remainingTotalRequestTime* is 0 (zero), the request has been timed out. If *remainingTotalRequestTime* is a negative value (-1), it would mean that each file request will be tried at least once.
q) When *srmAbortRequest* is requested for *srmBringOnline* request, the request gets aborted, but those files that are brought online will remain in the space where they are brought in, and are not removed. Clients need to remove those files through *srmPurgeFromSpace* or *srmRm*.
r) Streaming mode is allowed. If streaming mode is supported and there is not enough space to hold the request or partially hold the request, the SRM server returns SRM_REQUEST_QUEUED and keeps trying the request for the duration of *desiredTotalRequestTime* from the request. In the output parameter of explanation in *returnStatus*, the server may make explicit that the retry is being done. If streaming mode is not supported, the server returns SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests were successful) or SRM_FAILURE at the request level.

5.3.3. Return Status Code

For request level return status,

SRM_REQUEST_QUEUED
- successful request submission and acceptance. All file requests are on the queue. Request token must be returned.

SRM_REQUEST_INPROGRESS
- some files are completed, and some files are not completed yet. Request token must be returned.

SRM_SUCCESS
- All requests are successfully completed. All *SURLs* are successfully brought online.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some files are successfully brought online, and some files are failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to submit the request

SRM_INVALID_REQUEST
- *arrayOfFileRequest* is empty
- Access latency refers to something other than ONLINE.
- If both input parameters *targetSpaceToken* and *TRetentionPolicyInfo* are provided, then their types must match exactly.
- *targetSpaceToken* does not refer to an existing known space in the SRM server.
- *deferredStartTime* is negative.

SRM_SPACE_LIFETIME_EXPIRED
- space associated with the *targetSpaceToken* is expired.

SRM_EXCEED_ALLOCATION
- space associated with the *targetSpaceToken* is not enough to hold all requested SURLs.

SRM_NO_USER_SPACE
- user space is not enough to hold all requested *SURLs*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold all requested *SURLs* for free.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- SRM server does not support the given input parameters. For example, client requested bbftp for the only transfer protocol, but SRM cannot support that. Client requested *desiredFileStorageType* that is not supported by the SRM server.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM server.
- *deferredStartTime* is not supported in the SRM server.
- Directory operation is not supported in the SRM server.
- Recursive directory operation is not supported in the SRM server.
- None of the file transfer protocols are supported in the SRM server.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL. SURL* is successfully brought online.

SRM_REQUEST_QUEUED
- file request is on the queue.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_AUTHORIZATION_FAILURE
- client is not authorized to retrieve the file that is associated with the *SURL*

SRM_ABORTED
- The requested file has been aborted.

SRM_RELEASED
- The requested file has been released.

SRM_FILE_LOST
- the requested file is permanently lost.

SRM_FILE_BUSY

- client requests for files which there is an active *srmPrepareToPut* (no *srmPutDone* is not yet called) for.

SRM_FILE_UNAVAILABLE
- the requested file is temporarily unavailable.

SRM_INVALID_PATH
- *SURL* does not refer to an existing known file request that is associated with the request token

SRM_FILE_LIFETIME_EXPIRED
- *SURL* is expired
- pin lifetime has expired, but the file is still in the cache

SRM_NO_USER_SPACE
- user space is not enough to hold requested *SURL*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold requested *SURL* for free. When client does not specify the *targetSpaceToken*, SRM uses a default space. The default space is not sufficient to accommodate the request.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.
- The file request would not be able to be completed within the *totalRequestTime*.
- The requested file has been suspended because the request has timed out.

5.4. srmStatusOfBringOnlineRequest

This function is used to check the status of the previous request to *srmBringOnline,* when asynchronous operation is necessary in the SRM. Request token must have been provided in response to the *srmBringOnline*.

5.4.1. Parameters

In:	string	requestToken,
	string	authorizationID
	anyURI []	arrayOfSourceSURLs,
Out:	TReturnStatus	returnStatus,
	TBringOnlineRequestFileStatus[]	arrayOfFileStatuses
	int	remainingTotalRequestTime
	int	remainingDeferredStartTime

5.4.2. Notes on the Behavior

a) If *arrayOfSourceSURLs* is not provided, returns status for all files in this request.
b) When the file is ready online, the return status code should be SRM_FILE_IN_CACHE.
c) Output parameter *remainingDeferredStartTime* indicates how long the deferredStartTime is left, if supported. Negative value is not valid.
d) When the file is ready for the client, the file is implicitly pinned in the cache and lifetime will be enforced, subject to the policies associated with the underlying storage.
e) If any of the request files is temporarily unavailable, SRM_FILE_UNAVAILABLE must be returned for the file.

f) If any of the request files is permanently lost, SRM_FILE_LOST must be returned for the file.
g) The file request must fail with an error SRM_FILE_BUSY if srmBringOnline requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.
h) SRM must fail (SRM_FAILURE) only if all files in the request failed.
i) *totalRequestTime* means: All the file transfer for this request must be complete within this *totalRequestTime*. Otherwise, SRM_REQUEST_TIMED_OUT must be returned as the request status code with individual file status of SRM_FAILURE with an appropriate explanation.
j) Output parameter *remainingTotalRequestTime* indicates how long the *desiredTotalRequestTime* is left. If *remainingTotalRequestTime* is 0 (zero), the request has been timed out. If *remainingTotalRequestTime* is a negative value (-1), it would mean that each file request will be tried at least once.
k) If SRM decides not to bring any files until input parameter *deferredStartTime* is reached, SRM_REQUEST_QUEUED must be returned.
l) Streaming mode is allowed. If streaming mode is supported and there is not enough space to hold the request or partially hold the request, the SRM server returns SRM_REQUEST_QUEUED and keeps trying the request for the duration of *desiredTotalRequestTime* from the request. *remainingTotalRequestTime* is being returned. In the output parameter of explanation in *returnStatus*, the server may make explicit that the retry is being done. If streaming mode is not supported, the server returns SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests were successful) or SRM_FAILURE at the request level. Clients may need to release files or clean up the target space when target space token was provided.
m) Output parameter *returnStatus* must always refer to the request status of the whole request, even if a subset of the whole request was specified in the input for specific file statuses.

5.4.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. All *SURLs* are successfully brought online.

SRM_REQUEST_QUEUED
- successful request submission and all files request is on the queue

SRM_REQUEST_INPROGRESS
- some files are completed, and some files are not completed yet.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some files are successfully brought online, and some files are failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to submit the request

SRM_INVALID_REQUEST
- *requestToken* does not refer to an existing known request in the SRM server.

SRM_NOT_SUPPORTED
- SRM server does not support the given input parameters. For example, client requested bbftp for the only transfer protocol, but SRM cannot support that. Client requested *desiredFileStorageType* that is not supported by the SRM server.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM server.

- *deferredStartTime* is not supported in the SRM server.
- Directory operation is not supported in the SRM server.
- Recursive directory operation is not supported in the SRM server.
- None of the file transfer protocols are supported in the SRM server.

SRM_SPACE_LIFETIME_EXPIRED
- space associated with the *targetSpaceToken* is expired.

SRM_EXCEED_ALLOCATION
- space associated with the *targetSpaceToken* is not enough to hold all requested SURLs.

SRM_NO_USER_SPACE
- user space is not enough to hold all requested SURLs.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold all requested SURLs for free.

SRM_ABORTED
- The request has been aborted.

SRM_REQUEST_TIMED_OUT
- Total request time is over and the rest of the request is failed.

SRM_REQUEST_SUSPENDED
- request is suspended.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL. SURL* is successfully brought online.

SRM_REQUEST_QUEUED
- file request is on the queue.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_AUTHORIZATION_FAILURE
- client is not authorized to retrieve the file that is associated with the *SURL*

SRM_ABORTED
- The requested file has been aborted.

SRM_RELEASED
- The requested file has been released.

SRM_REQUEST_SUSPENDED
- File request is suspended.

SRM_FILE_LOST
- the requested file is permanently lost.

SRM_FILE_BUSY
- client requests for files which there is an active *srmPrepareToPut* (no *srmPutDone* is not yet called) for.

SRM_FILE_UNAVAILABLE
- the requested file is temporarily unavailable.

SRM_INVALID_PATH

- *SURL* does not refer to an existing known file request that is associated with the request token

SRM_FILE_LIFETIME_EXPIRED
- *SURL* is expired
- pin lifetime has expired, but the file is still in the cache

SRM_NO_USER_SPACE
- user space is not enough to hold requested *SURL*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold requested *SURL* for free. When client does not specify the *targetSpaceToken*, SRM uses a default space. The default space is not sufficient to accommodate the request.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.
- The file request would not be able to be completed within the *totalRequestTime*.
- The requested file has been suspended because the request has timed out.

5.5. srmPrepareToPut

This function is used to write files into the storage. Upon the client's request, SRM prepares a TURL so that client can write data into the TURL. Lifetime (pinning expiration time) is assigned on the TURL. When a specified target space token is provided, the files will be located finally in the targeted space associated with the target space token. It is an asynchronous operation, and request token must be returned if the request is valid and accepted to the SRM. The status may be checked through srmStatusOfPutRequest with the returned request token.

5.5.1. Parameters

In:	string	authorizationID,
	TPutFileRequest[]	arrayOfFileRequests,
	string	userRequestDescription,
	TOverwriteMode	overwriteOption,
	TExtraInfo[]	storageSystemInfo,
	int	desiredTotalRequestTime
	int	desiredPinLifetime, // on TURL
	int	desiredFileLifetime, // on SURL
	TFileStorageType	desiredFileStorageType,
	string	targetSpaceToken
	TRetentionPolicyInfo	targetFileRetentionPolicyInfo
	TTransferParameters	transferParameters
Out:	TReturnStatus	returnStatus
	string	requestToken,
	TPutRequestFileStatus[]	arrayOfFileStatuses
	int	remainingTotalRequestTime

5.5.2. Notes on the Behavior

a) The default value of "lifetime" for Volatile or Durable files will be the lifetime left in the space of the corresponding file type. The default value of "fileStorageType" is Volatile.
b) TURL returned by the srmPrepareToPut may not be used for read access with any protocol. An explicit srmPrepareToGet or srmBringOnline is required.
c) Optional input parameters in *TTransferParameters* may collide with the characteristics of the space specified. In this case, *TTransferParameters* as an input parameter must be ignored.
d) Input parameter *userRequestDescription* may be null, and it is case-sensitive when provided. SRM server is expected to keep it as client provides. It can be reused by the client. It can be used in the *srmGetRequestTokens* function to get back the system assigned request tokens. *srmGetRequestTokens* will return all the request tokens that have the *userRequestDescription.*
e) Input parameter *targetSpaceToken* is provided at the request-level, and all files in the request will end up in the space that is associated with the target space token if the space is enough for all files.
f) Input parameter *targetFileRetentionPolicyInfo* of *TRetentionPolicyInfo* is to specify the desired retention policy information on the file when the file is written into the target storage system.
g) If both input parameters *targetSpaceToken* and *TRetentionPolicyInfo* are provided, then their types must match exactly. Otherwise, the request may be rejected and SRM_INVALID_REQUEST must be returned.
h) Only push mode is supported for file transfers that client must "push" the file to the prepared TURL.
i) Input parameter *targetSURL* in the *TPutFileRequest* has to be local to SRM. If *targetSURL* is not specified, SRM will generate a reference SURL for the file request automatically and put it in the specified user space if provided. This reference SURL will be returned along with the "Transfer URL". Some SRM implementation may require *targetSURL*.
j) *srmPutDone*() is expected after each file is "put" into the prepared TURL.
k) Input parameter *desiredPinLifetime* is the lifetime (expiration time) on the TURL when the Transfer URL is prepared. It does not refer to the lifetime of the SURL. TURLs will not be valid any more after the *desiredPinLifetime* is over if *srmPutDone* or *srmAbortRequest* is not submitted on the SURL before expiration. In such case, the server returns SRM_FAILURE at the file level.
l) Input parameter *desiredFileLifetime* is the lifetime of the SURL when the file is put into the storage system. It does not refer to the lifetime (expiration time) of the TURL. Lifetime on SURL starts when successrul srmPutDone is executed.
m) The lifetime of the SURL starts as soon as SRM receives the *srmPutDone*(). If *srmPutDone*() is not provided, then the files in that space are subject to removal when the lifetime on the TURL expires or the lifetime on the space expires. The lifetime on the TURL can be found in the status of the file request as output parameter *remainingPinTime* in *TPutRequestFileStatus*.
n) If request is accepted, SRM assigns the *requestToken* for asynchronous status checking. In such case, the returned status code should be SRM_REQUEST_QUEUED.
o) *totalRequestTime* means: All the file transfer for this request must be complete within this *totalRequestTime*. Otherwise, SRM_REQUEST_TIMED_OUT must be returned as the request status code with individual file status of SRM_FAILURE with an appropriate explanation.
p) If input parameter *desiredTotalRequestTime* is unspecified as NULL, the request will be retried for a duration which is dependent on the SRM implementation.
q) If input parameter *desiredTotalRequestTime* is 0 (zero), each file request will be tried at least once. Negative value is invalid.
r) Output parameter *remainingTotalRequestTime* indicates how long the *desiredTotalRequestTime* is left. If *remainingTotalRequestTime* is 0 (zero), the request has been timed out. If

remainingTotalRequestTime is a negative value (-1), it would mean that each file request will be tried at least once.

s) Streaming mode is allowed. If streaming mode is supported and there is not enough space to hold the request or partially hold the request, the SRM server returns SRM_REQUEST_QUEUED and keeps trying the request for the duration of *desiredTotalRequestTime* from the request. In the output parameter of explanation in *returnStatus*, the server may make explicit that the retry is being done. If streaming mode is not supported, the server returns SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests were successful) or SRM_FAILURE at the request level.

t) Upon srmPrepareToPut, SURL entry is inserted to the name space, and any methods that access the SURL such as srmLs, srmBringOnline and srmPrepareToGet must return SRM_FILE_BUSY at the file level. If another srmPrepareToPut or srmCopy were requested on the same SURL, SRM_FILE_BUSY must be returned if the SURL can be overwritten, otherwise SRM_DUPLICATION_ERROR must be returned at the file level.

u) Input parameter *overwriteOption* is assumed to be NEVER when not specified.

v) When requested file storage type is VOLATILE, it cannot be promoted to PERMANENT to avoid complexities in space accounting and other cleanup tasks. SRM_NOT_SUPPORTED must be returned if the requested file storage type is not supported, or the request must be processed.

w) After TURL is returned, *srmMv* operation on the corresponding SURL may be requested. *srmPutDone* on the original SURL will succeed, and SRM_SUCCESS must be returned at the file level upon successful srmPutDone.

x) Zero length files must not fail on srmPrepareToPut.

y) When a VOLATILE file is put into an unreserved replica quality space without any space token being used, and the VOLATILE file gets expired, SRM must remove its SURL from the file system. The file may or may not be removed physically right away.

5.5.3. Return Status Code

For request level return status,

SRM_REQUEST_QUEUED
- successful request submission and acceptance. All file requests are on the queue. Request token must be returned.

SRM_REQUEST_INPROGRESS
- some files are completed, and some files are still on the queue. Request token must be returned.

SRM_SUCCESS
- All requests are successfully completed. For all *SURLs*, spaces are allocated, and *TURLs* are prepared.

SRM_PARTIAL_SUCCESS
- All requests are completed. For some file requests, the spaces are allocated and *TURLs* are prepared, but for some file requests, it is failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to submit the request

SRM_INVALID_REQUEST
- If both input parameters *targetSpaceToken* and *TRetentionPolicyInfo* are provided, then their types must match exactly.

- *targetSpaceToken* does not refer to an existing known space in the SRM server.

SRM_SPACE_LIFETIME_EXPIRED
- space associated with the *targetSpaceToken* is expired.

SRM_EXCEED_ALLOCATION
- space associated with the *targetSpaceToken* is not enough to hold all requested SURLs.

SRM_NO_USER_SPACE
- user space is not enough to hold all requested *SURLs*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold all requested *SURLs* for free.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- SRM server does not support the given input parameters. For example, client requested bbftp for the only transfer protocol, but SRM cannot support that. Client requested *desiredFileStorageType* that is not supported by the SRM server.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM server.
- None of the file transfer protocols are supported in the SRM server.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_SPACE_AVAILABLE
- successful request completion for the *"put"* request. The space is allocated, and *TURL* is prepared.

SRM_REQUEST_QUEUED
- file request is on the queue.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_FILE_IN_CACHE
- lifetime on *SURL* has expired, but the file is still in the cache.

SRM_INVALID_PATH
- *targetSURL* does not refer to a valid path.

SRM_DUPLICATION_ERROR
- *targetSURL* refers to an existing SURL and overwriting is not allowed.

SRM_FILE_BUSY
- client requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) or srmCopy for.

SRM_AUTHORIZATION_FAILURE
- client is not authorized to retrieve the file that is associated with the *SURL*

SRM_ABORTED
- The requested file has been aborted.

SRM_NO_USER_SPACE
- user space is not enough to hold the requested *SURL*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold the requested *SURL* for free.

SRM_FAILURE

- any other request failure. *Explanation* needs to be filled for details.
- The file request would not be able to be completed within the *totalRequestTime*.
- The requested file has been suspended because the request has timed out.
- The file request is not aborted or completed by srmPutDone, and the TURL (available space allocation for the file) is not valid any more.

5.6. srmStatusOfPutRequest

This function is used to check the status of the previously requested srmPrepareToPut. Request token from srmPrepareToPut must be provided.

5.6.1. Parameters

In:	string	requestToken,
	string	authorizationID
	anyURI []	arrayOfTargetSURLs,
Out:	TReturnStatus	returnStatus,
	TPutRequestFileStatus[]	arrayOfFileStatuses
	int	remainingTotalRequestTime

5.6.2. Notes on the Behavior

a) The default value of "lifetime" for Volatile or Durable files will be the lifetime left in the space of the corresponding file type. The default value of "fileStorageType" is Volatile.
b) If *arrayOfTargetSURLs* is not provided, returns status for all the file requests in this request.
c) When the space is ready for client to "put" data and TURL is prepared, the return status code should be SRM_SPACE_AVAILABLE.
d) When the file space is ready for the client, the TURL is available in the cache and pin lifetime on the TURL will be enforced. TURLs will not be valid any more after the pin lifetime is over if *srmPutDone* or *srmAbortRequest* is not submitted on the SURL before expiration. In such case, the server returns SRM_FAILURE at the file level.
e) If a targetSURL is provided with some directory structure, the directory structure must exist, and SRM will not create the directory structure for the targetSURL. In such case, SRM_INVALID_PATH must be returned. srmMkdir may be used to create the directory structure.
f) Lifetime on SURL starts when successrul srmPutDone is executed.
g) If the space for the requested files is full, and TURL cannot be returned, then SRM_EXCEED_ALLOCATION, SRM_NO_USER_SPACE, or SRM_NO_FREE_SPACE must be returned for the files.
h) SRM must fail (SRM_FAILURE) only if all files in the request failed.
i) *totalRequestTime* means: All the file transfer for this request must be complete within this *totalRequestTime*. Otherwise, SRM_REQUEST_TIMED_OUT must be returned as the request status code with individual file status of SRM_FAILURE with an appropriate explanation.
j) Output parameter *remainingTotalRequestTime* indicates how long the *desiredTotalRequestTime* is left. If *remainingTotalRequestTime* is 0 (zero), the request has been timed out. If *remainingTotalRequestTime* is a negative value (-1), it would mean that each file request will be tried at least once.

k) Streaming mode is allowed. If streaming mode is supported and there is not enough space to hold the request or partially hold the request, the SRM server returns SRM_REQUEST_QUEUED and keeps trying the request for the duration of *desiredTotalRequestTime* from the request. *remainingTotalRequestTime* is being returned. In the output parameter of explanation in *returnStatus*, the server may make explicit that the retry is being done. If streaming mode is not supported, the server returns SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests were successful) or SRM_FAILURE at the request level. Clients may need to clean up the target space when target space token was provided.

l) Upon srmPrepareToPut, SURL entry is inserted to the name space, and any methods that access the SURL such as srmLs, srmBringOnline and srmPrepareToGet must return SRM_FILE_BUSY at the file level. If another srmPrepareToPut or srmCopy were requested on the same SURL, SRM_FILE_BUSY must be returned if the SURL can be overwritten, otherwise SRM_DUPLICATION_ERROR must be returned at the file level.

m) *srmRm* may remove SURLs even if the statuses of the SURLs are SRM_FILE_BUSY. In this case, the status for srmPrepareToPut request must return SRM_INVALID_PATH upon status request or srmPutDone.

n) After TURL is returned, *srmMv* operation on the corresponding SURL may be requested. *srmPutDone* on the original SURL will succeed, and SRM_SUCCESS must be returned at the file level upon successful srmPutDone.

o) Output parameter *returnStatus* must always refer to the request status of the whole request, even if a subset of the whole request was specified in the input for specific file statuses.

5.6.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. For all *SURLs*, spaces are allocated, and *TURLs* are prepared.

SRM_REQUEST_QUEUED
- successful request submission and all files request is still on the queue

SRM_REQUEST_INPROGRESS
- some files are completed, and some files are still on the queue

SRM_PARTIAL_SUCCESS
- All requests are completed. For some file requests, the spaces are allocated and *TURLs* are prepared, but for some file requests, it is failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to submit the request

SRM_INVALID_REQUEST
- *requestToken* does not refer to an existing known request in the SRM server.
- *targetSpaceToken* that client provided does not refer to an existing space in the SRM server.

SRM_SPACE_LIFETIME_EXPIRED
- space associated with the *targetSpaceToken* is expired.

SRM_EXCEED_ALLOCATION

- space associated with the *targetSpaceToken* is not enough to hold all requested *SURLs*.

SRM_NO_USER_SPACE
- user space is not enough to hold all requested *SURLs*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold all requested *SURLs* for free.

SRM_REQUEST_TIMED_OUT
- Total request time is over and the rest of the request is failed.

SRM_ABORTED
- The request has been aborted.

SRM_REQUEST_SUSPENDED
- The request is suspended.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- SRM server does not support the given input parameters. For example, client requested bbftp for the only transfer protocol, but SRM cannot support that. Client requested *desiredFileStorageType* that is not supported by the SRM server.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM server.
- None of the file transfer protocols are supported in the SRM server.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_SPACE_AVAILABLE
- successful request completion for the *"put"* request. The space is allocated, and *TURL* is prepared.

SRM_REQUEST_QUEUED
- file request is on the queue.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_SUCCESS
- Client's file transfer into *TURL* is completed, and *srmPutDone* on the *targetSURL* is completed. The file is now in the cache and lifetime on the *targetSURL* is started.

SRM_FILE_IN_CACHE
- lifetime on *SURL* has expired, but the file is still in the cache.

SRM_INVALID_PATH
- *targetSURL* does not refer to a valid path.

SRM_DUPLICATION_ERROR
- *targetSURL* refers to an existing SURL and overwriting is not allowed.

SRM_FILE_BUSY
- client requests for files which there is an active srmPrepareToPut (no srmPutDone is not yet called) or srmCopy for.

SRM_AUTHORIZATION_FAILURE
- client is not authorized to retrieve the file that is associated with the *SURL*

SRM_ABORTED

- The requested file has been aborted.

SRM_REQUEST_SUSPENDED
- File request is suspended.

SRM_NO_USER_SPACE
- user space is not enough to hold the requested *SURL*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold the requested *SURL* for free.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.
- The file request would not be able to be completed within the *totalRequestTime*.
- The requested file has been suspended because the request has timed out.
- The file request is not aborted or completed by srmPutDone, and the TURL (available space allocation for the file) is not valid any more.

5.7. srmCopy

This function is used to copy files from source storage sites into the target storage sites. The source storage site or the target storage site needs to be the SRM itself that the client makes the *srmCopy* request. If both source and target are local to the SRM, it performed a local copy. There are two cases for remote copies: 1. Target SRM is where client makes a srmCopy request (PULL case), 2. Source SRM is where client makes a srmCopy request (PUSH case).

1. PULL case: Upon the client's *srmCopy* request, the target SRM makes a space at the target storage, and makes a request *srmPrepareToGet* to the source SRM. When TURL is ready at the source SRM, the target SR M transfers the file from the source TURL into the prepared target storage. After the file transfer completes, *srmReleaseFiles* is issued to the source SRM.
2. PUSH case: Upon the client's *srmCopy* request, the source SRM prepares a file to be transferred out to the target SRM, and makes a request *srmPrepareToPut* to the target SRM. When TURL is ready at the target SRM, the source SRM transfers the file from the prepared source into the prepared target TURL. After the file transfer completes, *srmPutDone* is issued to the target SRM.

When specified target space token is provided, the files will be located finally in the targeted space associated with the space token. It is an asynchronous operation, and request token must be returned. The status may be checked through srmStatusOfCopyRequest with the returned request token.

5.7.1. Parameters

In:	string	authorizationID,
	TCopyFileRequest[]	arrayOfFileRequests,
	string	userRequestDescription,
	TOverwriteMode	overwriteOption,
	int	desiredTotalRequestTime,
	int	desiredTargetSURLLifeTime,
	TFileStorageType	targetFileStorageType,
	string	targetSpaceToken,
	TRetentionPolicyInfo	targetFileRetentionPolicyInfo,
	TExtraInfo[]	sourceStorageSystemInfo,
	TExtraInfo[]	targetStorageSystemInfo

Out:	TReturnStatus	returnStatus,
	string	requestToken,
	TCopyRequestFileStatus[]	arrayOfFileStatuses,
	int	remainingTotalRequestTime

5.7.2. Notes on the Behavior

a) The default value of "lifetime" for Volatile or Durable files will be the lifetime left in the space of the corresponding file type. The default value of "fileType" is Volatile.
b) When aborted, target SURLs need to be provided.
c) Input parameter *userRequestDescription* may be null, and it is case-sensitive when provided. SRM server is expected to keep it as client provides. It can be reused by the client. It can be used in the *srmGetRequestTokens* function to get back the system assigned request tokens. *srmGetRequestTokens* will return all the request tokens that have the *userRequestDescription.*
d) Input parameter *targetSpaceToken* is provided at the request-level, and all files in the request will end up in the space that is associated with the target space token.
e) Input parameter *targetFileRetentionPolicyInfo* of *TRetentionPolicyInfo* is to specify the desired retention policy information on the file when the file is written into the target storage system.
f) If both input parameters *targetSpaceToken* and *TRetentionPolicyInfo* are provided, then their types must match exactly. Otherwise, the request may be rejected, and SRM_INVALID_REQUEST must be returned.
g) If request is accepted, SRM assigns the *requestToken* for asynchronous status checking. In such case, the returned status code should be SRM_REQUEST_QUEUED.
h) Pull mode: copy from remote location to the SRM. (e.g. from remote to MSS.)
i) Push mode: copy from the SRM to remote location.
j) Always release files through *srmReleaseFiles* from the source after copy is done, if source is an SRM and PULL mode was performed.
k) Always issue *srmPutDone* to the target after copy is done, if target is an SRM and PUSH mode was performed.
l) Note there is no protocol negotiation with the client for this request.
m) *totalRequestTime* means: if all the file transfer for this request must be complete in this *totalRequestTime*. Otherwise, the request is returned as failed at the end of the *totalRequestTime*, and SRM_REQUEST_TIMED_OUT must be returned as the request status code with individual file status of SRM_FAILURE with an appropriate explanation. All completed files must not be removed, but status of the files must be returned to the client.
n) If input parameter *desiredTotalRequestTime* is unspecified as NULL, the request will be retried for a duration which is dependent on the SRM implementation.
o) If input parameter *desiredTotalRequestTime* is 0 (zero), each file request will be tried at least once. Negative value is invalid.
p) Output parameter *remainingTotalRequestTime* indicates how long the *desiredTotalRequestTime* is left. If *remainingTotalRequestTime* is 0 (zero), the request has been timed out. If *remainingTotalRequestTime* is a negative value (-1), it would mean that each file request will be tried at least once.
q) When both sourceSURL and targetSURL are local, local copy must be performed.
r) Empty directories are copied as well.
s) If a targetSURL is provided with some directory structure, the directory structure must exist, and SRM will not create the directory structure for the targetSURL. In such case, SRM_INVALID_PATH must be returned. srmMkdir may be used to create the directory structure.

t) If the sourceSURL and targetSURL are provided as directories (copying directories) when SRM implementation supports, then all sub directories will be copied over from the source to the target, and complete sub-directory structure will be created only if *TDirOption* indicates them.
u) Streaming mode is allowed. If streaming mode is supported and there is not enough space to hold the request or partially hold the request, the SRM server returns SRM_REQUEST_QUEUED and keeps trying the request for the duration of *desiredTotalRequestTime* from the request. In the output parameter of explanation in *returnStatus*, the server may make explicit that the retry is being done. If streaming mode is not supported, the server returns SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests were successful) or SRM_FAILURE at the request level. Clients may need to clean up the target space when target space token was provided.
v) Upon srmCopy, SURL entry is inserted to the target name space, and any methods that access the target SURL such as srmLs, srmBringOnline and srmPrepareToGet must return SRM_FILE_BUSY at the file level. If another srmPrepareToPut or srmCopy were requested on the same target SURL, SRM_FILE_BUSY must be returned if the target SURL can be overwritten, otherwise SRM_DUPLICATION_ERROR must be returned at the file level.
w) Input parameter *overwriteOption* is assumed to be NEVER when not specified.

5.7.3. Return Status Code

For request level return status,

SRM_REQUEST_QUEUED
- successful request submission and acceptance. All file requests are on the queue. Request token must be returned.

SRM_REQUEST_INPROGRESS
- Some files are completed, and some files are still on the queue. Details are on the files status. Request token must be returned.

SRM_SUCCESS
- All requests are successfully completed. All source *SURLs* are copied into the target destination successfully.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some file request is successfully copied into the target destination, and some file request is failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to submit the request
- Client is not authorized to copy files into the space that client provided with *targetSpaceToken* or *targetFileRetentionPolicyInfo*

SRM_INVALID_REQUEST
- If both input parameters *targetSpaceToken* and *TRetentionPolicyInfo* are provided, then their types must match exactly.
- *targetSpaceToken* does not refer to an existing known space in the SRM server.

SRM_SPACE_LIFETIME_EXPIRED
- space associated with the *targetSpaceToken* is expired.

SRM_EXCEED_ALLOCATION
- space associated with the *targetSpaceToken* is not enough to hold all requested SURLs.

SRM_NO_USER_SPACE
- user space is not enough to hold all requested SURLs.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold all requested SURLs for free.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- SRM server does not support the given input parameters. For example, client requested *desiredFileStorageType* that is not supported by the SRM server.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM server.
- Directory operation is not supported in the SRM server.
- Recursive directory operation is not supported in the SRM server.
- any input parameter is not supported in the SRM server
- a particular type of an input parameter is not supported in the SRM server
- *function* is not supported in the SRM server

SRM_FAILURE
- all files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_SUCCESS
- successful request completion for the file. The source *SURL* is copied into the target destination *targetSURL* successfully, and lifetime on the *targetSURL* is started.

SRM_REQUEST_QUEUED
- file request is on the queue.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_FILE_LOST
- the request file (*sourceSURL*) is permanently lost.

SRM_FILE_BUSY
- client requests for files at the source (*sourceSURL*) which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.
- client requests for files at the target (*targetSURL*) which there is an active srmPrepareToPut (no srmPutDone is not yet called) or srmCopy for.

SRM_FILE_UNAVAILABLE
- the request file (*sourceSURL*) is temporarily unavailable.

SRM_FILE_LIFETIME_EXPIRED
- lifetime on *targetSURL* has expired, but the file is still in the cache.

SRM_INVALID_PATH
- *sourceSUR* does not exist
- *targetSURL* does not refer to a valid path.

SRM_DUPLICATION_ERROR
- *targetSURL* refers to an existing SURL and overwriting is not allowed.

SRM_AUTHORIZATION_FAILURE
- Client is not authorized to copy files from *sourceSURL*
- Client is not authorized to copy files into *targetSURL*

- Client is not authorized to copy files into the space that client provided with *targetSpaceToken* or *targetFileRetentionPolicyInfo*

SRM_ABORTED
- The requested file has been aborted.

SRM_RELEASED
- The requested file has been released.

SRM_NO_USER_SPACE
- user space is not enough to hold the requested *SURL*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold the requested *SURL* for free.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.
- The file request would not be able to be completed within the *totalRequestTime*.
- The requested file has been suspended because the request has timed out.

5.8. srmStatusOfCopyRequest

This function is used to check the status of the previously requested srmCopy. Request token from srmCopy must be provided.

5.8.1. Parameters

In:	string	requestToken,
	string	authorizationID,
	anyURI []	arrayOfSourceSURLs,
	anyURI []	arrayOfTargetSURLs,
Out:	TReturnStatus	returnStatus,
	TCopyRequestFileStatus[]	arrayOfFileStatuses,
	int	remainingTotalRequestTime

5.8.2. Notes on the Behavior

a) If *arrayOfSourceSURLs* and/or *arrayOfTargetSURLs* are not provided, return status for all file requests in the request.
b) If the target space for the requested files is full, then SRM_EXCEED_ALLOCATION, SRM_NO_USER_SPACE, or SRM_NO_FREE_SPACE must be returned.
c) SRM must fail (SRM_FAILURE) only if all files in the request failed.
d) *totalRequestTime* means: All the file transfer for this request must be complete within this *totalRequestTime*. Otherwise, SRM_REQUEST_TIMED_OUT must be returned as the request status code with individual file status of SRM_FAILURE with an appropriate explanation.
e) Output parameter *remainingTotalRequestTime* indicates how long the *desiredTotalRequestTime* is left. If *remainingTotalRequestTime* is 0 (zero), the request has been timed out. If *remainingTotalRequestTime* is a negative value (-1), it would mean that each file request will be tried at least once.
f) Streaming mode is allowed. If streaming mode is supported and there is not enough space to hold the request or partially hold the request, the SRM server returns SRM_REQUEST_QUEUED

and keeps trying the request for the duration of *desiredTotalRequestTime* from the request. *remainingTotalRequestTime* is being returned. In the output parameter of explanation in *returnStatus*, the server may make explicit that the retry is being done. If streaming mode is not supported, the server returns SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests were successful) or SRM_FAILURE at the request level. Clients may need to clean up the target space when target space token was provided.

g) Upon srmCopy, SURL entry is inserted to the target name space, and any methods that access the target SURL such as srmLs, srmBringOnline and srmPrepareToGet must return SRM_FILE_BUSY at the file level. If another srmPrepareToPut or srmCopy were requested on the same target SURL, SRM_FILE_BUSY must be returned if the target SURL can be overwritten, otherwise SRM_DUPLICATION_ERROR must be returned at the file level.

h) *srmRm* may remove SURLs even if the statuses of the SURLs are SRM_FILE_BUSY. In this case, the status for srmCopy request must return SRM_INVALID_PATH upon status request.

i) Output parameter *returnStatus* must always refer to the request status of the whole request, even if a subset of the whole request was specified in the input for specific file statuses.

5.8.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. All source *SURLs* are copied into the target destination successfully.

SRM_REQUEST_QUEUED
- successful request submission and all files request is still on the queue

SRM_REQUEST_INPROGRESS
- Some files are completed, and some files are still on the queue. Details are on the files status.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some file request is successfully copied into the target destination, and some file request is failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to submit the request

SRM_INVALID_REQUEST
- *requestToken* does not refer to an existing known request in the SRM server.
- *targetSpaceToken* does not refer to an existing known space in the SRM server.

SRM_TOO_MANY_RESULTS
- Request produced too many results that SRM server cannot handle, and *arrayOfSourceURLs and arrayOfTargetURLs* cannot fit the results to return.

SRM_REQUEST_TIMED_OUT
- Total request time is over and the rest of the request is failed.

SRM_REQUEST_SUSPENDED
- The request is suspended.

SRM_SPACE_LIFETIME_EXPIRED
- space associated with the *targetSpaceToken* is expired.

SRM_EXCEED_ALLOCATION

- space associated with the *targetSpaceToken* is not enough to hold all requested SURLs.

SRM_NO_USER_SPACE
- Insufficient space left in the space that is associated with spaceToken.

SRM_NO_FREE_SPACE
- When client does not specify the spaceToken, SRM uses a default space. The default space is insufficient to accommodate the request.

SRM_ABORTED
- The request has been aborted.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- SRM server does not support the given input parameters. For example, client requested bbftp for the only transfer protocol, but SRM cannot support that. Client requested *desiredFileStorageType* that is not supported by the SRM server.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM server.
- Overwrite option is not supported in the SRM server.
- Directory operation is not supported in the SRM server.
- Recursive directory operation is not supported in the SRM server.
- any input parameter is not supported in the SRM server
- a particular type of an input parameter is not supported in the SRM server
- *function* is not supported in the SRM server

SRM_FAILURE
- all files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

For file level return status,

SRM_SUCCESS
- successful request completion for the file. The source *SURL* is copied into the target destination *targetSURL* successfully, and lifetime on the *targetSURL* is started.

SRM_REQUEST_QUEUED
- file request is on the queue.

SRM_REQUEST_INPROGRESS
- file request is being served.

SRM_FILE_LOST
- the request file (*sourceSURL*) is permanently lost.

SRM_FILE_BUSY
- client requests for files at the source (*sourceSURL*) which there is an active srmPrepareToPut (no srmPutDone is not yet called) for.
- client requests for files at the target (*targetSURL*) which there is an active srmPrepareToPut (no srmPutDone is not yet called) or srmCopy for.

SRM_FILE_UNAVAILABLE
- the request file (*sourceSURL*) is temporarily unavailable.

SRM_FILE_LIFETIME_EXPIRED
- lifetime on *targetSURL* has expired, but the file is still in the cache.

SRM_INVALID_PATH
- *sourceSUR* does not exist

- *targetSURL* does not refer to a valid path.

SRM_DUPLICATION_ERROR
- *targetSURL* refers to an existing SURL and overwriting is not allowed.

SRM_AUTHORIZATION_FAILURE
- Client is not authorized to copy files from *sourceSURL*
- Client is not authorized to copy files into *targetSURL*
- Client is not authorized to copy files into the space that client provided with *targetSpaceToken* or *targetFileRetentionPolicyInfo*

SRM_ABORTED
- The requested file has been aborted.

SRM_RELEASED
- The requested file has been released.

SRM_REQUEST_SUSPENDED
- File request is suspended.

SRM_NO_USER_SPACE
- user space is not enough to hold the requested *SURL*.

SRM_NO_FREE_SPACE
- SRM space is not enough to hold the requested *SURL* for free.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.
- The file request would not be able to be completed within the *totalRequestTime*.
- The requested file has been suspended because the request has timed out.

5.9. srmReleaseFiles

This function is used to release pins on the previously requested "copies" (or "state") of the SURL. This function normally follows srmPrepareToGet or srmBringOnline functions.

5.9.1. Parameters

In:	string	requestToken,
	string	authorizationID,
	anyURI []	arrayOfSURLs,
	Boolean	doRemove
Out:	TReturnStatus	returnStatus,
	TSURLReturnStatus[]	arrayOfFileStatuses

5.9.2. Notes on the Behavior

a) *doRemove* by default is false. If remove is true, the pin on the file is released, the "copy" or "state" is removed and SRM may release the resource.
b) Directory is okay for SURL. In such case, it will release all files recursively in the directory.
c) If *requestToken* is not provided and SURLs are provided, then the SRM will release all the files specified by the SURLs owned by the caller, regardless of the *requestToken*.
d) If *requestToken* is provided and SURLs are not provided, then the SRM will release all the files in the request that is associated with the *requestToken*.
e) At least one of requestToken and SURLs must be provided.

f) If *requestToken* is not provided, then *authorizationID* may be needed as an additional verification method for the client authorization to release files. It may be inferred or provide in the call.

g) *srmReleaseFiles* is only valid after *srmPrepareToGet* or *srmBringOnline* operations. To release TURLs after a *srmPrepareToPut*, *srmAbortRequest* or *srmAbortFiles* must be used. If a client submits *srmReleaseFiles* after *srmPrepareToPut* or *srmPutDone*, then the SRM server returns SRM_INVALID_REQUEST.

5.9.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. All *SURLs* are released successfully.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some *SURLs* are successfully released, and some *SURLs* are failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to release files

SRM_INVALID_REQUEST
- *arrayOfSURLs* is empty.
- *requestToken* does not refer to an existing known request of *srmPrepareToGet* or *srmBringOnline* in the SRM server.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM
- input parameter *doRemove* is not supported in the SRM. *srmRm* must be used.

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL*. *SURL* is released successfully.

SRM_INVALID_PATH
- *SURL* does not refer to an existing file

SRM_AUTHORIZATION_FAILURE
- client is not authorized to release *SURL*

SRM_LAST_COPY
- *SURL* is the last copy when *remove* flag is on

SRM_FILE_LIFETIME_EXPIRED
- *SURL* is expired already.

SRM_ABORTED
- The requested file has been aborted.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

5.10. srmPutDone

srmPutDone() is used to notify the SRM that the client completed a file transfer to the TransferURL in the allocated space. This call should normally follow srmPrepareToPut.

5.10.1. Parameters

In:	string	requestToken,
	string	authorizationID,
	anyURI []	arrayOfSURLs
Out:	TReturnStatus	returnStatus,
	TSURLReturnStatus[]	arrayOfFileStatuses

5.10.2. Notes on the Behavior

a) Called by client after srmPrepareToPut() prepares the TURL and the client completes the file transfer into the prepared TURL.
b) *srmRm* may remove SURLs even if the statuses of the SURLs are SRM_FILE_BUSY. In this case, SRM_INVALID_PATH must be returned upon *srmPutDone* request.
c) If any additional *srmPutDone* is requested on the same SURL, SRM_DUPLICATION_ERROR must be returned at the file level.
d) When srmPutDone is called on a subset of srmPrepareToPut request, the request level status for the srmPutDone must refer to the subset of the request that srmPutDone was called on.
e) When srmPutDone is called without any file transfers into the TURL, SRM_INVALID_PATH must be returned at the file level status.
f) Before srmPutDone is called, if one of the parent directories is “moved”, srmPutDone on the old SURL must fail. The SURL must reflect the changes from the directory move.

5.10.3. Return Status Code

For request level return status,

SRM_SUCCESS
- All requests are successfully completed. *TURLs* contain data, and file lifetimes on the *SURLs* start.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some file requests are successfully completed, and some file requests are failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to call the request specified by the *requestToken*

SRM_INVALID_REQUEST
- *arrayOfSURLs* is empty.
- *requestToken* is empty.
- *requestToken* does not refer to an existing known request in the SRM server.

SRM_REQUEST_TIMED_OUT
- Total request time is over and the request is failed.

SRM_ABORTED

- The request has been aborted.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM

For file level return status,

SRM_SUCCESS
- successful request completion of the "put done" for the *targetSURL*

SRM_INVALID_PATH
- *SURL* does not refer to an existing file request
- no file transfer was performed on the SURL

SRM_AUTHORIZATION_FAILURE
- client is not authorized to call the request *srmPutDone*() on the *SURL*

SRM_DUPLICATION_ERROR
- *targetSURL* exists already.

SRM_FILE_LIFETIME_EXPIRED
- *targetSURL* has an expired *TURL*.

SRM_SPACE_LIFETIME_EXPIRED
- *targetSURL* has an expired space allocation.

SRM_ABORTED
- The requested SURL file has been aborted.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

5.11. srmAbortRequest

srmAbortRequest() allows clients to prematurely terminate asynchronous requests of any types. It may involve data transfer requests initiated by a call to *srmPrepareToGet*(), *srmBringOnline*(), *srmPrepareToPut*() or *srmCopy*(). The effect of *srmAbortRequest*() depends on the type of request. For data transfer request, the SRM will attempt a complete cleanup of running transfers and files in intermediate state.

5.11.1. Parameters

In:	string	requestToken,
	string	authorizationID
Out:	TReturnStatus	returnStatus

5.11.2. Notes on the Behavior

a) Terminate all files in the request regardless of the file state. Remove files from the queue, and release cached files if a limited lifetime is associated with the file.

b) Those files that are brought online with unlimited lifetime will remain in the space where they are brought in and are not removed. Clients need to remove explicitly through *srmRm* or *srmPurgeFromSpace*.
c) Abort must be allowed to all requests with *requestToken.*
d) When aborting srmCopy request, the request may contain one source SURL and multiple target SURLs. If the request is aborted by the source SURL, all file request of the same source SURL must be aborted. If the request is aborted by the target SURL, a specific target file request must be aborted, and other file requests from the same source SURL must not be aborted.
e) When aborting srmPrepareToGet request, all uncompleted files must be aborted, and all successfully completed files must be released.
f) When aborting srmPrepareToPut request before srmPutDone and before the file transfer, the SURL must not exist as the result of the successful abort on the SURL. Any srmRm request on the SURL must fail.
g) When aborting srmPrepareToPut request before srmPutDone and after the file transfer, the SURL may exist, and a srmRm request on the SURL may remove the requested SURL.
h) When aborting after srmPutDone, it must be failed for those files. An explicit srmRm is required to remove those successfully completed files for srmPrepareToPut.
i) When duplicate abort request is issued on the same request, SRM_SUCCESS may be returned to all duplicate abort requests and no operations on duplicate abort requests are performed.

5.11.3. Return Status Code

SRM_SUCCESS
- successful request completion. Request is aborted successfully.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some *SURLs* are successfully aborted, and some *SURLs* are failed. Some abort may be failed because files were successfully completed already.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to abort files in the request specified by the *requestToken*

SRM_INVALID_REQUEST
- *requestToken* does not refer to an existing known request in the SRM server.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM

5.12. srmAbortFiles

srmAbortFiles() allows clients to abort selective file requests from the asynchronous requests of any type. It may include data transfer requests initiated by a call to srmPrepareToGet(), srmBringOnline(), srmPrepareToPut(), or srmCopy(). The effect of a srmAbortFiles() depends on the type of the request.

5.12.1. Parameters

In:	string	requestToken,
	anyURI []	arrayOfSURLs,
	string	authorizationID
Out:	TReturnStatus	returnStatus,
	TSURLReturnStatus[]	arrayOfFileStatuses

5.12.2. Notes on the Behavior

a) Abort all files in this call regardless of the state.
b) When aborting srmCopy request, the request may contain one source SURL and multiple target SURLs. If the request is aborted by the source SURL, all file request of the same source SURL must be aborted. If the request is aborted by the target SURL, a specific target file request must be aborted, and other file requests from the same source SURL must not be aborted.
c) When aborting srmPrepareToGet file requests, all uncompleted files must be aborted, and all successfully completed files must be released.
d) When aborting srmPrepareToPut file requests before srmPutDone and before the file transfers, the SURL must not exist as the result of the successful abort on the SURL. Any srmRm request on the SURL must fail.
e) When aborting srmPrepareToPut file requests before srmPutDone and after the file transfer, the SURL may exist, and a srmRm request on the SURL may remove the requested SURL.
f) When aborting after srmPutDone, it must be failed for those files. An explicit srmRm is required to remove those successfully completed files for srmPrepareToPut.
g) This method must not change the request level status of the completed requests. Once a request is completed, the status of the request remains the same.
h) When duplicate abort file request is issued on the same files, SRM_SUCCESS may be returned to all duplicate abort file requests and no operations on duplicate abort file requests are performed.

5.12.3. Return Status Code

For request level return status,

SRM_SUCCESS
- successful request completion. All *SURLs* are aborted successfully.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some *SURLs* are successfully aborted, and some *SURLs* are failed. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to abort files in the request specified by the *requestToken*

SRM_INVALID_REQUEST
- *arrayOfSURLs* is empty.
- *requestToken* is empty.
- *requestToken* does not refer to an existing known request in the SRM server.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All files requests are failed.

- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM

For file level return status,

SRM_SUCCESS
- successful abort request completion for the *SURL. SURL* is aborted successfully.

SRM_INVALID_PATH
- *SURL* does not refer to an existing file request that is associated with the request token

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

5.13. srmSuspendRequest

srmSuspendedRequest is to suspend a previously submitted active request.

5.13.1. Parameters

In:	string	requestToken
	string	authorizationID
Out:	TReturnStatus	returnStatus

5.13.2. Notes on the Behavior

a) Suspend all files in this request until srmResumeRequest is issued.

5.13.3. Return Status Code

SRM_SUCCESS
- successful request completion. Request is suspended successfully.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to suspend the request specified by the *requestToken*

SRM_INVALID_REQUEST
- *requestToken* is empty.
- *requestToken* does not refer to an existing known request in the SRM server.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server

5.14. srmResumeRequest

srmResumeRequest is to resume previously suspended requeststst**.**

5.14.1. Parameters

In:	string	requestToken,
	string	authorizationID
Out:	TReturnStatus	returnStatus

5.14.2. Notes on the Behavior

a) Resume the previously suspended request.

5.14.3. Return Status Code

SRM_SUCCESS
- successful request completion. Request is resumed successfully.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to resume the request specified by the *requestToken*

SRM_INVALID_REQUEST
- *requestToken* is empty.
- *requestToken* does not refer to an existing known request in the SRM server.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM server

5.15. srmGetRequestSummary

srmGetRequestSummary is to retrieve a summary of the previously submitted request**.**

5.15.1. Parameters

In:	string []	arrayOfRequestTokens,
	string	authorizationID
Out:	TReturnStatus	returnStatus
	TRequestSummary[]	arrayOfRequestSummaries

5.15.2. Return Status Code

For request interface level return status,

SRM_SUCCESS

- All requests are successfully completed. All requests summaries are checked and returned successfully. Details are on the request status.

SRM_PARTIAL_SUCCESS
- All requests are completed. Summaries of some requests are successfully checked and returned, but some requests summaries are failed. Details are on the request status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to get summary of the request specified by the *requestToken*

SRM_INVALID_REQUEST
- *arrayOfRequestTokens* is empty.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM

SRM_FAILURE
- SRM failed to get summaries of all requests that are associated with request tokens.
- any other request failure. *Explanation* needs to be filled for details.

For request level return status,

SRM_INVALID_REQUEST
- *requestToken* does not refer to an existing known request in the SRM server.

SRM_SUCCESS
- The request has been completed successfully.

SRM_REQUEST_QUEUED
- successful request submission and all files request is still on the queue

SRM_REQUEST_INPROGRESS
- some files are completed, and some files are still on the queue

SRM_REQUEST_TIMED_OUT
- Total request time is over and the request is failed.

SRM_REQUEST_SUSPENDED
- The request has been suspended.

SRM_ABORTED
- The request has been aborted.

SRM_PARTIAL_SUCCESS
- All requests are completed. Some request is successfully completed, and some request is failed.

SRM_FAILURE
- The request is failed. *Explanation* needs to be filled for details.

5.16. srmExtendFileLifeTime

srmExtendFileLifetime() allows clients to extend lifetime of existing *SURL*s of volatile and durable file storage types or lifetime of pinned files (TURLs and those *TURLs* are of the results of *srmPrepareToGet, srmPrepareToPut* or *srmBringOnline*).

5.16.1. Parameters

In:	string	authorizationID,
	string	requestToken,
	anyURI []	arrayOfSURLs,
	int	newFileLifetime,
	int	newPinLifetime
Out:	TReturnStatus	returnStatus,
	TSURLLifetimeReturnStatus []	arrayOfFileStatuses

5.16.2. Notes on the Behavior

a) This method allows to change only one lifetime at a time (either SURL lifetime by the *newFileLifetime* or pin lifetime by the *newPinLifetime*), depending on the presence or absence of the request token. When both *newFileLifetime* and *newPinLifetime* are provided in the same request, the request is invalid, and SRM_INVALID_REQUST must be returned. SURL lifetimes are on SURLs that resulted from the successful srmCopy or srmPrepareToPut followed by srmPutDone, and pin lifetimes are on TURLs or file copies that resulted from srmPrepareToGet, srmPrepareToPut or srmBringOnline.
b) *newPinLifetime* and *newFileLifetime* are relative to the calling time. Lifetime will be set from the calling time for the specified period.
c) When the *requestToken* is provided, only pin lifetime is extended with *newPinLifetime*.
d) When SURL lifetime is extended with *newFileLifetime*, the request token must not be specified.
e) The number of lifetime extensions maybe limited by SRM according to its policies.
f) If original lifetime is longer than the requested one, then the requested one will be assigned.
g) When none of lifetime input parameters (*newPinLifetime* and *newFileLifetime*) is not specified, the SRM server does not change the lifetimes.
h) Lifetime cannot be extended on the released files, aborted files, expired files, and suspended files. For example, pin lifetime cannot be extended after srmPutDone is requested on SURLs after srmPrepareToPut. In such case, SRM_INVALID_REQUEST at the file level must be returned, and SRM_PARTIAL_SUCCESS or SRM_FAILURE must be returned at the request level.
i) Extending file lifetime on SURL is similar to srmExtendFileLifetimeInSpace.
j) If input parameters *newFileLifetime* or *newPinLifetime* request exceeds the remaining lifetime of the space, then SRM_SUCCESS is returned at the request and file level, and *TSURLLifetimeReturnStatus* contains the remaining lifetime.
k) Lifetime extension must fail on SURLs when their status is SRM_FILE_BUSY.
l) This method intends to negotiate a request of extension of file or pin lifetime. When new lifetime request exceeds the remaining lifetime of the space where SURLs are, SRM_SUCCESS is returned at the request level and at the file level, and TSURLLifetimeReturnStatus includes the remaining lifetime.

5.16.3. Return Status Code

For request level return status,

SRM_SUCCESS

- All requests are successfully completed. All *SURLs* or *TURLs* associated with *SURLs* in the specified request have an extended lifetime. Details are on the files status.

SRM_PARTIAL_SUCCESS

- All requests are completed. Lifetimes on some *SURLs* or *TURLs* are successfully extended, and lifetimes on some *SURLs* or *TURLs* are failed to be extended. Details are on the files status.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to extend file lifetime

SRM_INVALID_REQUEST
- *requestToken* does not refer to an existing known request in the SRM server.
- *requestToken* is not provided, and extending pinning lifetime of *TURLs* associated with *SURLs* require *requestToken*.

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- All files requests are failed.
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM

For file level return status,

SRM_SUCCESS
- successful request completion for the *SURL. SURL* or *TURL* associated with the *SURL* in the request has an extended lifetime.

SRM_INVALID_PATH
- *SURL* does not refer to an existing file
- *SURL* does not refer to an existing file request that is associated with the request token

SRM_FILE_LIFETIME_EXPIRED
- Lifetime on *SURL* is expired already.

SRM_ABORTED
- The requested file has been aborted.

SRM_RELEASED
- The requested file has been released.

SRM_INVALID_REQUEST
- Attempt to extend pin lifetimes on TURLs that have been already expired.

SRM_FAILURE
- The requested file has been suspended because the request has timed out.
- any other request failure. *Explanation* needs to be filled for details.

5.17. srmGetRequestTokens

srmGetRequestTokens retrieves request tokens for the client's requests, given client provided request description. This is to accommodate lost request tokens. This can also be used for getting all request tokens.

5.17.1. Parameters

In:	string	userRequestDescription,
	string	authorizationID
Out:	TReturnStatus	returnStatus
	TRequestTokenReturn[]	arrayOfRequestTokens

5.17.2. Notes on the Behavior

a) If userRequestDescription is null, returns all requests the client has.
b) If the user assigned the same description to multiple requests, the client may get back multiple request tokens each with the time the request was made.

5.17.3. Return Status Code

SRM_SUCCESS
- successful request completion. Request tokens are returned successfully.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is notauthorized to get request tokens specified by the userRequestDescription

SRM_INVALID_REQUEST
- *userRequestDescription* does not refer to any existing known requests

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM

6. Discovery Functions

summary:

6.1. srmGetTransferProtocols

This function is to discover what transfer protocols are supported by the SRM.

6.1.1. Parameters

In:	string	authorizationID,
Out:	TReturnStatus	returnStatus,
	TSupportedTransferProtocol[]	protocolInfo

6.1.2. Notes on the Behavior

a) *srmGetTransferProtocols*() returns the supported file transfer protocols in the SRM with any additional information about the transfer protocol.

6.1.3. Return Status Code

SRM_SUCCESS
- successful request completion. List of supported transfer protocols are returned successfully.

SRM_AUTHENTICATION_FAILURE
- SRM server failed to authenticate the client

SRM_AUTHORIZATION_FAILURE
- client is not authorized to request storage information

SRM_INTERNAL_ERROR
- SRM has an internal transient error, and client may try again.

SRM_NOT_SUPPORTED
- *function* is not supported in the SRM

SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

6.2. srmPing

This function is used to check the state of the SRM. It works as an “are you alive” type of call.

6.2.1. Parameters

In:	string	authorizationID,
Out:	string	versionInfo

TExtraInfo[] otherInfo

6.2.2. Notes on the Behavior

a) *srmPing*() returns a string containing SRM v2.2 version number as a minimal "up and running" information. For this particular SRM v2.2 version, it must be "**v2.2**". Other versions may have "**v1.1**", "**v3.0**", and so on.
b) Any additional information about the SRM can be provided in the output parameter *otherInfo*.

7. Appendix : Storage Resource Managers Concepts

Summary
Storage management is one of the most important enabling technologies for large-scale scientific investigations. Having to deal with multiple heterogeneous storage and file systems is one of the major bottlenecks in managing, replicating, and accessing files in distributed environments. Storage Resource Managers (SRMs), named after their web services protocol, provide the technology needed to manage the rapidly growing distributed data volumes, as a result of faster and larger computational facilities. SRMs are Grid storage services providing interfaces to storage resources, as well as advanced functionality such as dynamic space allocation and file management on shared storage systems. They call on transport services to bring files into their space transparently and provide effective sharing of files. SRMs are based on a common specification that emerged over time and evolved into an international collaboration. This approach of an open specification that can be used by various institutions to adapt to their own storage systems has proven to be a remarkable success – the challenge has been to provide a consistent homogeneous interface to the Grid, while allowing sites to have diverse infrastructures. In particular, one of the main goals to the SRM web service is to support optional features while preserving interoperability. The specification of the version described in this document, SRM v2.2, was also influenced by needs of a large international High Energy Physics collaboration, called WLCG, which adapted the SRM standard in order to handle the large volume of data expected when the Large Hadron Collider (LHC) goes online at CERN. This intense collaboration led to refinements and additional functionality in the SRM specification, and the development of multiple interoperating implementations of SRM for various complex multi-component storage systems.

7.1. Overview

Increases in computational power have created the opportunity for new, more precise and complex scientific simulations leading to new scientific insights. Similarly, large experiments generate ever increasing volumes of data. At the data generation phase, large volumes of storage have to be allocated for data collection and archiving. At the data analysis phase, storage needs to be allocated to bring a subset of the data for exploration, and to store the subsequently generated data products. Furthermore, storage systems shared by a community of scientists need a common data access mechanism which allocates storage space dynamically, manages stored content, and automatically remove unused data to avoid clogging data stores.

When dealing with storage, the main problems facing users today are the need to interact with a variety of storage systems and to pre-allocate storage to ensure data generation and analysis tasks can take place. Typically, each storage system provides different interfaces and security mechanisms. There is an urgent need to standardize and streamline the access interface, the dynamic storage allocation and the management of the content of these systems. The goal is to present to the users the same interface regardless of the type of system being used. Ideally, the management of storage allocation should become transparent.

To accommodate this need, the concept of Storage Resource Managers (SRMs) was devised [SSG02, SSG03] in the context of a project that involved High Energy Physics (HEP) and Nuclear Physics (NP). SRM is a specific set of web services protocols used to control storage systems from the Grid, and should not be confused with the more general concept of Storage Resource Management as used in industry, where Storage Resource Management refers to the process of optimizing the efficiency and speed of

storage devices (primary and secondary) and the efficient backup and recovery of data. By extension, a Grid component providing an SRM interface is usually called "an SRM."

After recognizing the value of this concept as a way to interact with multiple storage systems in a uniform way, several Department of Energy Laboratories (LBNL, Fermilab, and TJNAF), as well as CERN and Rutherford Appleton Lab in Europe, joined forces and formed a collaboration that evolved into a stable version, called SRM v1.1, that they all adopted. This led to the development of SRMs for several disk-based systems and mass storage systems, including HPSS [hpss] (at LBNL), CASTOR [castor] (at CERN), Enstore [enstore] (at Fermilab), and JasMINE [jasmine] (at TJNAF). The interoperation of these implementations was demonstrated and proved an attractive concept. However, the functionality of SRM v1.1 was limited, since space was allocated by default policies, and there was no support for directory structures. The collaboration is open to any institution willing and able to contribute. For example, when INFN, the Italian institute for nuclear physics, started working on their own SRM implementation they joined the collaboration. The collaboration also has an official standards body, the Open Grid Forum, OGF, where it is registered as GSM-WG (GSM is Grid Storage Management; SRM was already taken for a different purpose).

Subsequent collaboration efforts led to advanced features such as explicit space reservations, directory management, and support for Access Control Lists (ACL) to be supported by the SRM protocol, referred to as version 2.1. As with many advanced features, it was optional for the implementations to support them in order to be inclusive of implementations choosing not to support specific features.

Later, when a large international HEP collaboration, WLCG (the World-wide LHC Computing Grid) [wlcg-collab] decided to adopt the SRM standard, it became clear that many concepts needed clarification, and new functionality was added, resulting in SRM v2.2. While the WLCG contribution has been substantial, the SRM can also be used by other Grids, such as those using the EGEE gLite software [glite], or the Earth System Grid [esg]. There are many such Grids, often collaborations between the EU and developing countries. Having open source and license-free implementations based on the same standard is the best way to share this middleware technology.

7.2. The Basic Concepts

The ideal vision of a distributed Grid-based system is to have middleware facilities that give clients the illusion that all the compute and storage resources needed for their jobs are running on their local system. This implies that a client only logs in and gets authenticated once, and that some middleware software figures out where are the most efficient locations to move data to, to run the job, and to store the results in. The middleware software plans the execution, reserves compute and storage resources, executes the request, and monitors the progress. The traditional emphasis is on sharing large compute resource facilities, sending jobs to be executed at remote computational sites. However, very large jobs are often "data intensive", and in such cases it may be necessary to move the job to where the data sites are in order to achieve better efficiency. Alternatively, partial replication of the data can be performed ahead of time to sites where the computation will take place. Thus, it is necessary to also support applications that produce and consume large volumes of data. In reality, most large jobs in the scientific domain involve the generation of large datasets, the consumption of large datasets, or both. Therefore, it is essential that software systems exist that can provide space reservation and schedule the execution of large file transfer requests into the reserved spaces. Storage Resource Managers (SRMs) are designed to fill this gap.

In addition to storage resources, SRMs also need to be concerned with the *data resource* (or files that hold the data). A data resource is a chunk of data that can be shared by more than one client. In many applications, the granularity of a data resource is a file. It is typical in such applications that tens to hundreds of clients are interested in the same subset of files when they perform data analysis. Thus, the management of shared files on a shared storage resource is also an important aspect of SRMs. The decision of which files to keep in the storage resource is dependent on the cost of bringing files from remote systems, the size of the file, and the usage level of that file. The role of the SRM is to manage the space under its control in a way that is most cost beneficial to the community of clients it serves.
In general, an SRM can be defined as a middleware component that manages the dynamic use and content of a storage resource in a distributed system. This means that space can be allocated dynamically to a client, and that the decision of which files to keep in the storage space is controlled dynamically by the SRM. The main concepts of SRMs are described in [SSG02] and subsequently in more detail in a book chapter [SSG03]. The concept of a storage resource is flexible: an SRM could be managing one or more disk caches, or a hierarchical tape archiving system, or a combination of these. In what follows, they are referred to as "storage components". When an SRM at a site manages multiple storage resources, it may have the flexibility to store each file at any of the physical storage systems it manages (referred to as storage components) or even to replicate the files in several storage components at that site. The SRMs do not perform file transfer, but rather use file transfer services, such as GridFTP, to get files in/out of their storage systems. Some SRMs also provide access to their files through Posix or similar interfaces.

SRMs are designed to provide the following main capabilities:

1) *Non-interference with local policies*. Each storage resource can be managed independently of other storage resources. Thus, each site can have its own policy on which files to keep in its storage resources and for how long. The SRM will not interfere with the enforcement of local policies. Resource monitoring and management of both space usage and file sharing that enforce their local policies are the responsibility of SRMs.
2) *Pinning files.* Files residing in one storage component can be temporarily locked in place while used by an application, before being removed for resource usage optimization or transferred to another component. We refer to this capability as *pinning* a file, since a pin is a lock with a *lifetime* associated with it. A pinned file can be actively *released* by a client, in which case the space occupied by the file is made available to the client. SRMs can choose to keep or remove a released file depending on their storage management needs.
3) *Advance space reservations.* SRMs are components that manage the storage content dynamically. Therefore, they can be used to plan the storage system usage by permitting advance space reservations by clients.
4) *Dynamic space management.* Managing shared disk space usage dynamically is essential in order to avoid clogging of storage resources. SRMs use file replacement policies whose goal is to optimize service and space usage based on access patterns.
5) *Support abstract concept of a file name.* SRMs provide an abstraction of the file namespace using "Site URLs" (SURLs), while the files can reside in any one or more of the underlying storage components. An example of an SURL is: srm://ibm.cnaf.infn.it:8444/dteam/test.10193, where the first part "ibm.cnaf.infn.it:8444" is the address and port of the machine where the SRM resides, and the second part "dteam/test.10193" is the abstract file path, referred to as the Site File Name (SFN).
6) *Temporary assignment of transfer file names.* When requesting a file from an SRM, an SURL (see above) is provided. The SRM can have the file in several locations, or can bring it from tape to disk for access. Once this is done a "Transfer URL" (TURL) is returned for a temporary access to the file controlled by the pinning lifetime. A similar capability exists when a client wishes to put a file into

the SRM. The request provides the desired SURL for the file, and the SRM returns a TURL for the transfer of the file into the SRM. A TURL must have a valid transfer protocol such as: gsiftp://ibm139.cnaf.infn.it:2811//gpfs/dteam/test.10193. Note that the port 2811 is a GridFTP port.

7) *Directory Management and ACLs.* The advantage of organizing files into directories is well known, of course. However, SRMs provide directory management support to the SURL abstractions and keep the mapping to the actual files stored in the underlying file systems. Accordingly, Access Control Lists (ACLs) are associated with the SURLs.
8) *Transfer protocol negotiation.* When making a request to an SRM, the client needs to end up with a protocol for the transfer of the files that the storage system supports. In general, systems may be able to support multiple protocols and clients should be able to use different protocols depending on the system they are running on. SRM supports protocol negotiation, by matching the highest protocol they can support given an ordered list of preferred protocols by the client.
9) *Peer to peer request support.* In addition to responding to clients requests, SRMs are designed to communicate with each other. Thus, one SRM can be asked to copy files from/to another SRM.
10) *Support for multi-file requests*. The ability to make a single request to get, put, or copy multiple files is essential for practical reasons. This requirement is supported by SRMs by specifying a set of files. Consequently, such requests are asynchronous, and status functions need to be provided to find out the progress of the requests.
11) *Support abort, suspend, and resume operations*. These are necessary because requests may be running for a long time, in case that a large number of files is involved.

The main challenges for a common interface specification are to design the functionality of SRMs and their interfaces to achieve the goals stated above, and to achieve the interoperation of SRM implementations that adhere to the common interface specification. More details of the basic functionality can be found in [SSG03]. The specification of SRM interfaces and their corresponding WSDL can be found at the SRM collaboration web site [srm-collab].

The functions supported by SRMs in order to get or put files into the SRMs are referred to as "srmPrepareToGet" and "srmPrepareToPut". A set of files (or a directory) is provided by the client in the form of SURLs, and TURLs are returned by the SRM . The TURLs are used by the requesting clients to get or put files from/into the SRM using the TURL's transfer protocol. The function srnCopy provides the capability to replicate files from one SRM to another.

When using the space reservation function srmReserveSpace, the client can specify the desired space and duration of the reservation. The SRM returns the space and duration it is willing to allocate according to its policies, and a space token. If the client does not wish to accept that, it can issue srmReleaseSpace. Otherwise, it can put files into the reserved space by referring to the space token.

Directory functions are very similar to the familiar Unix functions and include srmLs, srmMkdir, srmRmdir, srmMv, and srmRm. Since files may have a limited lifetime in the SRM, these functions need to reflect lifetime status as well.

7.3. Additional concepts introduced with v2.2

Soon after the WLCG collaboration decided to try and adopt version 2.1 of the SRM specification as a standard for all their storage systems, it became clear that some concepts needed to be clarified, and perhaps new functionality added. The main issues were: 1) the specification of the storage properties; 2) the clarification of space and the meaning of a space token when it is returned after a space

reservation is made; and 3) the ability to request that files will be brought from archival storage into an online disk system for subsequent access. This led to a new SRM specification, referred to as SRM v2.2, presented in this document. We discuss each of these concepts further next.

Storage component properties

The issue of how to expose expected behavior of a storage component by the SRM was debated at great length. In the end, it was concluded that it is sufficient to expose two orthogonal properties: Retention Policy and Access Latency. These are defined below:

1) **Retention Policy**: REPLICA, OUTPUT, CUSTODIAL

The Quality of Retention is a kind of Quality of Service. It refers to the probability that the storage system loses a file. The type is used to describe the retention policy assigned to the files in the storage system at the moment when the files are written into the desired destination in the storage system. It is used as a property of space allocated through the space reservation function. Once the retention policy is assigned to a space, the files put in the reserved space will automatically be assigned the retention policy of the space. The description of Retention Policy Types is:

- REPLICA quality has the highest probability of loss, but is appropriate for data that can be replaced because other copies can be accessed in a timely fashion.
- OUTPUT quality is an intermediate level and refers to the data which can be replaced by lengthy or effort-full processes.
- CUSTODIAL quality provides low probability of loss.

2) **Access Latency**: ONLINE, NEARLINE

Files may be Online or Nearline. These terms are used to describe the latency to access a file. Latency can be improved by storage systems by replicating a file from nearline to online storage. We do not include here "offline" access latency, since a human has to be involved in getting offline storage mounted. For SRMs, one can only specify ONLINE and NEARLINE. The type is used to describe an access latency property that can be requested at the time of space reservation. The files that are contained in the space may have the same or lower access latency as the space. The ONLINE cache of a storage system is the part of the storage system which provides file access with online latencies. The description of Access Latency types is:

- ONLINE has the lowest latency possible. No further latency improvements can be applied to online files.
- NEARLINE files can have their latency improved to online latency automatically by staging the files to online cache.

Storage Areas and Storage Classes

Because of fairly complex storage systems used by the WLCG collaboration, it was obvious that referring to "storage system" is imprecise. Instead, the concept of a "storage area" is used. A storage system usually is referred to as a Storage Element, viz. a grid element providing storage services.

A Storage Element can have one or more storage areas. Each storage area includes parts of one or more hardware components (single disk, RAID, tape, DVD, …). Any combination of components is permissible. A storage area is specified by its properties which include the Access Latency and Retention Policy described above. Explicitly supported combinations are known as Storage Classes: online-replica (e.g. a common disk space allocated for online access), nearline-custodial (e.g. a high-quality robotic tape system), or online-custodial (e.g. a highly protected online disk that may keep multiple replicas, or an online disk with backup on a high-quality robotic tape system). Storage areas that consist of heterogeneous components are referred to as "composite storage areas" and the storage space in them as "composite space". "Composite storage elements" are storage elements serving composite storage

areas. Storage areas can share one or more storage components. This allows storage components to be partitioned for use by different user-groups or Virtual Organizations (VOs).

The SRM interface exposes only the storage element as a whole and its storage areas, not their components. However, a space reservation to a composite storage element can be made requesting Access Latency-Retention Policy combinations that may determine which storage components are assigned. Specifically, a space reservation to a composite storage element can request the following combinations to target the online or nearline storage components: a) online-replica to target the online storage components; b) nearline-custodial to target the nearline storage components (assuming they support custodial retention policy); c) online-custodial to target both the online and nearline storage components.

The function srmBringOnline

When a file is requested from a mass storage system (MSS), it is brought onto disk from tape in case that the file is not already on disk. The system determines which files to keep on disk, depending on usage patterns and system loads. However, this behavior is not always acceptable to large projects, since they need to be in control of what is online in order to ensure efficient use of computing resources. A user performing a large analysis may need to have all the files online before starting the analysis. Similarly, a person in charge of a group of analysts may wish to bring all the files for that group online for all of them to share. Therefore the concept of bringing files online was introduced.

srmBringOnline can be applied only to a composite space that has nearline as well as online components. When performing this function the SRM is in full control as to where files end up and this information is not visible to the client. For example, the SRM may have multiple online spaces, and it can choose which will be used for each file of the request. Similarly, the SRM can choose to keep multiple online replicas of the same file for transfer efficiency purposes. Once srmBringOnline is performed, subsequent srmPrepareToGet requests can be issued by clients, and TURLs returned, where each TURL indicates where the corresponding file can be accessed, and the protocol to be used.

7.4. Current SRM Implementations Based on V2.2 specification

Over the last 5-6 years, there were several implementations of SRMs. The first implementations were based on the v1.1 specifications (see [srm-collab]), at several institutions in the US and Europe, including Fermilab, Jlab, LBNL, and CERN. More recently, new implementation emerged that are based on the richer v2.2 specification described in this document. We mention here five such implementations, in order to illustrate the ability of SRMs to have the same interface to a variety of storage systems. The underlying storage systems can vary from a simple disk, multiple disk pools, mass storage systems, parallel file systems, to complex multi-component multi-tiered storage systems. While the implementations use different approaches, we illustrate the power of the SRM standard approach in that such systems exhibit a uniform interface and can successfully interoperate. While they adhere to the SRM v2.2 specification, some chose not to support some of the functionality. For example, some implementations do not support ACLs. Other implementation to a variety of systems built on top of innovative and sophisticated file system capabilities, such as SRB [srb] and L-Store [l-store] are underway. In addition, two test programs have been developed and are run daily to check the interoperability of these systems [MD'07, srm-tester]. The SRMs mentioned below (in alphabetical order) are fully implemented.

BeStMan – Berkeley Storage Manager

BeStMan is a java-based SRM implementation from LBNL. Its modular design allows different types of storage systems to be integrated in BeStMan while providing the same interface for the clients. Based on immediate needs, two particular storage systems are currently used. One supports multiple disks accessible from the BeStMan server, and the other is the HPSS storage system. Another storage system that was adapted with BeStMan is a legacy MSS at NCAR in support of the Earth System Grid project (www.earthsystemgrid.org).
BeStMan supports space management functions and data movement functions. Users can reserve space in the preferred storage system, and move files in and out of their space. When necessary BeStMan interacts with remote storage sites on their behalf, e.g. another gsiftp server, or another SRM. BeStMan does not support ACLs.

Castor-SRM
The SRM implementation for the CERN Advanced Storage system (CASTOR) is the result of collaboration between Rutherford Appleton Laboratory and CERN. Like that of other implementations, the implementation faced unique challenges. These challenges were based around the fundamental design concepts under which CASTOR operates, which are different from those of other mass storage systems. CASTOR trades some flexibility for performance, and this required the SRM implementation to have some loss of flexibility, but with gains in performance.
CASTOR is designed to work with a tape back-end and is required to optimise data transfer to tape, and also to ensure that data input to front-end disk cache is as efficient as possible. It is designed to be used in cases where it is essential to accept data at the fastest possible rate and have that data securely archived. These requirements are what cause differences between the CASTOR SRM implementation and others.
Space management in the CASTOR SRM is significantly different to those of other implementations. Since the design of the MSS is to optimise moving data from disk to tape, there is no provision for allowing dynamic space allocation at a user level. The CASTOR SRM does support space reservation, but as an asynchronous process involving physical reallocation of the underlying disk servers.

dCache-SRM
dCache [dcache] is a Mass Storage System developed jointly by Fermilab and DESY which federates a large number of disk systems on heterogeneous server nodes to provide a storage service with a unified namespace. dCache provides multiple means of file access protocols, including FTP, Kerberos GSSFTP, GSIFTP, HTTP, and dCap and xrootd [xrootd], POSIX APIs to dCache. dCache can act as a standalone Disk Storage System or as a front-end disk cache in a hierarchical storage system backed by a tape interface such as OSM, Enstore [enstore], Tsm, HPSS [hpss], DMF or CASTOR [castor]. dCache storage system has a highly scalable distributed architecture that allows easy addition of new services and data access protocols.
dCache provides load balancing and replication across nodes for “hot” files, i.e. files that are accessed often. It also provides a resilient mode, which guarantees that a specific number of copies of each file are maintained on different hardware. This mode can take advantage of otherwise unused and unreliable disk space on compute-nodes. This is a cost-effective means of storing files robustly and maintaining access to them in the face of multiple hardware failures.

DPM – Disk Pool Manager
The DPM (Disk Pool Manager) aims at providing a reliable and managed disk storage system, including multiple disk pools. The architecture is based on a database and multi-threaded daemons. It supports gsiftp, rfio, https and xrootd [xrootd] protocols.

A database backend (both MySQL and Oracle are supported) is used as a central information repository. It contains two types of information: 1) Data related to the current DPM configuration (pool and file system) and the different asynchronous requests (get and put) with their statuses. This information is accessed only by the DPM daemon. The SRM daemons only put the asynchronous requests and poll for their statuses. 2) Data related to the namespace, file permissions (ACLs included) and virtual IDs which allow a full support of the ACLs. Each user DN (Distinguished Name) or VOMS (Virtual Organization Membership Service) [voms] attribute is internally mapped to an automatically allocated virtual ID.

StoRM - Storage Resource Manager

StoRM (acronym for Storage Resource Manager) is an SRM service designed to manage file access and space allocation on high performing parallel and cluster file systems as well as on standard POSIX file systems. The StoRM project is the result of the collaboration between INFN – the Italian National Institute for Nuclear Physics - and the Abdus Salam ICTP for the EGRID Project for Economics and Finance research.

StoRM is designed to respond to a set of requests coming from various Grid applications allowing for standard POSIX access to files in local environment, and leveraging on the capabilities provided by modern parallel and cluster file systems such as the General Parallel File System (GPFS) from IBM. The StoRM service supports guaranteed space reservation and direct access (by native POSIX I/O calls) to the storage resource, as well as supporting other standard Grid file access libraries like RFIO and GFAL.

8. Security Considerations

The security requirements are achieved by combining Web Service/Grid standards with configuration description. For example, descriptions may be signed and encrypted. The deployment service must be allowed to decrypt configuration descriptions in order to process them.

9. Contributors

9.1. Editors information

Alex Sim
Lawrence Berkeley National Laboratory
1 Cyclotron Road, MS 50B-3238
Berkeley, CA 94720, USA
Email: asim@lbl.gov

Arie Shoshani
Lawrence Berkeley National Laboratory
1 Cyclotron Road, MS 50B-3238
Berkeley, CA 94720, USA
Email: shoshani@lbl.gov

9.2. Contributors

We gratefully acknowledge the contributors made to this specification by Timur Perelmutov and Don Petravick from Fermi National Accelerator Laboratory (FNAL) USA, Ezio Corso from International Centre for Theoretical Physics (ICTO) Italy, Luca Magnoni from Istituto Nazionale di Fisica Nucleare (INFN) Italy, Junmin Gu from Lawrence Berkeley National Laboratory (LBNL) USA, Paolo Badino, Olof Barring, Jean-Philippe Baud, Flavia Donno and Maarten Litmaath from LHC Computing Project (LCG, CERN) Switzerland, Shaun De Witt and Jens Jensen from Rutherford Appleton Laboratory (RAL) England, Michael Haddox-Schatz, Bryan Hess, Andy Kowalski and Chip Watson from Thomas Jefferson National Accelerator Facility (TJNAF) USA.

9.3. Acknowledgement

This document preparation has been partially supported by the Office of Energy Research, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

10. Intellectual Property Statement

The OGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the OGF Secretariat.

The OGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the OGF Executive Director.

11. Disclaimer

This document and the information contained herein is provided on an “As Is” basis and the OGF disclaims all warranties, express or implied, including but not limited to any warranty that the use of the information herein will not infringe any rights or any implied warranties of merchantability or fitness for a particular purpose.

12. Full Copyright Notice

13. References

[castor] http://castor.web.cern.ch/castor/
[dcache] http://www.dcache.org/
[enstore] http://www-ccf.fnal.gov/enstore/
[esg] http://www.earthsystemgrid.org/
[jasmine] The design and evolution of Jefferson lab’s Jasmine mass storage system, Hess, B.K.; Haddox-Schatz, M.; Kowalski, M.A., Proceedings of 13th NASA Goddard Conference on Mass Storage Systems, April 2005.
[glite] http://glite.web.cern.ch/glite/
[hpss] http://www.hpss-collaboration.org/hpss/index.jsp

[l-store] www.lstore.org/index.php
[MD'07] J. Mencak, F. Donno, The S2 testing suite, http://s-2.sourceforge.net
[srb] http://www.sdsc.edu/srb/index.php/Main_Page
[srm-tester] SRM Storage Tests and Monitoring, http://datagrid.lbl.gov/
[srm-collab] http://sdm.lbl.gov/srm-wg
[SSG02] Arie Shoshani, Alex Sim, Junmin Gu, Storage Resource Managers: Middleware Components for Grid Storage, Nineteenth IEEE Symposium on Mass Storage Systems, 2002
[SSG03] Storage Resource Managers: Essential Components for the Grid, *Arie Shoshani, Alexander Sim, and Junmin Gu*, in Grid Resource Management: State of the Art and Future Trends, Edited by Jarek Nabrzyski, Jennifer M. Schopf, Jan weglarz, Kluwer Academic Publishers, 2003
[voms] The Virtual Organization Membership Service,
http://www.globus.org/grid_software/security/voms.php
[wlcg-collab] http://cerncourier.com/cws/article/cern/29856
[xrootd] http://xrootd.slac.stanford.edu/