

GFD-R-P.129

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

A. Sim, Editor*, LBNL
A. Shoshani, Editor*, LBNL
Paolo Badino, CERN
Olof Barring, CERN
Jean-Philippe Baud, CERN
Ezio Corso, ICTO
Shaun De Witt, RAL
Flavia Donno, CERN
Junmin Gu, LBNL
Michael Haddox-Schatz, TJNAF
Bryan Hess, TJNAF
Jens Jensen, RAL
Andy Kowalski, TJNAF
Maarten Litmaath, CERN
Luca Magnoni, INFN
Timur Perelmutov, FNAL
Don Petravick, FNAL
Chip Watson, TJNAF

5/23/2008

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.

Copyright Notice

Copyright © Open Grid Forum (2007,2008). All Rights Reserved.

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	5
1. Storage Resource Managers Concepts	6
1.1. Summary	6
1.2. Overview	6
1.3. The Basic Concepts.....	7
1.4. Additional concepts introduced with v2.2	9
1.5. SRM Implementations	11
2. Common Type Definitions	13
2.1. Meaning of terms	13
2.2. File Storage Type.....	13
2.3. File Type.....	14
2.4. Retention Policy.....	14
2.5. Access Latency.....	14
2.6. Permission Mode.....	14
2.7. Permission Type.....	14
2.8. Request Type.....	15
2.9. Overwrite Mode.....	15
2.10. File Locality	15
2.11. Access Pattern.....	16
2.12. Connection Type.....	16
2.13. Status Codes	16
2.14. Retention Policy Info.....	17
2.15. Request Token.....	17
2.16. User Permission.....	17
2.17. Group Permission.....	17
2.18. Size in Bytes.....	17
2.19. UTC Time.....	18
2.20. Time in Seconds (Lifetime and RequestTime)	18
2.21. SURL.....	18
2.22. TURL	18
2.23. Return Status.....	18
2.24. Return Status for SURL	18
2.25. File MetaData.....	19
2.26. Space MetaData	19
2.27. Directory Option.....	20
2.28. Extra Info	20
2.29. Transfer Parameters.....	20
2.30. File Request for srmPrepareToGet.....	21
2.31. File Request for srmPrepareToPut.....	21
2.32. File Request for srmCopy	21
2.33. Return File Status for srmPrepareToGet	21
2.34. Return File Status for srmBringOnline.....	21
2.35. Return File Status for srmPrepareToPut.....	21
2.36. Return File Status for srmCopy.....	22
2.37. Request Summary.....	22
2.38. Return Status for SURL	22
2.39. Return File Permissions.....	23
2.40. Return Permissions on SURL.....	23
2.41. Return Request Tokens	23
2.42. Supported File Transfer Protocol.....	23
3. Space Management Functions	24
3.1. srmReserveSpace.....	24
3.2. srmStatusOfReserveSpaceRequest.....	26

3.3. srmReleaseSpace	27
3.4. srmUpdateSpace.....	29
3.5. srmStatusOfUpdateSpaceRequest.....	30
3.6. srmGetSpaceMetaData.....	31
3.7. srmChangeSpaceForFiles	32
3.8. srmStatusOfChangeSpaceForFilesRequest	35
3.9. srmExtendFileLifeTimeInSpace	37
3.10. srmPurgeFromSpace	38
3.11. srmGetSpaceTokens.....	40
4. Permission Functions.....	42
4.1. srmSetPermission	42
4.2. srmCheckPermission.....	43
4.3. srmGetPermission.....	44
5. Directory Functions	46
5.1. srmMkdir.....	46
5.2. srmRmdir	46
5.3. srmRm	47
5.4. srmLs.....	49
5.5. srmStatusOfLsRequest.....	51
5.6. srmMv.....	53
6. Data Transfer Functions	55
6.1. srmPrepareToGet.....	55
6.2. srmStatusOfGetRequest	58
6.3. srmBringOnline	61
6.4. srmStatusOfBringOnlineRequest	65
6.5. srmPrepareToPut.....	67
6.6. srmStatusOfPutRequest.....	71
6.7. srmCopy.....	74
6.8. srmStatusOfCopyRequest.....	78
6.9. srmReleaseFiles.....	81
6.10. srmPutDone	83
6.11. srmAbortRequest.....	84
6.12. srmAbortFiles	86
6.13. srmSuspendRequest	87
6.14. srmResumeRequest.....	88
6.15. srmGetRequestSummary	88
6.16. srmExtendFileLifeTime	89
6.17. srmGetRequestTokens	91
7. Discovery Functions.....	93
7.1. srmGetTransferProtocols.....	93
7.2. srmPing.....	93
8. Appendix I : Current SRM Implementations Based on v2.2 specification.....	95
8.1. BeStMan - Berkeley Storage Manager	95
8.2. Castor-SRM	95
8.3. dCache-SRM.....	96
8.4. DPM - Disk Pool Manager	96
8.5. StoRM - Storage Resource Manager.....	96
9. Appendix II : WLCG use case.....	96
Introduction.....	96
9.1. Storage classes.....	97
9.2. Removal policies	98
9.3. Protocol negotiation	98
9.4. Information discovery	98
9.5. srmReserveSpace.....	98
9.6. srmChangeSpaceForFiles.....	98
9.7. srmPurgeFromSpace	99

9.8. srmRm	99
9.9. srmLs	99
9.10. srmPrepareToGet	100
9.11. srmBringOnline	100
9.12. srmPrepareToPut	101
9.13. srmCopy	101
10. Security Considerations	102
11. Contributors	102
11.1. Editors information	102
11.2. Contributors	102
11.3. Acknowledgement	102
12. Intellectual Property Statement	103
13. Disclaimer	103
14. Full Copyright Notice	103
15. References	103

Introduction

This document contains the concepts and 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 followed by a 3-day meeting in September 2006 at CERN. Since that time several smaller meetings have taken place 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. This document reflects the current status of the specification, which has been frozen in order to allow multiple implementations to proceed.

The document is organized in seven sections. The first 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 second, called “Common Type Definitions” contains 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.

Appendix I lists several implementations of SRM v2.2 around the world, and their deployment in various sites.

As can be expected, when a large collaboration decide to use the SRM specification, it may choose to restrict some of the functionality according to their common projects requirements. For example, some collaboration may choose to restrict space reservations to administrators only, and not permit dynamic reservations by other users. Similarly, the collaboration may choose to support only permanent storage files, rather than allow automatic removal of files whose lifetime has expired by the SRM.

An interesting and influential collaboration is described in Appendix II. The collaboration is in the High Energy Physics domain, and its purpose is to develop the tools to manage the petabytes of data expected from the Large Hadron Collider (LHC). The collaboration, called Worldwide LHC Computing Grid (WLCG) project, involves implementing Storage Resource Managers on top of various storage systems based on the SRM v2.2 specification described here. Appendix II describes the restrictions and behaviors the WLCG project has chosen in order to achieve interoperability of all SRM implementations under a tight time schedule. It is important to note that the WLCG collaboration also added enhancement in terms of functionality and clarity of the specification, an invaluable contribution based on practical requirements.

For people not familiar with SRM concepts, it is advisable to read the first chapter. 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. Another SRM-related activity that was recently published is to provide a formal conceptual model of the SRM behavior [ISGC2007].

1. Storage Resource Managers Concepts

1.1. 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 as 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.

1.2. 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 removes 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 that data generation and analysis tasks can take place successfully. 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 U.S. Department of Energy Laboratories (LBNL, FNAL, and TJNAF), as well as CERN and RAL 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 FNAL), and JasMINE [jasmine] (at TJNAF). The interoperation of these implementations was demonstrated and proved to be 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.

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, SRMs are also used by other Grids, such as 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.

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; the acronym SRM was already taken for a different purpose in OGF).

1.3. 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 service is provided, 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 are 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 files from the SRMs 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 srmCopy 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.

1.4. 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.

1.4.1. 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:

a) **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.

b) **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.

1.4.2. 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.

1.4.3. 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.

1.5. SRM Implementations

Over the last 6-7 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 FNAL, Jlab, LBNL, and CERN. More recently, new implementations to a variety of storage systems emerged that are based on the richer v2.2 specification described in this document. 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]. Current SRM implementations based on v2.2 specification are described in Appendix I (in alphabetical order).

2. Common Type Definitions

Namespace SRM

2.1. Meaning of terms

- a) **Underlined** attributes are **REQUIRED**. The required attributes must be parsed correctly and must give proper error messages when not supported.
- b) 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.
- c) 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 to 18446744073709551615
 - unsignedInt ranges (inclusive): 0 to 4294967295
 - unsignedShort ranges (inclusive): 0 to 65535
- d) 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]".
- e) In “localSURL”, we mean local to the SRM that is processing the request.
- f) 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.
- g) Regarding file sharing by the SRM, it is a local implementation decision. An SRM can choose to share files by providing 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.
- h) The word “pinning” is limited to the “copies” or “states” of SURLs and the Transfer URLs (TURLs).
- i) 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.

2.2. File Storage Type

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

- Volatile file has a lifetime 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.

2.3. File Type

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

2.4. 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 is 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 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 assigned retention policy on the file can be found through the TMetaDataPathDetail structure returned by the srmLs function.

2.5. Access Latency

enum **TAccessLatency** { ONLINE, NEARLINE }

- 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.
- 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.
- For the SRM, ONLINE and NEARLINE are specified, and files may be ONLINE and/or NEARLINE.

2.6. Permission Mode

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

2.7. Permission Type

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

2.8. Request Type

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

2.9. Overwrite Mode

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

- 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.
- 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.

2.10. File Locality

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

- 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.
- The type is used to describe a file property that indicates the current location or status in the storage system.

2.11. Access Pattern

enum **TAccessPattern** { TRANSFER_MODE, PROCESSING_MODE }

- TAccessPattern may be passed as an input parameter to the srmPrepareToGet and srmBringOnline functions. It provides 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 allows 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.

2.12. Connection Type

enum **TConnectionType** { WAN, LAN }

- 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 input parameter may be passed to the srmPrepareToGet, srmPrepareToPut and srmBringOnline functions.

2.13. 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 must be used, in general
 - If a server does not support a method
 - If a server does not support particular optional input parameters

2.14. 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 must be rejected.

2.15. 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 can be immutable.
- Request tokens are case-sensitive.
- Request token is valid until the request is completed. However, SRM 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.

2.16. 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.

2.17. 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.

2.18. Size in Bytes

- Size in bytes is represented in unsigned long.

2.19. 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 extension.
- Format is same as in XML `dateTime` type [XSDDATE], 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.

2.20. 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 must have zero (0) second when no lifetime is left.
 - Some special meaning of negative time is defined when needed depending on the operation. E.g. `remainingTotalRequestTime` in `srmStatusOfGetRequest`

2.21. 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.

2.22. TURL

- TURL is represented in `anyURI`.

2.23. Return Status

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

2.24. Return Status for SURL

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

2.25. 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.

2.26. 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 *srnGetSpaceMetaData*.
- *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 *srnReserveSpace* and *srnStatusOfReserveSpaceRequest*.
- *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.

2.27. Directory Option

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

2.28. 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 *srnPing*, *srnGetTransferProtocols*, *srnStatusOfGetRequest*, and *srnStatusOfPutRequest*. 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 storage device, storage login ID, storage login authorization, but not limited to. Formerly, it was TStorageSystemInfo.

2.29. 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 *srnPrepareToGet*, *srnBringOnline*, *srnPrepareToPut* and *srnReserveSpace*. 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 must be specified in a preferred order on all SRM transfer functions.

- `arrayOfClientNetworks` provides a hint of the client IPs that SRM/dCache may use for optimization of its internal storage systems based on the client's accessible IP addresses.

2.30. File Request for `srmPrepareToGet`

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

2.31. File Request for `srmPrepareToPut`

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

2.32. File Request for `srmCopy`

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

2.33. 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* can provide more information about the transfer protocol so that client can access the TURL efficiently.
- *estimatedWaitTime* must 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, must be used.

2.34. Return File Status for `srmBringOnline`

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

- *estimatedWaitTime* must 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, must be used.

2.35. Return File Status for `srmPrepareToPut`

```
typedef struct { anyURI          SURL,
```

```

TResponseStatus      status,
unsigned long        fileSize,
int                  estimatedWaitTime,
int                  remainingPinLifetime, // on TURL
int                  remainingFileLifetime, // on SURL
anyURI               transferURL,
TExtraInfo[]         transferProtocolInfo
} TPutRequestFileStatus

```

- *transferProtocolInfo* of type *TExtraInfo* can provide 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* must 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, must be used.
- *remainingFileLifetime* is the lifetime on the SURL, and 0 means it expired. If SURL has an indefinite lifetime, then negative value, -1, must be used.

2.36. Return File Status for srmCopy

```

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

```

- *estimatedWaitTime* must 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, must be used.

2.37. Request Summary

```

typedef      struct { string      requestToken,
                  TResponseStatus 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.

2.38. 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.

2.39. Return File Permissions

```

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

```

2.40. 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*.

2.41. Return Request Tokens

```

typedef      struct { string          requestToken,
                dateTime             createdAtTime
            } TRequestTokenReturn

```

2.42. 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.

3. Space Management Functions

summary:

[srmReserveSpace](#)
[srmStatusOfReserveSpaceRequest](#)
[srmReleaseSpace](#)
[srmUpdateSpace](#)
[srmGetSpaceMetaData](#)
[srmChangeSpaceForFiles](#)
[srmStatusOfChangeSpaceForFilesRequest](#)
[srmExtendFileLifeTimeInSpace](#)
[srmPurgeFromSpace](#)
[srmGetSpaceTokens](#)

3.1. 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.

3.1.1. Parameters

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

3.1.2. Notes on the Behavior

- Input parameter *userSpaceTokenDescription* is case-sensitive. SRM is expected to keep it as client provides. It can be reused by the client. *srmGetSpaceTokens* must return all the space tokens that have the same *userSpaceTokenDescription*.
- If the input parameter *desiredLifetimeOfReservedSpace* is not provided, the lifetime of the reserved space may be set to "infinite (indefinite)" by default, if indefinite lifetime on space is supported by the implementation and local policy.

- c) If particular values of the input parameter *retentionPolicyInfo* cannot be satisfied by the SRM, 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 must be SRM_REQUEST_QUEUED.
- f) Input parameter *arrayOfExpectedFileSize* provides a hint that SRM may 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* provides information on 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 access information.
- h) SRM may return its default space size and lifetime if not requested by the client. SRM must return SRM_INVALID_REQUEST if SRM does not support default space sizes.
- i) If input parameter *desiredSizeOfTotalSpace* is not specified, the SRM may 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* must be returned 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.

3.1.3. 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

- 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 fails 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.
- Some input parameters are invalid.

SRM_NO_USER_SPACE

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

SRM_NO_FREE_SPACE

- SRM does not have enough free space for client to reserve.
- SRM does not have enough free space for the requested retentionPolicyInfo to reserve

SRM_EXCEED_ALLOCATION

- SRM does not have enough space for the request because the client requests 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
- specific values of the input parameter *retentionPolicyInfo* is not supported by the SRM
- a specified input parameter is not supported in the SRM

3.2. srmStatusOfReserveSpaceRequest

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

3.2.1. Parameters

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

3.2.2. Notes on the Behavior

- a) If the space reservation is not completed yet, *estimateProcessingTime* may be 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* must be returned to the client.

3.2.3. 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 fails 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.

SRM_EXCEED_ALLOCATION

- SRM does not have enough space for the request because the client requests more than the allocated space for the client.

SRM_NO_USER_SPACE

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

SRM_NO_FREE_SPACE

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

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
- a specified input parameter is not supported in the SRM

3.3. srmReleaseSpace

`srmReleaseSpace()` releases an occupied space.

3.3.1. Parameters

In:	string	authorizationID,
	string	<u>spaceToken</u> ,
	TExtraInfo[]	storageSystemInfo,

	boolean	forceFileRelease
Out:	TReturnStatus	<u>returnStatus</u>

3.3.2. Notes on the Behavior

- a) Input parameter *forceFileRelease* must be 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 must be released, even in OUTPUT or CUSTODIAL retention quality space.
- c) *srnReleaseSpace* 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 *srnRm* 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, those files are released at the end of their lifetime.
- d) If space is being released with *forceFileRelease* option while SURLS are being created with *srnPrepareToPut* or *srnCopy*, the file must be removed and SRM_INVALID_PATH must be returned by the *srnPutDone*, *srnStatusOfPutRequest*, or *srnStatusOfCopyRequest* when the file is volatile. If the file has permanent type, the file may be moved to the space that SRM manages, and the space would be successfully released. The subsequent *srnPutDone*, *srnStatusOfPutRequest*, or *srnStatusOfCopyRequest* must be successful.
- e) If space is being released without *forceFileRelease* option while SURLS are being created with *srnPrepareToPut* or *srnCopy*, SRM_FAILURE must be returned in *srnReleaseSpace*.
- 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.

3.3.3. Return Status Code

SRM_SUCCESS

- successful request completion. Space is successfully released.

SRM_AUTHENTICATION_FAILURE

- SRM fails 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.

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 token is already released.
- any other request failure. *Explanation* needs to be filled for details.

3.4. srmUpdateSpace

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

3.4.1. Parameters

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

3.4.2. Notes on the Behavior

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

3.4.3. 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 fails 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.
 - input parameter size or lifetime is not provided.
- SRM_EXCEED_ALLOCATION
- SRM does not have enough space for the request because the client requests more than the allocated space for the client.
- SRM_NO_USER_SPACE
- SRM does not have enough user space for the client request
- SRM_NO_FREE_SPACE
- SRM does not have enough free space for 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 occupied space.
 - any other request failure. *Explanation* needs to be filled for details.
- SRM_NOT_SUPPORTED
- *function* is not supported

3.5. srmStatusOfUpdateSpaceRequest

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

3.5.1. Parameters

In:	string string	authorizationID, <u>requestToken</u>
Out:	TReturnStatus unsigned long unsigned long int	<u>returnStatus</u> , sizeofTotalSpace, // best effort sizeofGuaranteedSpace, lifetimeGranted

3.5.2. Notes on the Behavior

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

3.5.3. 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 fails 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.
 - input parameter size or time is not provided.
- SRM_EXCEED_ALLOCATION
- SRM does not have enough space for the request because the client requests more than the allocated space for the client.
- SRM_NO_USER_SPACE
- SRM does not have enough user space for the client request
- SRM_NO_FREE_SPACE
- SRM does not have enough free space for 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 occupied space.
 - any other request failure. *Explanation* needs to be filled for details.
- SRM_NOT_SUPPORTED
- *function* is not supported

3.6. 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 previous space reservation through *srmReserveSpace* or *srmStatusOfReserveSpaceRequest*.

3.6.1. Parameters

In:	string string[]	authorizationID, <u>arrayOfSpaceTokens</u>
Out:	TReturnStatus TMetaDataSpace[]	<u>returnStatus</u> , arrayOfSpaceDetails

3.6.2. Notes on the Behavior

- a) Output parameters *unusedSize* in *TMetaDataSpace* must be 0 if there is no space left in the allocated space.

3.6.3. Return Status Code

For request level return Status,

SRM_SUCCESS

- successful request completion. Information of all requested spaces is 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 fails 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 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

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.

SRM_SPACE_LIFETIME_EXPIRED

- The lifetime on the space that is associated with the *spaceToken* has expired already

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.

3.7. 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 URLs will have a new space token. URLs 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 must be returned for later status request. There is no default behavior when target space token is not provided. In such case, the request must be failed, and the return status must be SRM_INVALID_REQUEST.

3.7.1. Parameters

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

3.7.2. Notes on the Behavior

- a) When space transition is completed successfully, SRM_SUCCESS must be returned for each SURL.
- b) For any forbidden transition, SRM_INVALID_REQUEST must be returned. It includes changing spaces on SURLs that their statuses are SRM_FILE_BUSY.
- c) Asynchronous operation may be necessary for some SRMs. 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 must be SRM_REQUEST_QUEUED, and *arrayOfFileStatuses* may not be filled and returned.
- e) All files specified in *arrayOfSURLs* will be associated with the new space specified by *targetSpaceToken*.
- f) Space allocation for a new target 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.
- i) When a space is successfully changed for a file from one space to another, the file must retain its remaining lifetime, or the file lifetime must 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.

3.7.3. 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 fails 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.
 - *targetSpaceToken* refers to a forbidden transition by the SRM.
- 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
- 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
 - a specified input parameter is not supported in the SRM

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.
 - 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 a file which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) request 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
- file request is failed.
 - any other request failure. *Explanation* needs to be filled for details.

3.8. 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*.

3.8.1. Parameters

In:	string string	authorizationID, <u>requestToken</u>
Out:	TReturnStatus int TSURLReturnStatus []	<u>returnStatus</u> , estimatedProcessingTime, arrayOfFileStatuses

3.8.2. 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* may be 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 must 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 must 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.

3.8.3. 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 is 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 fails 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.
- *targetSpaceToken* refers to a forbidden transition by the SRM.

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 large 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
- a specified input parameter is not supported in the SRM

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
- The status of *SURL* is *SRM_FILE_BUSY*.

SRM_EXCEED_ALLOCATION

- target space that is associated with *targetSpaceToken* is not large 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 a file which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) request 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
- file request is failed.
 - any other request failure. *Explanation* needs to be filled for details.

3.9. srmExtendFileLifetimeInSpace

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

3.9.1. Parameters

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

3.9.2. Notes on the Behavior

- a) *arrayOfSURLs* is optional. When SURLs are not provided, all files in the space must have the new extended lifetime.
- 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 must not change the lifetime.
- g) If input parameter *newLifeTime* exceeds 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.

3.9.3. 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 fails 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.
- 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
 - a specified input parameter is not supported in the SRM

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.

3.10. 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.

3.10.1. Parameters

In:	string	authorizationID,
	anyURI []	<u>arrayOfSURLs</u> ,
	string	<u>spaceToken</u> ,
	TExtraInfo[]	storageSystemInfo

Out: TReturnStatus returnStatus,
 TSURLReturnStatus[] arrayOfFileStatuses

3.10.2. 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* must be used.
- b) If the client has an administrative role that SRM can accept in an understandable form, the request must 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 must be returned.
- d) SRM must remove only the “copies” (or “state”) of the *SURLs* associated with the space token.

3.10.3. 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 fails 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 a file which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) request 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.

3.11. srmGetSpaceTokens

srmGetSpaceTokens() returns space tokens for currently allocated spaces for the client.

3.11.1. Parameters

In:	string string	userSpaceTokenDescription, authorizationID
Out:	TReturnStatus string[]	<u>returnStatus</u> , arrayOfSpaceTokens

3.11.2. Notes on the Behavior

- a) If *userSpaceTokenDescription* is null, returns all space tokens the client owns.
- b) Optional input parameter *userSpaceTokenDescription* is case-sensitive. SRM is expected to keep it as client provides. It may be reused by the client. *srmGetSpaceTokens* must return all the space tokens that have the *userSpaceTokenDescription*.
- c) If a client assigns the same description in multiple space reservations, the client may get back multiple space tokens.

3.11.3. Return Status Code

- SRM_SUCCESS
- All requests are successfully completed. Space tokens are returned successfully.
- SRM_AUTHENTICATION_FAILURE
- SRM fails 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

4. Permission Functions

summary:

[srmSetPermission](#)
[srmCheckPermission](#)
[srmGetPermission](#)

4.1. srmSetPermission

srmSetPermission is to set permission on local *SURL*.

4.1.1. Parameters

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

4.1.2. Notes on the Behavior

- a) Applies to both directories and files.
- 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 must be assumed that *TPermissionMode* is READ only.
- h) If *TPermissionType* is REMOVE, then the *TPermissionMode* must be 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 must work 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 user or group, the specified permissions must replace the current permissions. If an entry does not exist for the specified user or group, an entry must be created.
- j) *srmSetPermission* must modify permissions on *SURLs* even if the statuses of the *SURLs* are SRM_FILE_BUSY.

4.1.3. Return Status Code

- SRM_SUCCESS
- successful request completion. *SURL* has a new permission.
- SRM_AUTHENTICATION_FAILURE
- SRM fails 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
- a specified input parameter is not supported in the SRM

4.2. 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.

4.2.1. Parameters

In:	anyURI [] string TExtraInfo[]	<u>arrayOfSURLs</u> , authorizationID, storageSystemInfo
Out:	TResponseStatus TSURLPermissionReturn[]	<u>returnStatus</u> , arrayOfPermissions

4.2.2. Notes on the Behavior

- a) SRM checks files in its local online and nearline storage when supported.

4.2.3. 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 fails 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.

4.3. 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.

4.3.1. Parameters

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

4.3.2. Notes on the Behavior

- b) SRM checks files in its local online and nearline storage when supported.

4.3.3. 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 fails 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.

5. Directory Functions

summary:

[srmMkdir](#)
[srmRmdir](#)
[srmRm](#)
[srmLs](#)
[srmStatusOfLsRequest](#)
[srmMv](#)

5.1. srmMkdir

srmMkdir creates a directory in local SRM.

5.1.1. Parameters

In:	string	authorizationID,
	anyURI	<u>SURL</u> ,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	<u>returnStatus</u>

5.1.2. Notes on the Behavior

- Consistent with unix file system. Recursive creation of directories is not supported.
- SURL* may specify a new directory, as long as all parent directory hierarchy exists.

5.1.3. Return Status Code

SRM_SUCCESS

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

SRM_AUTHENTICATION_FAILURE

- SRM fails 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

5.2. srmRmdir

srmdir removes an empty directory in a local SRM.

5.2.1. Parameters

In:	string	authorizationID,
	anyURI	<u>SURL</u> ,
	TExtraInfo[]	storageSystemInfo,
	boolean	recursive
Out:	TReturnStatus	<u>returnStatus</u>

5.2.2. Notes on the Behavior

- It applies to directories only. To distinguish from *srmdir()*, this function is for directories only.
- Input parameter *recursive* must be false by default.
- When only expired volatile files are in the requested directory, *srmdir* 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 SRM, and may or may not be removed right away physically depending on the internal policy.

5.2.3. Return Status Code

SRM_SUCCESS

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

SRM_AUTHENTICATION_FAILURE

- SRM fails 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
- input parameter *recursive* is not supported in the SRM

5.3. srmdir

This function removes *SURLs* (the name space entries) in the storage system. Difference from *srmdirFromSpace* is that *srmdirFromSpace* removes only previously requested “copies” (or “state”) of the *SURL* in a particular space, and *srmdirFromSpace* must not remove *SURLs* or the name space entries.

5.3.1. Parameters

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

5.3.2. Notes on the Behavior

- To distinguish from *srmRmdir()*, this function applies to files only.
- srmRm* removes all copies or states of SURLs in the storage, and removes the entry from the name space.
- When an SURL is removed, all associated pinned TURLs must be all released and removed as well.
- srmLs*, *srmPrepareToGet* or *srmBringOnline* must not find these removed files any more. It must set file requests on SURL from *srmPrepareToGet* as SRM_ABORTED.
- srmRm* aborts the SURLs from *srmPrepareToPut* requests not yet in SRM_PUT_DONE state, and must set its file status as SRM_ABORTED.
- srmRm* must 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*.

5.3.3. 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 fails 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.

5.4. srmLs

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

5.4.1. Parameters

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

5.4.2. Notes on the Behavior

- a) Applies to both directories and files
- b) *fullDetailedList* must be 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 must be 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 must be provided.
- e) If *allLevelRecursive* is "true", it dominates, i.e. SRM ignores *numOfLevels*. If *allLevelRecursive* is "false" or missing, then SRM uses *numOfLevels*. If *numOfLevels* is "0" (zero) or missing, SRM assumes a single level. If both *allLevelRecursive* and *numOfLevels* are missing, SRM assumes a single level.
- f) Default value of *numOfLevels* is 1 when not provided.
- g) If *numOfLevels* is 0, then information about directory itself must be returned. Negative value must be invalid.
- h) If *numOfLevels* is 1, then information about files in the directory must be returned. Negative value must be 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*", if supported, only the files with that type must be returned.
- k) Empty directories must 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.

5.4.3. 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 fails 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 value for *numOfLevels*, *offset* or *count* is provided.
- Operation on the path such as non-existing or system-prohibited file or directory may be prohibited.

SRM_NOT_SUPPORTED

- Requested *fileStorageType* is not supported in SRM
- Filtering *fileStorageType* is not supported in SRM
- Operation on the path such as browsing the top directory is not supported. *Explanation* needs to be filled for details.
- Directory operation (directory *SURL*, *allLevelRecursive* or *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 a file which there is an active `srmPrepareToPut` (no `srmPutDone` is yet called) request for.
- SRM_FILE_LIFETIME_EXPIRED
- lifetime on *SURL* has 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 is not supported. *Explanation* needs to be filled for details.
- SRM_FAILURE
- any other request failure. *Explanation* needs to be filled for details.

5.5. srmStatusOfLsRequest

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

5.5.1. Parameters

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

5.5.2. Notes on the Behavior

- a) Empty directories must be returned.
- b) For non-existing files or directories in the client request, `SRM_INVALID_PATH` must be returned.

5.5.3. 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 fails 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 value for *offset* or *count* is 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*, browsing top directory, allLevelRecursive or 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 a file which there is an active srmPrepareToPut (no srmPutDone is yet called) request for.
- SRM_FILE_LIFETIME_EXPIRED
- lifetime on *SURL* has 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 is not supported. *Explanation* needs to be filled for details.

SRM_FAILURE

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

5.6. srmMv

srmMv moves a file or a directory to destination.

5.6.1. Parameters

In:	string	authorizationID,
	anyURI	<u>fromSURL</u> ,
	anyURI	<u>toSURL</u> ,
	TExtraInfo[]	storageSystemInfo
Out:	TReturnStatus	<u>returnStatus</u>

5.6.2. Notes on the Behavior

- Applies to both directories and files, and it works like unix *mv*.
- Authorization checks need to be performed on both *fromSURL* and *toSURL*.
- srmMv* must fail on SURL that its status is SRM_FILE_BUSY, and SRM_FILE_BUSY must be returned.
- Moving an SURL to itself results in no operation and SRM_SUCCESS must be returned for no operation.
- When moving an SURL to already existing SURL, SRM_DUPLICATION_ERROR must be returned.

5.6.3. 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 fails 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 a file which there is an active srmPrepareToPut (no srmPutDone is yet called) request 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

6. Data Transfer Functions

summary:

[srmPrepareToGet](#)
[srmStatusOfGetRequest](#)
[srmPrepareToPut](#)
[srmStatusOfPutRequest](#)
[srmCopy](#)
[srmStatusOfCopyRequest](#)
[srmBringOnline](#)
[srmStatusOfBringOnlineRequest](#)

[srmReleaseFiles](#)
[srmPutDone](#)

[srmAbortRequest](#)
[srmAbortFiles](#)
[srmSuspendRequest](#)
[srmResumeRequest](#)

[srmGetRequestSummary](#)

[srmExtendFileLifeTime](#)
[srmGetRequestTokens](#)

6.1. srmPrepareToGet

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

6.1.1. Parameters

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

TGetRequestFileStatus[]	arrayOfFileStatuses,
int	remainingTotalRequestTime

6.1.2. Notes on the Behavior

- a) The default value of “lifetime” for Volatile or Durable files may be the lifetime left in the space of the corresponding file type. The default value of “fileStorageType” is Volatile.
- b) If optional input parameter *targetSpaceToken* is provided, then the target space token must refer to an online space. All requested files must be prepared into the target space, when supported.
- c) Input parameter *targetFileRetentionPolicyInfo* of *TRetentionPolicyInfo* specifies 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 must be failed with SRM_INVALID_REQUEST.
- e) Access latency must be ONLINE always.
- f) If input parameter *TAccessPattern* is provided at the request-level, then all files must 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 is expected to keep it as client provides. It may be reused by the client. It may be used in the *srmGetRequestTokens* function to get back the system assigned request tokens. *srmGetRequestTokens* must 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* specifies a client-preferred lifetime (expiration time) on the prepared TURL.
- j) If the request is accepted and queued, SRM assigns the *requestToken* for asynchronous status checking. In such case, the returned status code must be SRM_REQUEST_QUEUED.
- k) *totalRequestTime* means: All the file transfer for this request must be completed 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, the request may be retried for a duration, which is dependent on the SRM, in case transient error occurs.
- m) If input parameter *desiredTotalRequestTime* is 0 (zero), each file request must be tried at least once. Negative value must be 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 must 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 must 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 must return SRM_REQUEST_QUEUED and keep trying the request for the duration of *desiredTotalRequestTime*. In the output parameter of explanation in *returnStatus*, the SRM may make explicit that the retry is being done. If streaming mode is not supported, the SRM

must return SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests are successful) or SRM_FAILURE at the request level.

r) Zero length files must not fail on srmPrepareToGet.

6.1.3. 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 fails to authenticate the client

SRM_AUTHORIZATION_FAILURE

- client is not authorized to submit the request

SRM_INVALID_REQUEST

- *arrayOfFileRequest* is empty
- Input parameters *targetSpaceToken* and *TRetentionPolicyInfo* do not match.
- Access latency is something other than ONLINE.
- *targetSpaceToken* does not refer to an existing known space in the SRM.

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 files.

SRM_NO_FREE_SPACE

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

SRM_NOT_SUPPORTED

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

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 a file which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) request for.

SRM_FILE_UNAVAILABLE

- the requested file is temporarily unavailable.

SRM_INVALID_PATH

- *SURL* does not refer to an existing known file in SRM

SRM_AUTHORIZATION_FAILURE

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

SRM_FILE_LIFETIME_EXPIRED

- *SURL* is expired
- *TURL* 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 file.

SRM_NO_FREE_SPACE

- SRM space is not enough to hold requested file 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.

6.2. srmStatusOfGetRequest

This function checks the status of the previously requested *srmPrepareToGet*. Request token from *srmPrepareToGet* must be provided.

6.2.1. Parameters

In:	string	<u>requestToken</u> ,
	string	authorizationID,
	anyURI []	arrayOfSourceSURLs

Out:	TReturnStatus	<u>returnStatus</u> ,
	TGetRequestFileStatus[]	arrayOfFileStatuses,
	int	remainingTotalRequestTime

6.2.2. Notes on the Behavior

- a) The default value of “lifetime” for Volatile or Durable files may be the lifetime left in the space of the corresponding file type. The default value of “fileStorageType” is Volatile.
- b) If *arrayOfSourceURLs* 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 must be SRM_FILE_PINNED.
- d) When the file is ready for the client, the file is implicitly pinned in the cache and lifetime must 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 yet called) request 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 completed 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 must 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 must return SRM_REQUEST_QUEUED and keep trying the request for the duration of *desiredTotalRequestTime* from the request. *remainingTotalRequestTime* may be returned. In the output parameter of explanation in *returnStatus*, the SRM may make explicit that the retry is being done. If streaming mode is not supported, the SRM must return SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests are successful) or SRM_FAILURE at the request level. Clients may need to release files or clean up the target space when target space token is provided in the request.
- 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.

6.2.3. 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 fails 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.
- 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 files.
- SRM_NO_USER_SPACE
- user space is not enough to hold all requested files.
- SRM_NO_FREE_SPACE
- SRM space is not enough to hold all requested files for free.
- SRM_NOT_SUPPORTED
- SRM does not support the given input parameters. For example, client requests *bbftp* for the only transfer protocol, but SRM does not support that. Client requests *desiredFileStorageType* that is not supported by the SRM.
 - *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM.
 - Directory operation is not supported in the SRM.
 - Recursive directory operation is not supported in the SRM.
 - None of the file transfer protocols are supported in the SRM.
- 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 a file which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) request 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
 - *SURL* does not refer to an existing file in SRM
- SRM_AUTHORIZATION_FAILURE
 - client is not authorized to retrieve the file that is associated with the *SURL*
- SRM_FILE_LIFETIME_EXPIRED
 - *SURL* is expired
 - *TURL* 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 file.
- SRM_NO_FREE_SPACE
 - SRM space is not enough to hold requested file 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.

6.3. srmBringOnline

This function brings files online upon the client 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 refer to an online space, the files must be brought online using the space associated with the space token. It may be an asynchronous operation, and request token must be returned if asynchronous operation is necessary in SRM. The status may only be checked through *srmStatusOfBringOnlineRequest* with the returned request token.

This function is similar to *srmPrepareToGet*, but it does not return Transfer URL (*TURL*).

6.3.1. Parameters

In:	string	authorizationID,
	TGetFileRequest[]	<u>arrayOfFileRequests</u> ,
	string	userRequestDescription,
	TExtraInfo[]	storageSystemInfo,
	TFileStorageType	desiredFileStorageType,
	int	desiredTotalRequestTime,
	int	desiredLifetime, // lifetime on online
	string	targetSpaceToken,

TRetentionPolicyInfo	targetFileRetentionPolicyInfo,
TTransferParameters	transferParameters,
int	deferredStartTime
Out: TReturnStatus	<u>returnStatus</u> ,
string	requestToken,
TBringOnlineRequestFileStatus[]	arrayOfFileStatuses,
int	remainingTotalRequestTime,
int	remainingDeferredStartTime

6.3.2. Notes on the Behavior

- a) Optional 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 must be invalid.
- b) Output parameter *remainingDeferredStartTime* indicates how long the *deferredStartTime* is left, if supported. Negative value must be invalid.
- c) Input parameter *targetFileRetentionPolicyInfo* of *TRetentionPolicyInfo* specifies 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 must be failed, and SRM_INVALID_REQUEST must 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, when supported.
- g) Access latency after the operation is successful must be ONLINE, and the file may also be on NEARLINE as well.
- h) It is up to the SRM to decide *TConnectionType* if not provided.
- i) The *userRequestDescription* is a user designated name for the request. It is case-sensitive. SRM must keep it as client provides. It may be reused by the client. It can be used in the *srmGetRequestTokens* function to get back the system assigned request tokens. *srmGetRequestTokens* must 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, accepted and queued. The returned status code must be SRM_REQUEST_QUEUED. To get subsequent status and results, separate calls must be made through *srmStatusOfBringOnline*.
- l) The returned request token must 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 completed 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, the request may be retried for a duration which is dependent on the SRM, in case transient error occurs.
- o) If input parameter *desiredTotalRequestTime* is 0 (zero), each file request must be tried at least once. Negative value must be 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 must 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 must return SRM_REQUEST_QUEUED and keep trying the request for the duration of *desiredTotalRequestTime*. In the output parameter of explanation in *returnStatus*, the SRM may make explicit that the retry is being done. If streaming mode is not supported, the SRM must return SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests are successful) or SRM_FAILURE at the request level.

6.3.3. 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 fails 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.
- Input parameters *targetSpaceToken* and *TRetentionPolicyInfo* do not match.
- *targetSpaceToken* does not refer to an existing known space in the SRM.
- *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 files.

SRM_NO_USER_SPACE

- user space is not enough to hold all requested files.

SRM_NO_FREE_SPACE

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

SRM_INTERNAL_ERROR

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

SRM_NOT_SUPPORTED

- SRM does not support the given input parameters. For example, client requests bbftp for the only transfer protocol, but SRM cannot support that. Client requests *desiredFileStorageType* that is not supported by the SRM.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM.
- Given *deferredStartTime* is not supported in the SRM.
- Directory operation is not supported in the SRM.
- Recursive directory operation is not supported in the SRM.
- None of the given file transfer protocols are 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 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 a file which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) request for.

SRM_FILE_UNAVAILABLE

- the requested file is temporarily unavailable.

SRM_INVALID_PATH

- *SURL* does not refer to an existing known file in SRM

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 file.

SRM_NO_FREE_SPACE

- SRM space is not enough to hold requested file 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.

6.4. srmStatusOfBringOnlineRequest

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

6.4.1. Parameters

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

6.4.2. Notes on the Behavior

- a) If *arrayOfSourceSURLs* is not provided, all files in the request must be returned.
- 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 must be invalid.
- 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 yet called) request 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 completed 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 must 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 must return SRM_REQUEST_QUEUED and keep trying the request for the duration of *desiredTotalRequestTime* from the request. *remainingTotalRequestTime* may be returned. In the output parameter of explanation in *returnStatus*, the SRM may make explicit that the retry is being done. If streaming mode is not supported, the SRM must return

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 is provided.

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

6.4.3. 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 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.

SRM_NOT_SUPPORTED

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

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 files.

SRM_NO_USER_SPACE

- user space is not enough to hold all requested files.

SRM_NO_FREE_SPACE

- SRM space is not enough to hold all requested files 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 a file which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) request 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 file.

SRM_NO_FREE_SPACE

- SRM space is not enough to hold requested file 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.

6.5. srmPrepareToPut

This function is used to write files into the storage. Upon the client 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 may be an asynchronous operation, and

request token must be returned if the request is valid, accepted and queued to the SRM. The status may only be checked through `srmStatusOfPutRequest` with the returned request token.

6.5.1. 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	<u>returnStatus</u> ,
	string	requestToken,
	TPutRequestFileStatus[]	arrayOfFileStatuses,
	int	remainingTotalRequestTime

6.5.2. Notes on the Behavior

- The default value of “lifetime” for Volatile or Durable files may be the lifetime left in the space of the corresponding file type. The default value of “fileStorageType” is Volatile.
- TURL returned by the `srmPrepareToPut` may not be used for read access with any protocol. An explicit `srmPrepareToGet` or `srmBringOnline` is required.
- 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.
- Input parameter *userRequestDescription* may be null, and it is case-sensitive when provided. SRM must keep it as client provides. It may be reused by the client. It can be used in the `srmGetRequestTokens` function to get back the system assigned request tokens. `srmGetRequestTokens` must return all the request tokens that have the *userRequestDescription*.
- When optional input parameter *targetSpaceToken* is provided at the request-level, 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.
- Optional input parameter *targetFileRetentionPolicyInfo* of *TRetentionPolicyInfo* specifies the desired retention policy information on the file when the file is written into the target storage system.
- If both input parameters *targetSpaceToken* and *TRetentionPolicyInfo* are provided, then their types must match exactly. Otherwise, the request must be failed and SRM_INVALID_REQUEST must be returned.
- Only push mode is supported for file transfers that client must “push” the file to the prepared TURL.
- Input parameter *targetSURL* in the *TPutFileRequest* must to be local to SRM. If *targetSURL* is not specified, SRM may generate a reference SURL for the file request automatically and put

it in the specified user space if provided. Then, this reference SURL must 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) Optional 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 must 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 SRM must return SRM_FAILURE at the file level.
- l) Optional 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 successful *srmPutDone* is executed.
- m) The lifetime of the SURL starts as soon as SRM receives the *srmPutDone()*. If *srmPutDone()* is not provided, then the file in that space is 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 and queued, SRM assigns the *requestToken* for asynchronous status checking. In such case, the returned status code must be SRM_REQUEST_QUEUED.
- o) *totalRequestTime* means: All the file transfer for the request must be completed within the *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 must be retried for a duration which is dependent on the SRM.
- q) If input parameter *desiredTotalRequestTime* is 0 (zero), each file request must be tried at least once. Negative value must be 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 must 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 must return SRM_REQUEST_QUEUED and keep trying the request for the duration of *desiredTotalRequestTime* from the request. In the output parameter of explanation in *returnStatus*, the SRM may make explicit that the retry is being done. If streaming mode is not supported, the SRM must return SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests are 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* is 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, otherwise the request must be processed.
- w) After TURL is returned, *srmMv* operation on the corresponding SURL may be requested. *srmPutDone* on the original SURL must be successful, 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.

6.5.3. 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 fails 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.

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 files.

SRM_NO_FREE_SPACE

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

SRM_INTERNAL_ERROR

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

SRM_NOT_SUPPORTED

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

SRM_FILE_BUSY

- client requests for files which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) or *srmCopy* request for.

SRM_AUTHORIZATION_FAILURE

- client is not authorized to write 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 file.

SRM_NO_FREE_SPACE

- SRM space is not enough to hold the requested file 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.

6.6. srmStatusOfPutRequest

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

6.6.1. Parameters

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

6.6.2. Notes on the Behavior

- a) The default value of “lifetime” for Volatile or Durable files may be the lifetime left in the space of the corresponding file type. The default value of “fileStorageType” is Volatile.
- b) If *arrayOfTargetURLs* is not provided, all the file statuses in the request must be returned.
- c) When the space is ready for client to “put” data and TURL is prepared, the return status code must 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 must be enforced. TURLs must not be valid any more after the pin lifetime is over if *srmPutDone* or *srmAbortRequest* is not submitted on the SURL before the expiration. In such case, the SRM must return SRM_FAILURE at the file level.
- e) If a *targetSURL* is provided with some directory structure, the directory structure must exist, and SRM must 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 by the client.
- f) Lifetime on SURL starts when successful *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 fail.
- i) *totalRequestTime* means: All the file transfer for this request must be completed within the *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 must 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 must return SRM_REQUEST_QUEUED and keep trying the request for the duration of *desiredTotalRequestTime* from the request. *remainingTotalRequestTime* may be returned. In the output parameter of explanation in *returnStatus*, the SRM may make explicit that the retry is being done. If streaming mode is not supported, the SRM must return SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests are successful) or SRM_FAILURE at the request level. Clients may need to clean up the target space when target space token is provided.
- l) Upon *srmPrepareToPut*, SURL entry is added 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* is 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* must 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 must be successful, 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 is specified in the input for specific file statuses.

6.6.3. 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 fails 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.
- *targetSpaceToken* that client provided does not refer to an existing space in the SRM.

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 files.

SRM_NO_USER_SPACE

- user space is not enough to hold all requested files.

SRM_NO_FREE_SPACE

- SRM space is not enough to hold all requested files 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 does not support the given input parameters. For example, client requests *bbftp* for the only transfer protocol, but SRM cannot support that. Client requests *desiredFileStorageType* that is not supported by the SRM.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM.
- None of the file transfer protocols are 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_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 permitted.
- SRM_FILE_BUSY
- client requests for files which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) or *srmCopy* request for.
- SRM_AUTHORIZATION_FAILURE
- client is not authorized to write 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 file.
- SRM_NO_FREE_SPACE
- SRM space is not enough to hold the requested file 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.

6.7. 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 to. 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) to, 2. Source SRM is where client makes a *srmCopy* request (PUSH case) to.

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

6.7.1. Parameters

In:	string TCopyFileRequest[] string TOverwriteMode int int TFileStorageType string TRetentionPolicyInfo TExtraInfo[] TExtraInfo[]	authorizationID, <u>arrayOfFileRequests</u> , userRequestDescription, overwriteOption, desiredTotalRequestTime, desiredTargetSURLLifeTime, targetFileStorageType, targetSpaceToken, targetFileRetentionPolicyInfo, sourceStorageSystemInfo, targetStorageSystemInfo
Out:	TReturnStatus string TCopyRequestFileStatus[] int	<u>returnStatus</u> , requestToken, arrayOfFileStatuses, remainingTotalRequestTime

6.7.2. Notes on the Behavior

- a) The default value of *desiredTargetSURLLifeTime* for Volatile or Durable files may be the lifetime left in the space of the corresponding file type. The default value of *targetFileStorageType* is Volatile.
- b) When aborted, target URLs must be provided.
- c) Input parameter *userRequestDescription* may be null, and it is case-sensitive when provided. SRM is expected to keep it as client provides. It may be reused by the client. It can be used in the *srmGetRequestTokens* function to get back the system assigned request tokens. *srmGetRequestTokens* must return all the request tokens that have the *userRequestDescription*.
- d) If optional input parameter *targetSpaceToken* is provided at the request-level, then 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* specifies 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 must be failed, and SRM_INVALID_REQUEST must be returned.
- g) If request is accepted and queued, SRM must assign the *requestToken* for asynchronous status checking. In such case, the returned status code must be SRM_REQUEST_QUEUED.
- h) Pull mode: copy from remote storage location to the SRM. (e.g. from remote storage to underlying MSS.)

- i) Push mode: copy from the SRM to remote storage location.
- j) SRM must always release files through *srmReleaseFiles* from the source after copy is completed, if source is an SRM and PULL mode is performed.
- k) SRM must always issue *srmPutDone* to the target after copy is completed, if target is an SRM and PUSH mode is performed.
- l) Note there is no protocol negotiation with the client for the request.
- m) *totalRequestTime* means: All the file transfer for the request must be completed in this *totalRequestTime*. Otherwise, the request must be 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 must be retried for a duration which is dependent on the SRM.
- o) If input parameter *desiredTotalRequestTime* is 0 (zero), each file request must be tried at least once. Negative value must be 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 must 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 must 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 supports, then all sub directories must be copied over from the source to the target, and complete sub-directory structure must 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 must return SRM_REQUEST_QUEUED and keep trying the request for the duration of *desiredTotalRequestTime* from the request. In the output parameter of explanation in *returnStatus*, the SRM may make explicit that the retry is being done. If streaming mode is not supported, the SRM must return SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests are successful) or SRM_FAILURE at the request level. Clients may need to clean up the target space when target space token is provided.
- v) Upon *srmCopy* request, *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* are 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.

6.7.3. 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 fails 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.

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 does not support the given input parameters. For example, client requests *desiredFileStorageType* that is not supported by the SRM.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM.
- Directory operation is not supported in the SRM.
- Recursive directory operation is not supported in the SRM.
- A specified input parameter is not supported in the SRM.
- *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 file. The *sourceSURL* 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 yet called) request for.
 - client requests for files at the target (*targetSURL*) which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) or *srmCopy* request 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 permitted.
- 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 provides 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.

6.8. srmStatusOfCopyRequest

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

6.8.1. Parameters

In:	string	<u>requestToken</u> ,
	string	authorizationID,
	anyURI []	arrayOfSourceSURLs,
	anyURI []	arrayOfTargetSURLs

Out: TReturnStatus	<u>returnStatus</u> ,
TCopyRequestFileStatus[]	arrayOfFileStatuses,
int	remainingTotalRequestTime

6.8.2. Notes on the Behavior

- a) If *arrayOfSourceSURLs* and/or *arrayOfTargetSURLs* are not provided, all file statuses in the request must be returned.
- 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 the request must be completed 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) Optional 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 must 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 must return SRM_REQUEST_QUEUED and keep trying the request for the duration of *desiredTotalRequestTime* from the request. *remainingTotalRequestTime* is being returned. In the output parameter of explanation in *returnStatus*, the SRM may make explicit that the retry is being done. If streaming mode is not supported, the SRM must return SRM_NO_USER_SPACE or SRM_NO_FREE_SPACE at the file level and SRM_PARTIAL_SUCCESS (if some file requests are 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 request, 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 are 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* must 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.

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

SRM_TOO_MANY_RESULTS

- Request produced too many results that SRM 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 URLs.

SRM_NO_USER_SPACE

- Insufficient space left in the user space.

SRM_NO_FREE_SPACE

- SRM space is not enough to hold requested file for free.

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 does not support the given input parameters. For example, client requests *bbftp* for the only transfer protocol, but SRM cannot support that. Client requests *desiredFileStorageType* that is not supported by the SRM.
- *targetFileRetentionPolicyInfo* does not refer to a supported retention policy in the SRM.
- Overwrite option is not supported in the SRM.
- Directory operation is not supported in the SRM.
- Recursive directory operation is not supported in the SRM.
- a specified input parameter is not supported in the SRM.
- *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 file. The *sourceSURL* is successfully copied into the target destination *targetSURL*, 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 yet called) request for.
- client requests for files at the target (*targetSURL*) which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called) or *srmCopy* request 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

- *sourceSURL* 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 provides 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.

6.9. 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* function.

6.9.1. Parameters

In:	string string anyURI [] Boolean	requestToken, authorizationID, arrayOfSURLs, doRemove
Out:	TReturnStatus	<u>returnStatus</u> ,

TSURLReturnStatus[] arrayOfFileStatuses

6.9.2. Notes on the Behavior

- a) *doRemove* by default is false. If *doRemove* is true, the pin on the file must be released, the “copy” or “state” must be removed. It must not remove SURLs, and *srmRm* must be used to remove SURLs.
- b) Directory is okay for SURL. In such case, it must release all files recursively in the directory.
- c) If *requestToken* is not provided and SURLs are provided, then the SRM must release all the files specified by the SURLs owned by the client, regardless of the *requestToken*.
- d) If *requestToken* is provided and SURLs are not provided, then the SRM must 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 request.
- g) *srmReleaseFiles* is only valid after successful *srmPrepareToGet* or *srmBringOnline* request. To release TURLs after a *srmPrepareToPut*, *srmAbortRequest* or *srmAbortFiles* must be used. If a client submits *srmReleaseFiles* after *srmPrepareToPut* or *srmPutDone*, then the SRM must return SRM_INVALID_REQUEST.

6.9.3. 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 fails 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.

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.

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 *doRemove* 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.

6.10. srmPutDone

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

6.10.1. Parameters

In:	string string anyURI []	<u>requestToken</u> , <u>authorizationID</u> , <u>arrayOfSURLs</u>
Out:	TReturnStatus TSURLReturnStatus[]	<u>returnStatus</u> , arrayOfFileStatuses

6.10.2. Notes on the Behavior

- a) Client calls srmPutDone after srmPrepareToPut() prepares the TURL and the client completes the file transfer into the prepared TURL.
- b) *srmRm* must 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 is 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.

6.10.3. 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 fails 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.
- 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 is performed on the *targetSURL*
- SRM_AUTHORIZATION_FAILURE
 - client is not authorized to call the request *srmPutDone()* on the *targetSURL*
- 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* has been aborted.
- SRM_FAILURE
 - any other request failure. *Explanation* needs to be filled for details.

6.11. 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 must attempt a complete cleanup of running transfers and files in intermediate state.

6.11.1. Parameters

In:	string	<u><i>requestToken</i></u>
	string	<i>authorizationID</i>

Out: TReturnStatus returnStatus

6.11.2. Notes on the Behavior

- a) `srmAbortRequest` terminates 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 must remain in the space where they are brought in and must not be 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 `srmPrepareToPut` request 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.

6.11.3. 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 are successfully completed already.

`SRM_AUTHENTICATION_FAILURE`

- SRM fails 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.

`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

6.12. 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.

6.12.1. Parameters

In:	string anyURI [] string	<u>requestToken</u> , <u>arrayOfSURLs</u> , authorizationID
Out:	TReturnStatus TSURLReturnStatus[]	<u>returnStatus</u> , arrayOfFileStatuses

6.12.2. Notes on the Behavior

- a) srmAbortFiles aborts all files in the request regardless of the state.
- b) When aborting srmCopy request, the srmCopy 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 request, all uncompleted files must be aborted, and all successfully completed files must be released.
- d) 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.
- e) 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.
- f) When aborting srmPrepareToPut request after srmPutDone, it must be failed for those files. An explicit srmRm is required to remove those successfully completed files for srmPrepareToPut.
- g) srmAbortFiles 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 must be returned to all duplicate abort file requests and no operations on duplicate abort file requests must performed.

6.12.3. 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 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.
- 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.

6.13. srmSuspendRequest

srmSuspendedRequest is used to suspend a previously submitted active request.

6.13.1. Parameters

In:	string	<u>requestToken</u>
	string	authorizationID
Out:	TReturnStatus	<u>returnStatus</u>

6.13.2. Notes on the Behavior

- a) Suspend all files in the request until srmResumeRequest or srmAbortRequest is requested.

6.13.3. Return Status Code

- SRM_SUCCESS
- successful request completion. Request is suspended successfully.
- SRM_AUTHENTICATION_FAILURE
- SRM fails 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.
- 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

6.14. srmResumeRequest

srmResumeRequest is used to resume previously suspended request.

6.14.1. Parameters

In: string requestToken,
 string authorizationID

Out: TReturnStatus returnStatus

6.14.2. Notes on the Behavior

- Resume the previously suspended request.

6.14.3. Return Status Code

SRM_SUCCESS

- successful request completion. Request is resumed successfully.

SRM_AUTHENTICATION_FAILURE

- SRM fails 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.

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

6.15. srmGetRequestSummary

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

6.15.1. Parameters

In: string [] arrayOfRequestTokens,
 string authorizationID

Out: TReturnStatus returnStatus,
 TRequestSummary[] arrayOfRequestSummaries

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

For each request level return status,

SRM_INVALID_REQUEST

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

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.

6.16. srmExtendFileLifeTime

`srmExtendFileLifeTime()` allows clients to extend lifetime of existing *SURLs* of volatile and durable file storage types or lifetime of pinned files (*TURLs* of the results of *srmPrepareToGet*, *srmPrepareToPut* or *srmBringOnline*).

6.16.1. Parameters

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

6.16.2. Notes on the Behavior

- a) `srmExtendFileLifeTime` allows to change only one type of 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. Either *newFileLifetime* or *newPinLifetime* must be provided. When both *newFileLifetime* and *newPinLifetime* are provided in the same request, the request must be invalid, and `SRM_INVALID_REQUEST` 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 may be limited by SRM according to its policies.
- f) If original lifetime is longer than the requested one, then the requested one must be assigned.
- g) When none of lifetime input parameters (*newPinLifetime* and *newFileLifetime*) is specified, the SRM must 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* for `srmPrepareToPut` request. 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) `srmExtendFileLifeTime` intends to negotiate a request of extension of file or pin lifetime. If input parameter *newFileLifetime* or *newPinLifetime* exceeds the remaining lifetime of the space where *SURLs* are, then `SRM_SUCCESS` is returned at the request and at the file level, and *TSURLLifetimeReturnStatus* must contain the remaining lifetime of the space.
- k) Lifetime extension must fail on *SURLs* when their status is `SRM_FILE_BUSY`.

6.16.3. 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 fails 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.
- *requestToken* is not provided, and extending pinning lifetime of *TURLs* associated with *SURLs* requires *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.

6.17. srmGetRequestTokens

`srmGetRequestTokens` retrieves request tokens for the client, using specified request description. This is to accommodate lost request tokens. This can also be used for getting all request tokens back that belong to the client.

6.17.1. Parameters

In:	string	userRequestDescription,
	string	authorizationID
Out:	TReturnStatus	<u>returnStatus</u> ,
	TRequestTokenReturn[]	arrayOfRequestTokens

6.17.2. Notes on the Behavior

- a) If *userRequestDescription* is null, *srmGetRequestTokens* must return all request tokens that belong to the client.
- b) If the client assigns the same description to multiple requests, the client may get back multiple request tokens each with the time the request is created.

6.17.3. Return Status Code

SRM_SUCCESS

- successful request completion. Request tokens are returned successfully.

SRM_AUTHENTICATION_FAILURE

- SRM fails to authenticate the client

SRM_AUTHORIZATION_FAILURE

- client is not authorized 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

7. Discovery Functions

summary:

[srmGetTransferProtocols](#)
[srmPing](#)

7.1. srmGetTransferProtocols

srmGetTransferProtocols is used to discover what transfer protocols are supported by the SRM.

7.1.1. Parameters

In:	string	authorizationID
Out:	TReturnStatus TSupportedTransferProtocol[]	<u>returnStatus</u> , protocolInfo

7.1.2. Notes on the Behavior

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

7.1.3. Return Status Code

SRM_SUCCESS

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

SRM_AUTHENTICATION_FAILURE

- SRM fails to authenticate the client

SRM_AUTHORIZATION_FAILURE

- client is not authorized to request transfer protocol 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.

7.2. srmPing

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

7.2.1. Parameters

In:	string	authorizationID
Out:	string TExtraInfo[]	<u>versionInfo</u> , otherInfo

7.2.2. 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*.

8. Appendix I : Current SRM Implementations Based on v2.2 specification

Over the last 6-7 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 implementations 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.

8.1. 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 gridftp server, or another SRM. BeStMan does not support ACLs.

8.2. Castor-SRM

The SRM implementation for the CERN Advanced Storage system (CASTOR) is the result of collaboration between Rutherford Appleton Laboratory (RAL) and CERN. Like that of other implementations, the implementation faced unique challenges. These challenges were based around the fundamental design concepts which CASTOR operates under, which are different from those of other mass storage systems. CASTOR trades some flexibility for performance, and this requires 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.

8.3. dCache-SRM

dCache [dcache] is a Mass Storage System developed jointly by FNAL 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, GSIDcap, dCap, xrootd [xrootd], and POSIX APIs to dCache. dCache can act as a stand-alone 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.

8.4. 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.

8.5. 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.

9. Appendix II : WLCG use case

Introduction

The European research centre CERN currently builds the Large Hadron Collider (LHC) to allow for the study of elementary particles and forces at the highest energy densities ever obtained in a laboratory.

The LHC will support four experiments that are expected to produce roughly 15 Petabytes (15 million Gigabytes) of data annually, which thousands of scientists around the world will access and analyze. The mission of the Worldwide LHC Computing Grid Project (WLCG) is to develop, build and maintain a computing infrastructure for storage and analysis of the LHC data. Data will be distributed around the globe according to a four-tiered model. A primary backup will be recorded at CERN, the "Tier-0" centre of WLCG. After initial processing, these data will be distributed to a series of Tier-1 centers, large computer centers with sufficient compute and storage capacities as well as round-the-clock support for Grid operations. The Tier-1 centers will perform recurrent reconstruction passes and make data available to Tier-2 centers, each consisting of one or several collaborating computing facilities, which can store sufficient data and provide adequate computing power for specific analysis tasks. Individual scientists will access these facilities through Tier-3 computing resources, which can consist of local clusters in a University Department or even individual PCs [WLCG-INTRO].

The WLCG had to deal with multiple storage systems and using the SRM concept of a uniform layer on top of these storage systems is appealing. A decision was made to adopt the SRM v2.2 specification, as well as to contribute to its enhancement in terms of functionality and clarity of the specification. These enhancements are reflected in this document. It was also decided that test programs will be developed to check the adherence of various implementations to the SRM v2.2 specification, as well as the interoperability of the implementations. Several implementations have been developed as well as test suites and were recently reported in the Mass Storage Systems and Technologies conference [MSST 2007].

In order to meet the timescale for LHC experiments' requirements for grid storage, the WLCG has had to agree on certain restrictions and interpretations of the SRM v2.2 specification. This Appendix specifies the server behavior for the SRM v2.2 implementations used by WLCG applications. It specifies how the various existing methods and data objects, as they are specified in the agreed v2.2 specification, should be used in order meet the WLCG requirements. The outlined WLCG interpretation of the interfaces is somewhat more restrictive than other implementations of SRM client and servers, although care is taken to preserve interoperability with the latter.

9.1. Storage classes

One of the concepts that need to be supported for large storage systems is how to interpret storage of files in systems that have both tape and disk storage. The descriptive behavior often refers to that as "how many copies are kept on tape and how many on disk. The label "TapeXDiskY" was used to express X copies on tape and Y copies on disk. In practice, X and Y assume values of 0 or 1, and rarely 2 (2 copies for storing invaluable data). There was a need to map these concepts to the more general features of Access Latency and Retention Policy in the SRM specification. This is described next.

Wherever usage of storage class argument is agreed to be required by WLCG, the agreed terminology is:

- TAccessLatency {ONLINE, NEARLINE}
- TRetentionPolicy {REPLICA, CUSTODIAL}

The "OUTPUT" value of the TRetentionPolicy as well as the "OFFLINE" value of the TAccessLatency shall not be used by any WLCG application. The mapping to labels is given by:

- Tape1Disk0: NEARLINE + CUSTODIAL
- Tape1Disk1: ONLINE + CUSTODIAL
- Tape0Disk1: ONLINE + REPLICA

WLCG clients shall only use those three combinations of *TAccessLatency* and *TRetentionPolicy*. In all subsequent methods listed in the following, the allowed/disallowed usage of storage classes attributes is specified as needed.

9.2. Removal policies

All WLCG files are “permanent” meaning that any implicit automatic removal of the file from the storage system is disallowed. The file lifecycle within an SRM is always terminated by the user calling the *srmRm* method.

9.3. Protocol negotiation

The protocols are specified in the preferred order on all SRM transfer functions (*srmPrepareToGet*, *srmPrepareToPut*, *srmBringOnline*).

9.4. Information discovery

- There is no requirement for an SRM information interface for WLCG production.
- Client implementations provide list of required information.
- GLUE WG [glue-wg] defines an external information system schema accordingly.
- In the longer term, implementations may provide the interface to query for the information required for the GLUE schema.

9.5. *srmReserveSpace*

For practical use of WLCG, *srmReserveSpace* only deals with space reservations on disk cache while tape space is always considered infinite. SRM implementations may allow dynamic space reservations for any user while other implementations may only allow static reservations restricted for use by certain VO (Virtual Organization) roles. For static reservations, detailed requirements for the reservation, like network connectivity, may be negotiated directly between storage provider and client.

- WLCG clients shall only specify *desiredSizeOfGuaranteedSpace* while *desiredSizeOfTotalSpace* shall be ignored by all WLCG servers.
- WLCG server implementations are allowed to restrict permission to reserve space to certain VO roles. The WLCG client shall handle the corresponding error: SRM_AUTHORIZATION_FAILURE.
- The *arrayOfExpectedFileSizes* shall not be used.
- If the requested *retentionPolicyInfo* cannot be satisfied, SRM_INVALID_REQUEST error code shall be returned.
- WLCG clients shall not specify *storageSystemInfo*, and servers shall ignore any value provided in this parameter.
- SRM implementation may return infinite lifetime no matter what was specified by the user. It is then the client’s responsibility to remove the space.
- WLCG clients shall use *srmGetSpaceTokens* method with the user space description to get to the space token.
- WLCG clients shall not use dynamic space reservations initially.

9.6. *srmChangeSpaceForFiles*

The common use case for this method is the transition of Tape1Disk1 -> Tape1Disk0 or vice versa (both supported in WLCG), or change of retention policy Tape0Disk1 -> Tape1Disk[0-1] or vice versa (neither supported in WLCG).

- *srmChangeSpaceForFiles* is asynchronous.
- For any transition forbidden by the SRM implementation, an error SRM_INVALID_REQUEST is returned to the client.
- *srmChangeSpaceForFiles* is not required for the start LHC production but is needed later.
- WLCG SRM implementations shall not change the SURL when a file is moved to a new space.
- The signature of *srmStatusOfChangeSpaceForFilesRequest* remains to be discussed between the SRM implementers.
- WLCG SRM implementations and clients shall decide which transitions should be supported.

9.7. srmPurgeFromSpace

Purges files from the given space. The difference from *srmReleaseFiles* and *srmAbortFiles* is that *srmPurgeFromSpace* is not associated with a request. *srmReleaseFiles* is enhanced with a flag “doRemove” giving a hint for the immediate garbage collection of disk copies.

- If the specified space contains the only remaining copy of the file in the storage system, an error SRM_LAST_COPY is returned.
- The method shall only succeed if there are no outstanding pins or requests for the specified files. Otherwise an error SRM_FILE_BUSY is returned.

9.8. srmRm

Removes name space entry and all associated copies of the file.

9.9. srmLs

WLCG clients use *srmLs* for getting metadata attributes of individual files. Support for directory listings can be motivated for cross-checking file catalogues.

- *TFileLocality* {ONLINE, NEARLINE, ONLINE_AND_NEARLINE, LOST, NONE, UNAVAILABLE} attribute of the *srmLs* output shall be supported for individual file queries but not for directory listings, even when a detailed directory listing has been requested by the client.
- *spaceToken* attribute of the *srmLs* output shall be supported for individual file queries but not for directory listings, even when a detailed directory listing has been requested by the client.
- The “LOST” value of *TFileLocality* shall be used if the file is lost because of a permanent hardware failure. The “NONE” value shall be used if the file is empty (zero size). The “UNAVAILABLE” value shall be used if the file is unavailable due to temporary hardware failure.
- For directory listings, *srmLs* output may be constrained to an upper limit on the number of entries returned.
- *srmLs* for a non-existing file or directory shall result in an SRM_INVALID_PATH error
- If *numberOfLevels* is specified to be zero, the returned information shall be for the directory itself and not for the files in the directory.

Support for *offset* and *count* in the *srmLs* function at the timescale for LHC production will require more discussion. Currently no WLCG implementation supports it.

9.10. srmPrepareToGet

srmPrepareToGet is used for preparing a file for transfer or access. WLCG usage excludes the usage of *srmPrepareToGet* for changing space or retention attributes of the file.

- WLCG tools/clients shall not use space tokens, or retention policy/access latency on a *srmPrepareToGet*.
- *transferParameters* for transfer protocols is a required input parameter.
- *srmPrepareToGet* is asynchronous.
- It is up to the storage system to decide the *TConnectionType* of *TTransferParameters* if not provided.
- WLCG clients shall not have expectations of any value returned for the *estimatedWaitTime*.
- If any of the requested files is temporarily unavailable (e.g. due to hardware failure), an error SRM_FILE_UNAVAILABLE is returned for that file.
- If any of the requested files is permanently lost (e.g. due to hardware failure), an error SRM_FILE_LOST is returned for that file.
- The SRM implementation should fail with an error SRM_FILE_BUSY for a file for which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called).
- If the request does not complete within the *desiredTotalRequestTime*, the request will fail with SRM_TIMEDOUT. The *desiredTotalRequestTime* can be negotiated with the SRM implementation. This functionality is desired for the longer term but not required for the start of LHC production.
- The WLCG SRM implementations shall allow an *srmPrepareToGet* request to continue, provided that at least one file is successful or in progress.

9.11. srmBringOnline

- *srmBringOnline* is asynchronous.
- Transfer protocol matching shall work like for *srmPrepareToGet*.
- The *deferredStartTime* input parameter means that the client does not intend to use the files before that time. The purpose is to be able to support CE – SE resource co-allocations and tape mounting efficiency.
- WLCG tools/clients shall not use *targetSpaceToken* or *targetFileRetentionPolicyInfo* on a *srmBringOnline*.
- If the transfer protocol list in *transferParameters* is not specified, the default is assumed to be the {PROCESSING_MODE, LAN} for the *TAccessPattern* and *TConnectionType*. WLCG clients are strongly recommended to specify the transfer protocols.
- It is up to the storage system to decide the *TConnectionType* if not provided.
- WLCG clients shall not have expectations of any value returned for the *estimatedWaitTime*.
- If any of the requested files is temporarily unavailable (e.g. due to hardware failure), an error SRM_FILE_UNAVAILABLE is returned for that file.
- If any of the requested files is permanently lost (e.g. due to hardware failure), an error SRM_FILE_LOST is returned for that file.
- The SRM implementations should fail with an error SRM_FILE_BUSY for a file for which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called).
- If the request does not complete within the *desiredTotalRequestTime*, the request will fail with SRM_TIMEDOUT. The *desiredTotalRequestTime* is negotiated with the SRM implementation. This functionality is desired for the longer term but not required for the start of LHC production.

- The WLCG SRM implementations shall allow an *srmBringOnline* request to continue provided that at least one file is successful or in progress.

9.12. srmPrepareToPut

- *transferParameters* for transfer protocol is a required input parameter.
- *srmPrepareToPut* is asynchronous.
- It is up to the storage system to decide the *TConnectionType* of *TTransferParameters* if not provided.
- WLCG clients shall not have expectations of any value returned for the *estimatedWaitTime*.
- *transferProtocolInfo* of *TPutRequestFileStatus* output parameter will initially not be supported for WLCG production.
- The WLCG client shall provide *targetURLs* as input parameters, and hence SRM implementations are not required to generate names.
- If the request does not complete within the *desiredTotalRequestTime*, the request will fail with SRM_TIMEDOUT. The *desiredTotalRequestTime* is negotiated with the SRM implementation. This functionality is desired for the longer term, but not required for the start of LHC production.
- The WLCG SRM implementations shall allow an *srmPrepareToPut* request to continue, provided that at least one file is successful or in progress.
- If both *targetSpaceToken* and *targetFileRetentionPolicyInfo* are provided, the retention policy attribute must match that of the specified space. Otherwise the request fails with SRM_INVALID_REQUEST.
- WLCG clients shall use the *targetSpaceToken*, but not the *targetFileRetentionPolicyInfo*. If the *targetSpaceToken* is not provided, a default space will be chosen by the SRM.
- For non-WLCG implementations, *desiredFileLifetime* for SURL shall not be used by WLCG clients, and WLCG SRM implementations are not required to support it.
- The *desiredPinLifetime* for TURL is the allowed time during which the file may be written; there is no guarantee as to the availability of the TURL for writing after this time.
- A TURL returned by *srmPrepareToPut* may not be used for read access with any protocols. An explicit *srmPrepareToGet* call is required.
- The SRM implementation should fail with an error SRM_FILE_BUSY for a file for which there is an active *srmPrepareToPut* (no *srmPutDone* is yet called).

9.13. srmCopy

- The *dirOption* in *TCopyFileRequest* will not be supported initially.
- The *desiredTargetSURLLifetime* shall not be used by WLCG clients, and WLCG SRM implementations are not required to support it.
- Files are immutable. WLCG clients shall not specify *overwriteOption*. If SRM returns failure, the state of the system shall be as if the file transfer did not take place.
- If the request does not complete within the *desiredTotalRequestTime*, the request will fail with SRM_TIMEDOUT. The *desiredTotalRequestTime* is negotiated with the SRM implementation. This functionality is desired for the longer term but not required for the start of LHC production.
- WLCG clients shall use the *targetSpaceToken* and not the *targetFileRetentionPolicyInfo*. If the *targetSpaceToken* is not provided, a default space will be chosen by the SRM.

- The SURLs passed in a call to the *srAbortFiles* method following an *srCopy* are the target SURLs, which are bound to be unique.
- The SRM implementation should fail with an error SRM_FILE_BUSY for a file for which there is an active *srPrepareToPut* (no *srPutDone* is yet called), considering both source and target.

10. 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.

11. Contributors

11.1. 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

11.2. Authors

Authors of this document are people who have been involved in SRM specification over the years. The author list includes, by institution:

Paolo Badino, Olof Barring, Jean-Philippe Baud, Flavia Donno, Maarten Litmaath
European Organization for Nuclear Research (CERN) Switzerland

Timur Perelmutov, Don Petravick
Fermi National Accelerator Laboratory (FNAL) USA

Ezio Corso
International Centre for Theoretical Physics (ICTO) Italy

Luca Magnoni
Istituto Nazionale di Fisica Nucleare (INFN) Italy

Junmin Gu, Arie Shoshani, Alex Sim
Lawrence Berkeley National Laboratory (LBNL) USA

Shaun De Witt, Jens Jensen
Rutherford Appleton Laboratory (RAL) UK

Michael Haddox-Schatz, Bryan Hess, Andy Kowalski, Chip Watson
Thomas Jefferson National Accelerator Facility (TJNAF) USA

11.3. 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. This work is also co-funded by the European Commission through the EGEE-II project, contract number INFISO-RI-031688.

12. 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.

13. 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.

14. Full Copyright Notice

Copyright (C) Open Grid Forum (2007,2008). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the OGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the OGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the OGF or its successors or assignees.

15. 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, B.K. Hess, M. Haddox-Schatz, M.A. Kowalski, Proceedings of 13th NASA Goddard Conference on Mass Storage Systems, April 2005.

[glite] <http://glite.web.cern.ch/glite/>
[glue-wg] <http://forge.ogf.org/sf/projects/glue-wg>
[hpss] <http://www.hpss-collaboration.org/hpss/index.jsp>
[ISGC2007] A Model for the Storage Resource Manager, A. Domenici, F. Donno, International Symposium on Grid Computing (ISGC 2007), <http://event.twgrid.org/isgc2007/program.htm>
[l-store] www.lstore.org/index.php
[MD'07] The S2 testing suite, J. Mencak, F. Donno, <http://s-2.sourceforge.net>
[MSST2007] Storage Resource Managers: Recent International Experience on Requirements and Multiple Co-Operating Implementations, MSST 2007: 47-59
[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] Storage Resource Managers: Middleware Components for Grid Storage, A. Shoshani, A. Sim, J. Gu, 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
[storm] <http://storm.forge.cnaf.infn.it/>
[voms] The Virtual Organization Membership Service,
http://www.globus.org/grid_software/security/voms.php
[WLCG-INTRO] largely quoted from
<http://www.fzk.de/fzk/idcplg?IdcService=FZK&node=2899&lang=en>
[wlcg-collab] <http://cerncourier.com/cws/article/cern/29856>
[xrootd] <http://xrootd.slac.stanford.edu/>
[XSDDATE] <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/#dateTime>