Table of Contents

Issues in the SRM v2.2 Specifications

In what follows we list the issues found in the SRM v2.2 specifications that are published here.

1. SRM_DONE is not used and should be removed from the WSDL
   - **It won't be fixed in 2.2**
2. The Overwrite Mode WHEN_FILES_ARE_DIFFERENT is not supported by any implementation. What is the behavior foreseen ?
   - This functionality should not be implemented at the moment. Files can be different when the declared size for a SURL differs from the actual one
3. **srmLs** call can return SRM_REQUEST_QUEUED at file level.
   - It will be fixed in the spec for v2.2.
   - Condition checked in S2 test suite, ***basic*** tests
4. No methods allow getting the associated description (if it exists), given a space or request token.
   - **It won't be fixed in v2.2**
5. No methods allow for the retrieval of all space token descriptions for a given VO or FQAN.
   - **It won't be fixed in v2.2**
6. No methods allow for the listing of the content of a space.
   - **It won't be fixed in v2.2**
7. The meaning of the fields of **TRequestSummary** data structure needs clarification. In particular the fields: **numOfWaitingFiles**, **numOfFailedFiles**, **numOfCompletedFiles**
   - Under discussion
8. If the pin on a TURL is still not expired and the overwrite option was specified in the method that created the file (either **srmPrepareToPut** or **srmCopy**), then users can overwrite the files (which TURLs would they use since the TURL resulting from an **srmPrepareToPut** or **srmCopy** is not available/not valid ?). Once open sessions are completed, all other (disk) copies will be marked as invalid. The overwrite option does not imply a removal of the SURL. The file will stay in the namespace but the content will change.
   - Still under discussion
   - Proposal: The overwrite mode is considered more powerful over pin. It is allowed to mark an valid TURL invalid when the owner of the surl issues an overwrite command, just like it is OK to call srmRm() and cause the valid TURLs be erased.
9. **srmExtendFileLifeTime** has an ambiguity and allows for the following 2 possibilities in case the **newTimeExtended** exceeds the remaining lifetime of the space:
   1. Return PARTIAL_SUCCESS at the request level and SRM_FAILURE at the file level and the **TSURLLifetimeReturnStatus** returns the remaining lifetime.
   2. SUCCESS at the request and file level and **TSURLLifetimeReturnStatus** contains the remaining lifetime. Case b is the correct one and should be specified in the spec.
   - This will be added to the spec for v2.2.
10. It must be possible to list the TOP directory. We can impose that Directory SURLs must end with a "/", if this makes the implementation easy.
    - Still under discussion.
11. On page 54 point h) is empty
    - It will be corrected in the spec doc.
12. Specify what values can **lifetimeLeft** and **lifetimeAssigned** assume in **TMetaDataPathDetail** and in **TMetaDataSpace** after the execution of an Ls. Are 0 and -1 allowed values ? What is their meaning ? Same for **remainingPinLifetime** and **remainingFileLifetime** in **TPutRequestFileStatus**, **TCopyRequestFileStatus**, **TSURLLifetimeReturnStatus** and everytime the Lifetime appears as an output paramenter: **srmReserveSpace**, **srmUpdateSpace**, **srmStatusOfUpdateSpaceRequest**, **srmStatusOfReserveSpaceRequest**, **srmPrepareToGet**, **srmStatusOfPrepareToGetRequest**
    - This will be made explicit in the spec.
13. What is the lifetime assumed in **srmUpdateSpace** if the newLifetime is unspecified ? The default or infinite ? **srmExtendFileLifetimeInSpace** unspecify newLifetime means default, for **srmReserveSpace** it means infinite.

- Under discussion
- It has been proposed the following: if lifetime is not given or zero, the default lifetime used is "Infinite" if the request comes from the Storage Element admin or "one day" otherwise. The behaviour should be the same for **srmReserveSpace**, **srmUpdateSpace**. The behaviour in case of **srmExtendFileLifetimeInSpace** should be as of now.
- All lifetimes should be assumed to be the default if left unspecified.

14. What are the possible values for **remainingTotalRequestTime** and **remainingDeferredStartTime** in **srmBringOnline** ? They are declared as int. Are negative values possible ?
    - Waiting for an answer.
15. In the output of the **srmLs** for a file when **fullDetailedList** is true, only the fields **path**, **size**, **userPermission**, **lastModificationTime**, **file type** and **lifetimeLeft** must be returned. Should **fileStorageType**, **retentionPolicyInfo** and **fileLocality** be returned as well?
    - Waiting for an answer.
16. The copy from LBNL to FNAL in push mode fails because the FNAL server returns SRM_FILE_IN_CACHE, which is not a valid Copy status at the file level. When overwriting is not set and the SURL exists at the target, the target SRM may say duplication error (if lifetime is still valid) or file_in_cache (if lifetime is expired). If lifetime is expired at the target there is a conflict since **srmCopy** says SRM_LIFETIME_EXPIRED and **srmPrepareToPut** says SRM_FILE_IN_CACHE at the file level. What should be the correct behaviour ?
    - Waiting for an answer.
17. What are the possible values for **remainingTotalRequestTime** and **remainingDeferredStartTime** in **BringOnline** ? They are declared as int. Are negative values possible ?
    - Waiting for an answer.
    - Proposal from Alex:"Negative time in our spec means "indefinite" time, and we cannot have remainingDeferredStartTime as negative. **remainingTotalRequestTime** may be. If request will be tried, by default, until all files in the request will be completed, the server may return negative since there is no 'expiration time' of the request... ? "
18. What is the actual meaning of the field lifetimeAssigned present in the output of an srmLs or srmGetSpaceMetaData request?
    - Waiting for an answer.
    - Proposal from Alex:"It was intended to hold the original lifetime that was assigned upon the request. [lifetimeAssigned - lifetimeLeft] = time passed so far since the assignment. The question comes then when lifetime got extended and what to assign on this lifetimeAssigned...? How about we define this lifetimeAssigned as the full amount of lifetime on the file before decreased by the passing time? Then, it can hold all lifetime extention."
19. Since the recursive **srmLs** has the major problem that the number of files returned is limited, it is proposed not to support it for the moment.
    - Under discussion.
20. A recursive Rmdir cannot remove the files contained in the subdirectories, but it can only remove subdirectories. Is this useful ? Do we have a use case for it ? There is a proposal to drop support for recursive Rmdir.
    - Under discussion.
21. After a put cycle, in the space there is a copy of the file even if the handle (TURL) is expired/gone. What is its lifetime ? This has implications on **srmReleaseSpace** of that space.
    - Under discussion
    - Proposal by Jean-Philippe: a copy of the file stays in space and its lifetime equals the SURL lifetime.
    - To be deferred to v3.0
22. The following case is undocumented for the request level status code after an **srmAbortFiles** is executed:
    - Proposal from Junmin: if files in a request are some aborted, some successful, then request level is SRM_PARTIAL_SUCCESS; if files in a request are either aborted or failed, then request level is SRM_FAILURE
    - Proposal from Jean-Philippe: An **srmAbortFiles** should not change the request level status

code for the requests.
- Under discussion

23. What should be the behaviour of the system after **srmAbortRequest** is issued for files that are already completed ?
    - The spec at the moment foresees that pin lifetimes are set to 0 and file lifetimes are set to 0 for volatile files (PrepareToPut/PutDone or Copy)
    - The proposal is to do nothing on completed files
    - Under discussion
24. What is the default value for the parameter **overwriteOption** in an **srmPrepareToPut** request ?
    - Proposal (Junmin Gu): it is left up to the implementation
    - It will be documented in the 2.2 spec 1. It should be clarified the behaviour of the system after an **srmAbortRequest** is issued.
    - Proposal (Paolo Badino): if all the files are rather completed or aborted, this method should return SUCCESS: the server correctly cleaned up the request and there's nothing more the user should do to release the resources associated with the request. Eventually, the client has to call srmRm to delete the completed files.
    - It will be documented in the 2.2 spec
25. It should be explicitely said that in **srmExtendFileLifeTime** a request that specifies both file and pin lifetime is invalid.
    - It will be made explicit in the 2.2 spec
26. The function of the paramenter **overwriteOption** in an **srmPrepareToPut** should be clarified. It is valid at a request level and it does not have any permission meaning. It is the equivalent of the -f (force) option in the UNIX commands mkdir or cp.
    - **This will be clarified in SRM v3**
27. The behaviour of **srmAbortRequest** on a copy request should be clarified. In COPY cases, either source or target has to be local to the SRM server. There can be one source SURL and multiple target SURLs in a request, regardless of PULL or PUSH. The question is how do we abort the request by the file, by the source or the target? If we abort by the source SURL, all file transfer of the same source SURL will be aborted. If we abort by the target SURL, just the particular target file operation will be aborted and others from the same source will not be aborted. Therefor either matching will be honored.
    - This will be clarified in the 2.2 spec
    - This is tested in the USECASE S2 testsuite
28. In **srmAbortRequest**, note a. states that "Expired files are released.". This is not an effect of the **srmAbortRequest**. This sentence must be removed.
    - This will be fixed in the 2.2 spec
29. If a particular combination of Retention Policy and Access Latency is not supported in **srmReserveSpace** the server should return **SRM_NOT_SUPPORTED**
    - This will be clarified in the 2.2 spec
    - This is tested in the USECASE S2 testsuite
30. A general behaviour of all srm method is that if a particular server does not support a particular optional parameter, **SRM_NOT_SUPPORTED** must be returned.
    - This will be clarified in the 2.2 spec

-- Flavia Donno - 17 January 2007

This topic: SRMDev > IssuesInTheSpecifications
Topic revision: r21 - 2007-01-30 - FlaviaDonno

Copyright &© 2008-2024 by the contributing authors. All material on this collaboration platform is the property of the contributing authors.
or Ideas, requests, problems regarding TWiki? use Discourse or Send feedback