# LTFSArchiver 1.1 Interface specification

Summary

[LTFSArchiver 1.1 Interface specification 1](#_Toc365990247)

[Interface Specification 3](#_Toc365990248)

[1. Managing the tapes (TapeManager) 4](#_Toc365990249)

[Add command 4](#_Toc365990250)

[GetStatus command 7](#_Toc365990251)

[GetResult command 7](#_Toc365990252)

[Cancel command 7](#_Toc365990253)

[Withdraw command 8](#_Toc365990254)

[2. Querying the tapes (QueryKnownTapes) 8](#_Toc365990255)

[3. Querying free spaces (QueryFreeSpaces) 12](#_Toc365990256)

[4. Archiving operations (WriteToLTO) 14](#_Toc365990257)

[WriteFile / WriteFolder commands 15](#_Toc365990258)

[Cancel command 18](#_Toc365990259)

[Resubmit command 18](#_Toc365990260)

[GetStatus command 18](#_Toc365990261)

[GetResult command 19](#_Toc365990262)

[5. Restoring operations (RestoreFromLTO) 20](#_Toc365990263)

[RestoreFile command 20](#_Toc365990264)

[GetStatus, Cancel and Resubmit commands 21](#_Toc365990265)

[GetResult command 21](#_Toc365990266)

[6. Giving access through LTFS direct access (MakeAvailable) 22](#_Toc365990267)

[Mount command 23](#_Toc365990268)

[GetStatus command 23](#_Toc365990269)

[GetResult command 24](#_Toc365990270)

[Cancel command 25](#_Toc365990271)

[Unmount command 25](#_Toc365990272)

[Appendix A - Status and substatus description 27](#_Toc365990273)

[Appendix B – Archiving file commands and reports samples 28](#_Toc365990274)

[Submitting an archive task 28](#_Toc365990275)

[Monitoring an archive task 28](#_Toc365990276)

[Getting the final result of the task 29](#_Toc365990277)

[Different reports according to Checksum parameter use. 30](#_Toc365990278)

**Scope:**

This document describes in detail the functionalities of the programming interface (API) used to interact with the LTFSArchiver agent. Through this interface LTFSarchiver can be easily integrated in wider systems.

## Interface Specification

The interface is made of a set of *http* services that allows to:

* Manage the tapes
* Query the tapes inside the system
* Query the free space of each tape pool
* Book read/write requests and follow their current status and result
* Directly access the tapes content through LTFS (makeavailable)

## Managing the tapes (TapeManager)

Service URL:

http://<servername>/pprime/cgi-bin/TapeManager

It is used to add / remove LTO tapes to / from tape pools.

In Table 1 are shown all the possible combinations of commands and parameters, N/A means that the parameter is not applicable to that command.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Command** | **Parameters** | | | |
|  | **TapeID** | **PoolName** | **Format** | **TaskID** |
| **Add** | Mandatory | Optional | Optional | N/A |
| **Withdraw** | Mandatory | Mandatory | N/A | N/A |
| **GetStatus (Add only)** | N/A | N/A | N/A | Mandatory |
| **GetResult (Add only)** | N/A | N/A | N/A | Mandatory |
| **Cancel (Add only)** | N/A | N/A | N/A | Mandatory |

Table 1 - TapeManager commands and related parameters

All of the answers from the service are plain text, according to the following format:

RC<TAB>RESULT<TAB>Comment

Where RC will be one of the following:

200 -> Request satisfied

400 -> Bad request

500 -> System error

### Add command

The Add command is used to assign a new LTO tape to a pool.

TapeID parameter

TapeID must be a unique string with a maximum length of eight characters. The system DOESN'T allow to add more tapes with the same label, an attempt to do so will result in an error.

PoolName parameter

If PoolName value is not supplied, the newly added tape will be automatically assigned to the special default pool named “default”.

If the Poolname supplied does not yet exist, it will be automatically created.

Format parameter

The only valid values are “Y” (yes) and “F” (force), in any other case (unused option or option values different from “Y” and “F”), it will be set to “No”

Notes about format option

If the user wants to add a tape that already contains a LTFS file system to be preserved, choose “No”.

As soon as possible, the tape will be mounted to:

* check Mechanical un-protection
* evaluate actual available space (in case it is not empty)

If the user wants to add a blank tape to create a new LTFS files system on it, choose “Y”.

As soon as possible, the tape will be mounted and an LTFS file system will be mounted to:

* check mechanical un-protection
* create an LTFS file system on it according to the configuration parameter PPRIMELTO\_LTFSRULE (see documentation file LTFSArchiverConfiguration)

If the user wants to add a tape regardless his current content and create a new LTFS files system on it, choose “F”.

As soon as possible, the tape will be mounted and an LTFS file system will be mounted to:

* check mechanical un-protection
* create / recreate an LTFS file system on it according to the configuration parameter PPRIMELTO\_LTFSRULE (see documentation file LTFSArchiverConfiguration)

**Please note:**

Don’t choose “Yes” if the tape already contains a LTFS file system, or the format will fail: as a security feature, the creation of the LTFS is executed in “non-forced” mode, in order to avoid loss of pre-existing data. If you decide to use a previously LTFS formatted tape, please run the *unltfs* utility to clear it before adding the tape to a pool or set format parameter to “F”.

Example:

<http://servername/ltfsarchiver/cgi-bin/TapeManager?Command=Add&TapeID=NS04017B&PoolName=PoolB>

Will return

200<TAB>assigned-uuid<TAB>LTO with label: NS04017B added to system and assigned to pool: PoolB

where assigned-uuid is a unique id (e.g. 2f2437e2-7565-463c-aa0c-5755b254e24b) for the operation just taken in charge.

In case of error the client will get a 400/500 exit code with a description of the error occurred.

### GetStatus command

This command returns the status of an Add operation supplying the assigned-uuid with the parameter TaskID.

Example:

<http://servername/pprime/cgi-bin/TapeManager?Command=GetStatus&TaskID=2f2437e2-7565-463c-aa0c-5755b254e24b>

The answer when the operation is not finished yet will be like:

200<TAB>starting<TAB>Tape being loaded o positioning

When the task has completed, the answer is:

200<TAB>completed

See appendix A for a comprehensive list of possible status and descriptions.

### GetResult command

This command provides full information about the TapeManager/Add procedure completion.

Example:

<http://servername/pprime/cgi-bin/TapeManager?Command=GetStatus&TaskID=2f2437e2-7565-463c-aa0c-5755b254e24b>

200<TAB>Succesful

### Cancel command

This command allows to delete a previously submitted Add operation; assigned uuid must be supplied.

An operation can be deleted only if it is still waiting for some resource assignment.

Example:

<http://10.58.78.112/ltfsarchiver/cgi-bin/TapeManager?Command=Cancel&TaskID=e55c4fed-55a8-4459-bf68-ffe2ce52dc8b>

200<TAB>e55c4fed-55a8-4459-bf68-ffe2ce52dc8b removed from working queue

A successful cancel request will also remove the added tape from the assigned Pool.

### 

### Withdraw command

TheWithdraw command is used to remove a tape from a pool. Both LTOLabel and PoolName parameters are requested and must match with an existing label and pool.

Removing a tape that is currently in use or that has been booked for further use is not allowed (an error message will be returned).

Example

<http://servername/pprime/cgi-bin/TapeManager?Command==Withdraw&TapeID=NS04017B>&PoolName=PoolB

200<TAB>LTO with label: NS04017B has been deleted from pool: PoolB

## Querying the tapes (QueryKnownTapes)

Service URL:

http://<servername>/ltfsarchiver/cgi-bin/QueryKnownTapes

It is used to list pools and their associated tapes known by a specific LTFSArchiver instance, retrieved information also includes the remaining free space in Mbytes for each tape.LTFSArchiver can be deployed with multiple instances (on different servers), in this case the client application needs to know the URLs of all possible LTFSArchiver instances to be used.

Following Table2 reports the possible parameters of QueryKnownTapes.

|  |  |  |
| --- | --- | --- |
| **Parameters** | | |
| **TapeID** | **PoolName** | **Output** |
| Optional | Optional | Optional |

Table 2 -QueryKnownTapes commands and related parameters

TapeID parameter

It has to be supplied to look for a specific tape, if only a part of the label is given it will be used as pattern matching. If not supplied, all the tapes are reported.

PoolName parameter

By supplying the PoolName parameter, the search will look only for tapes in the specified pool. The given value must be the exact pool name (no pattern matching is allowed). If not supplied, all the pools are considered.

If neither TapeID nor PoolName are given, all the pools/tapes known by the system are reported.

Output parameter

If omitted, the output will be formatted in an HTML table.

Accepted other values are TEXT and JSON.

Examples of text output

[http://servername/pprime/cgi-bin/QueryKnownTapes?PoolName=TestExt&Output=TEXT](http://10.2.7.141/pprime/cgi-bin/QueryKnownTapes?PoolName=TestExt&Output=T)

200<TAB>(EX000001,140000,TestExt,LTO5)<TAB>(EX000002,90560,TestExt,LTO5)

Where:

200 means that at least one tape has been found.

For each tape the following four values in the order are given: Label, Free space (in MBytes), Poolname, LTOtype

<http://servername/pprime/cgi-bin/QueryKnownTapes?PoolName=ThisDoesNotExist&Output=TEXT>

400<TAB>No tape found matching criteria

Examples of json output

<http://10.2.7.141/pprime/cgi-bin/QueryKnownTapes?Output=JSON>

{

"exit\_code":"200",

"output":[

{

"TapeID":"000002L5",

"FreeSpace":"1443596",

"PoolName":"poolB",

"LTOtype":"LTO5"

},

{

"TapeID":"EX000001",

"FreeSpace":"1444049",

"PoolName":"poolC",

"LTOtype":"LTO5"

},

{

"TapeID":"000001L5",

"FreeSpace":"1433041",

"PoolName":"poolA",

"LTOtype":"LTO5"

}

]

}

[http://10.2.7.141/pprime/cgi-bin/QueryKnownTapes?PoolName=PoolDoesntExist&Output=JSON](http://10.2.7.141/pprime/cgi-bin/QueryKnownTapes?PoolName=goofy&Output=JSON)

{

"exit\_code":"400",

"message":"No tape found matching criteria"

}

## Querying free spaces (QueryFreeSpaces)

Service URL:

http://<servername>/pprime/cgi-bin/QueryFreeSpaces

It is used to get minimum and maximum available space on pools.

In Table 3 are shown all the accepted parameters.

|  |  |
| --- | --- |
| **Parameters** | |
| **PoolName** | **Output** |
| Optional | Optional |

Table 3 - QueryFreeSpaces command related parameters

PoolName parameter

If PoolName is supplied, the command will retrieve information only about the specified pool (supplying a non existing pool name will generate an error message). If omitted, the output gives information about all of the found pools.

Ouput parameter

If omitted, the output will be formatted in an HTML table.

Accepted other values are TEXT and JSON.

For each pool, the following information is returned:

* Number of tapes that have some free space (tapes without free space are not counted)
* Total amount of available free space on the pool
* Free space available on the “most full” tape of the pool
* Free space available on the “least full” tape of the pool

Examples:

http://>servername>/ltfsarchiver/cgi-bin/QueryFreeSpaces?Output=JSON

{  
   **"exit\_code"**:"200",  
   **"output"**:[  
      {  
         **"Poolname"**:"poolA",  
         **"NumTapes"**:"3",  
         **"Total"**:"3840694",  
         **"Min"**:"911478",  
         **"Max"**:"1464608"  
      },  
      {  
         **"Poolname"**:"poolB",  
         **"NumTapes"**:"3",  
         **"Total"**:"3871334",  
         **"Min"**:"962796",  
         **"Max"**:"1464608"  
      },      {  
         **"Poolname"**:"poolD",  
         **"NumTapes"**:"1",  
         **"Total"**:"1464608",  
         **"Min"**:"1464608",  
         **"Max"**:"1464608"  
      }  
   ]  
}

http://<servername>/ltfsarchiver/cgi-bin/QueryFreeSpaces?Output=JSON&PoolName=poolB

{"exit\_code":"200","output":[{"Poolname":"poolB","NumTapes":"3","Total":"3871334","Min":"962796","Max":"1464608"}]}

## Archiving operations (WriteToLTO)

Service URL:

*http://<servername>/pprime/cgi-bin/WriteToLTO*

Used to archive a file or a directory (recursive) on an LTO Tape. In Table 4 are shown all the possible combinations of commands and parameters, N/A means that the parameter is not applicable for that command.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Command** | **Parameters** | | | | | |
|  | **FileName** | **PoolName** | **TaskID** | **Output** | **Checksum** | **CecksumFile (\*)** |
| **WriteFile** | Mandatory | Optional | Output | N/a | Optional | Optional |
| **WriteFolder** | Mandatory | Optional | Output | N/a | Optional | Optional |
| **GetStatus** | N/a | N/a | Mandatory | N/a | N/a | N/a |
| **GetResult** | N/a | N/a | Mandatory | Optional | N/a | N/a |
| **Cancel** | N/a | N/a | Mandatory | N/a | N/a | N/a |
| **Resubmit** | N/a | N/a | Mandatory | N/a | N/a | N/a |

Table 4 - WiteLTO commands and related parameters

(\*) Mandatory when Checksum parameter is set to “FILE”, otherwise ignored even if supplied.

All of the answers are given in plain text format, according to the following format:

RC<TAB>RESULT<TAB>Comment

The return code (RC) can be one of the following:

200 -> Request satisfied

400 -> Bad request

500 -> System error

A detailed description of the commands follows.

### WriteFile / WriteFolder commands

WriteFile is used to ask for a single file archiving.

WriteFolder is used to ask for a whole directory archiving (including subdirectories).

FileName parameter

It is the full path to the single file (or directory) to be archived. No pattern or partial path are allowed, the path must be absolute and local to the machine (e.g. /mnt/repository/trythis.mxf).

PoolName parameter

Is the name of the tape pool where LTFSArchiver will try to find a tape with space enough to write the data. If PoolName is not specified, a default pool named “default” is used.

Checksum parameter

This parameter instructs LTFSArchiver about the management (if any) of the archived file checksums.

Several values are supported, with the explained meaning:

* N (default)

LTFSArchiver will not manage checksums at all. The file (or the folder) will be simply copied from disk to tape, without any further check, except I/O errors events.

* MD5

LTFSArchiver will calculate the MD5 checksum of the archived copy of each file involved in archiving process. This value will be shown in the output report (see **GetResult** command further) .

* MD5\_both

LTFSArchiver will calculate MD5 checksum for both source (disk) and target (tape) copy of each file involved in archiving process. The values will be compared and, in case of mismatch, the archived file will be renamed appending a “.corrupted” extension, and both the checksum will be shown in the report file (see **GetResult** command further).

* SHA1

LTFSArchiver will calculate the SHA1checksum of the archived copy of each file involved in archiving process. This value will be shown in the output report (see **GetResult** command further) .

* SHA1\_both

LTFSArchiver will calculate SHA1 checksum for both source (disk) and target (tape) copy of each file involved in archiving process. The values will be compared and, in case of mismatch, the archived file will be renamed appending a “.corrupted” extension, and both the checksum will be shown in the report (see **GetResult** command further).

* FILE

In this case the additional parameter ChecksumFile must be provided with a fullpath to the *CHECKSUMFILE* (format of this file is explained in the next paragraph). LTFSArchiver will calculate MD5 (or SHA1) checksums of the files to be archived and check them against those supplied through the *CHECKSUMFILE*. In case of mismatch, the archived files will be renamed appending a “.corrupted” extension, and both the checksums will be shown in the report (see **GetResult** command further).

**Checksumfile format specification**

The checksumfile is a plain text file. The first line must begin with the special character “#”, immediately followed by the checksum type MD5 or SHA1.The following lines must be in the form:

*checksum\_expected\_value*<blank>\**full\_path\_to\_file*

or

*checksum\_expected\_value*<blank>\**flocat\_to\_already\_archived\_file*

A string “space+asterisk” (hex 20+2A) must separate the two fields.

If the supplied checksum file does not follow these specifications, the archive request will be immediately refused as a bad one (error code 400). If the checksum file content does not correspond with the real file/folder content (different filenames and/or different number of files), the task will be initially taken in charge but it will switch to “bad request” status after a consistency check performed by the LTFSArchiver. In this case the archive operation is not executed.

Each Archive task will produce three reports, in three different file formats:

* Text file (TXT): it is a simple report indicating the final exit code and (if successful) the full file locator (if run to archive a single file) or the base folder file locator (if run to archive a folder).
* XML file: a more complete report, containing ALL the file locators (just one in case of single file archiving) and their checksums if calculated. If checksum match was required and some mismatch has occurred, the “failing” checksum will be reported too.
* JSON file: contains the same content as for the XML, but formatted according to JSON standard.

Only when some checksum operation is required (Checksum parameter specified AND different from “N”), a copy of each report is stored into tape itself, at the same directory level of archived data; their name are *taskid.*txt, *taskid*.xml and *taskid*.json

A general call to WriteToLTO is like:

http://servername/pprime/cgi-bin/WriteToLTO?

Command=<WriteFile|WriteFolder>&FileName=*absolutepathtodata*[&Pooname=*poolname*]

[&Checksum=<MD5|SHA1|MD5\_both|SHA1\_both|File&ChecksumFile=*absolutepathtochecksumfile*>]

Example:

<http://servername/pprime/cgi-bin/WriteToLTO?Command=WriteFile&FileName=/mnt/repository/TryThis.mxf&PoolName=ThisPool>

A successful request returns a message like:

200<TAB>assigned-uuid

where assigned-uuid is a unique id (e.g. 2f2437e2-7565-463c-aa0c-5755b254e24b ) for the operation just taken in charge.

If the source file or directory does not exist, the following answer is given

400<TAB>thisdoesntexist

If the object specified with the parameter FileName does not match with the specific archive command (e.g. when FileName value refers to a directory and the command used is WriteFile), a warning message is returned, but the request is satisfied anyway.

### Cancel command

This command can be used to delete a request but only if it is still waiting for some resource or has been sent to exception due to some error.

<http://servername/pprime/cgi-bin/WriteToLTO?Command=Cancel&TaskID=assigned-uuid>

possible answer are:

200<TAB>assigned-uuid deleted

400<TAB>assigned-uuid doesn't exist

400<TAB>assigned-uuid cannot be deleted

### Resubmit command

This command can be used to re-submit a request that failed.

<http://servername/pprime/cgi-bin/WriteToLTO?Command=Resubmit&TaskID=assigned-uuid>

An operation can be resubmitted only if it has previously failed.

possible answer are:

400<TAB>assigned-uuid doesn't exist

400<TAB>assigned-uuid is not in fallout status

### GetStatus command

This command returns the status of a the operation specified by giving the assigned-uuid with the parameter TaskID.

This command also tries to guess the percentage of the data copied from disk to tape when the task is still writing data. During the interval between the start of the archive command and the actual “first byte write”, the size of the archived data will be unavailable.

Example:

<http://servername/pprime/cgi-bin/WriteToLTO?Command=GetStatus&TaskID=2f2437e2-7565-463c-aa0c-5755b254e24b>

The answer during the writing can be like:

200<TAB>running p=80.124

While when the task has completed, the answer is:

200<TAB>completed

See appendix A for a more comprehensive list of possible status and status description.

**Please note that GetStatus returns “completed” even if the task actually failed, to get the actual final status of the task, use GetResult** (see following chapter)

### GetResult command

This command provides full information about the a specific archiving task result.

Output parameter

Possible values are “TEXT” (default if not specified), “JSON” or “XML”

Using a value rather than the other one not only affects the output formatting, but also affects the type of the information given.

Examples are available in **Appendix B**.

## Restoring operations (RestoreFromLTO)

Service URL:

[http://servername/pprime/cgi-bin/RestoreFromLTO](http://10.2.7.141/pprime/cgi-bin/QueryKnownTapes)

Used to ask for a restore of a file or a directory from an LTO tape and to follow the ongoing restore operations and get their result. **The system leaves to the clients the responsibility to remember which file is on which tape** hence for restoring is necessary to specify the full Flocat URN of the file or folder. Of course is possible to list the content of a specific tape to discover all the files stored in (see MakeAvailable command).

In Table 5 are shown all the possible combinations of commands and parameters, N/A means that the parameter is not applicable for that command.

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Parameters** | | |
|  | **FileName** | **DestPath** | **TaskID** |
| **RestoreFile** | Mandatory | Mandatory | Output |
| **GetStatus** | N/a | N/a | Mandatory |
| **GetResult** | N/a | N/a | Mandatory |
| **Cancel** | N/a | N/a | Mandatory |
| **Resubmit** | N/a | N/a | Mandatory |

Table 5 - RestoreFromLTO commands and related parameters

### RestoreFile command

This command is used to ask for a recovery of a file or a directory previously archived.

FileName parameter

It is the Flocat URN that identifies the file or the directory to restore. It is exactly what was given back as a result of a WriteToLTO operation.

DestPath parameter

It is the name of the destination file/directory of the restored object.

To avoid overwriting and/or creation of unwanted directories, the following checks are applied:

* the DestPath object MUST NOT exist
* the upper-level of DestPath MUST exist

e.g. if DesPath is /var/mydir/mydata, the directory “var/mydir” must exist and it must not contain a file or directory named “mydata”.

If one of the check fails, an error message is returned:

400<TAB>Destination path DestPath exists

400<TAB>Destination upper level path nameofupperleveldir doesn't exist

If all checks has passed, the message will be:

200<TAB>assigned-uuid

Meaning that the operation has been queued for later execution.

### GetStatus, Cancel and Resubmit commands

**GetStatus**, **Cancel,** and **Resubmit** commands have the same syntax, usage and meaning as for the WriteToLTO service.

### GetResult command

The **GetResult** command returns the result of the operation, i.e. if the restore operation has been completed successfully or not.

In case of error, a specific description of the problem occurred can be retrieved using GetErrorDescr as follow:

[http://servername/pprime/cgi-bin/GetErrorDescr?TaskID=assigned-uuid](http://10.2.7.141/pprime/cgi-bin/GetErrorDescr?TaskID=0de63e38-18d4-4cd1-b9a3-ab41f6517ea6)

## Giving access through LTFS direct access (MakeAvailable)

Service URL:

[http://servername/pprime/cgi-bin/MakeAvailable](http://10.2.7.141/pprime/cgi-bin/QueryKnownTapes)

It is used to open/close a **read only** access to the whole content of a LTO. The tape will be locally mounted (using LTFS) on the controlling host and possibly made available remotely through a share.

The client application will be allowed to read data directly from the tape file system, thus allowing services to run processes on archived files without need to copy them locally. Examples of such use include partial extraction of big files (e.g. multimedia), services for regular data integrity check, storage or format migrations.

In Table 6 are shown all the possible combinations of commands and parameters, N/A means that the parameter is not applicable for that command.

|  |  |  |
| --- | --- | --- |
| **Command** | **Parameters** | |
|  | **TapeID** | **TaskID** |
| **Mount** | Mandatory | Output |
| **Unmount** | Mandatory | Output |
| **GetStatus** | N/A | Mandatory |
| **GetResult** | N/A | Mandatory |
| **Cancel** | N/A | Mandatory |
| **Resubmit** | N/A | Mandatory |

Table 6 - MakeAvailable commands and related parameters

All of the answers from the service are plain text, according to the following format:

RC<TAB>RESULT<TAB>Comment

Where RC will be one of the following:

200 -> Request satisfied

400 -> Bad request

500 -> System error

and RESULT value may vary according to the command issued.

### Mount command

This command is used to get access (in read only mode) to the whole content of a single tape.

TapeID

Is the only needed (and mandatory) parameter to supply and is the label of the tape to get read only access to.

Example:

[http://servername/pprime/cgi-bin/MakeAvail?Command=Mount&LTOLabel=ThisTape](http://10.2.7.141/pprime/cgi-bin/MakeAvail?Command=Mount%3cOLabel=EX000001)

If the TapeID supplied exists in one of the available pool, the answer is:

200<TAB>assigned-uuid

where assigned-uuid is the unique identifier assigned to the request.

If the supplied TapeID is not found in the system, the answer is:

400<TAB>LTO with label: ThisDoesntExists does not exists

### GetStatus command

The GetStatus command is used to know if the request “assigned-uuid” is still waiting to be satisfied or it has been executed. TaskID must be supplied, and must match the uuid given as answer returned when issuing the Mount command.

[http://servername/pprime/cgi-bin/MakeAvail?Command=GetStatus&TaskID=assigned-uuid](http://10.2.7.141/pprime/cgi-bin/MakeAvail?Command=GetStatus&TaskID=cda49c21-1805-43a0-84dc-482e367cb971)

Possible answers are:

400<TAB>uuid doesn't exist

200<TAB>wait

200<TAB>running

200<TAB>completed

400 is self-explaining: the GetStatus has been issued on a non-existing/allowed operation.

The “wait” status means that the operation is waiting for its execution. The cause of this is in general that no free resources are available (tape in use by another operation, all of the tape drives are already in use) or, in case a manual operation required, that the physical tape loading has not been done and confirmed by the operator.

The “running” status means that the tape is in loading/positioning status. Normally the operation is going to be completed in few seconds.

The “completed” status means that the agent has completed the load / position / check operation. This status DOES NOT ASSURE that the tape has been made available successfully (It could have failed to an I/O error or FS error). To know the real result of the command it is necessary to issue the GetResult command (see further).

### GetResult command

This command is used to know if a MakeAvailble request had an happy ending or not.

[http://servername/pprime/cgi-bin/MakeAvail?Command=GetResult&TaskID=assigned-uuid](http://10.2.7.141/pprime/cgi-bin/MakeAvail?Command=GetResult&TaskID=assigned-uuid)

Possible answers are:

200<TAB>Success<TAB>local-path-to-LTFS

400<TAB>uuid doesn't exist

400<TAB>Not completed

500<TAB>Failure

200 Success means that the command was completed without errors, and the LTO content is accessible through the path “local-path-to-LTFS”

400 messages are self-explaining: the GetStatus has been issued on a non-existent or uncompleted operation

500 Means that something went wrong, e.g. FS errors, wrong tape loaded, etc.

To know exactly which error occurred, the generic GetErrorDescr service has to be called:

[http://servername/pprime/cgi-bin/GetErrorDescr?TaskID=assigned-uuid](http://10.2.7.141/pprime/cgi-bin/GetErrorDescr?TaskID=0de63e38-18d4-4cd1-b9a3-ab41f6517ea6)

The answer will be one of the following:

400<TAB>assigned-uuid is not in fallout status

200<TAB>NumericErrorcode<TAB>Errordescription

### Cancel command

This command can be used to delete a Makeavailable request but only if it is still waiting for some resource assignment.

[http://servername/pprime/cgi-bin/MakeAvail?Command=Cancel&TaskID=assigned-uuid](http://10.2.7.141/pprime/cgi-bin/MakeAvail?Command=Cancel&TaskID=assigned-uuid)

possible answer are:

200<TAB>assigned-uuid deleted

400<TAB>assigned-uuid doesn't exist

400<TAB>assigned-uuid is not in wait status

### Unmount command

This command is used to switch off the read only access previously given to a tape through the Mount command.

LTOLabel is the only needed (and mandatory) parameter to supply and is the label of the made available tape.

<http://servername/pprime/cgi-bin/MakeAvail?Command=Unmount&TapeID=>[ThisTape](http://10.2.7.141/pprime/cgi-bin/MakeAvail?Command=Mount%3cOLabel=EX000001)

If the TapeID supplied refers to an LTO tape currently available as a local file system, the answer will be:

200<TAB>assigned-uuid

If the supplied TapeID is not currently in a makeavailable status, the answer is:

400<TAB>ThisTape is not a made available tape at time

The commands: **GetStatus**, **GetResult**, **Cancel** and **Resubmit** have the same meaning and behavior as when used in conjunction with a uuid assigned to a Mount operations.

**Warning Note:**

If a client is still accessing the made available LTO content, the Unmount request will fail and the request automatically requeued. Be sure to release every access to it before unmounting the tape.

## Appendix A - Status and substatus description

All the described services (except Withdraw in TapeManager) support the GetStatus command whose output may slightly vary according to the service used. In any case the common following rules are always applied:

* All the fields are separated by a <TAB> (Hex 09) character
* The first field is a numeric code (see table 7)
* The second field is a single world describing the main status (table 8)
* The third field (if present) supplies the sub-status or more specific information (table 8)

Numeric code field

|  |  |
| --- | --- |
| Value | Meaning |
| 200 | Valid query, message follows |
| 400 | Query referring an un-existing uuid |
| 500 | Failure |

Table – GetStatus Return code meaning

Status field

|  |  |  |
| --- | --- | --- |
| Status | Meaning | Possible Descriptions |
| Wait | LTFSArchiver isn’t still able to assign all of the requested resources (i.e. all of the tape drives are full) | Waiting to be dispatched |
| Dispatched, waiting for tape device |
| Dispatched, waiting for tape transferring from/to device |
| Starting | LTFSArchiver is moving/preparing the requested tape | Tape being loaded o positioning |
| Tape loaded and ready |
| Running | LTFSArchiver is executing the requested action | *Varying, according to involved interface* |
| Completed | LTFSArchiver has terminated the requested action | n/a |

Table - Possible Statuses and their meaning

## Appendix B – Archiving file commands and reports samples

### Submitting an archive task

The syntax to be used for scheduling an archive operation is the following:

http://servername/pprime/cgi-bin/WriteToLTO?

Command=<WriteFile|WriteFolder>&FileName=*absolutepathtodata*[&Pooname=*poolname*]

[&Checksum=<MD5|SHA1|MD5\_both|SHA1\_both|File<&ChecksumFile=*absolutepathtochecksumfile*>>]

If a valid combination of commands and parameters is issued, the expected answer (assuming that the object to be archived exists) will be:

200 *taskid*

### Monitoring an archive task

The value of taskid has to be used to monitor the archive task, using the GetStatus command.

http://10.58.78.112/ltfsarchiver/cgi-bin/WriteToLTO?Command=GetStatus&TaskID=*taskid*

Several values can be returned:

200 starting Tape being loaded o positioning

200 starting Tape loaded and ready

200 running p=90.175

200 Completed

### Getting the final result of the task

After GetStatus command has returned “Completed” it is possible to get the final result:

http://10.58.78.112/ltfsarchiver/cgi-bin/WriteToLTO?Command=GetResult&TaskID=*taskid*

If the archive task has been successfully completed the command will return:

200 success *flocat*

Where *flocat* is the file locator of the archived file (or the base file locator of the archived folder).

The file locator syntax is the following:

lto-ltfs:*ltolabel*:*taskid*/*archiveditem*

|  |  |
| --- | --- |
| lto-ltfs | Indicates the type of physical support and filesystem type |
| *ltolabel* | The label of the LTO tape used to archive item(s) |
| *taskid* | The task assigned when WriteFile/WriteFolder command has been issued |
| *archiveditem* | File or directory basename that has been archived |

If some error occurred, a 500 exit code will be reported:

500 success failure *(error description follows)*

A more comprehensive report will be available in XML or JSON formats

To get them, add &Output=<XML|JSON> parameter to the GetStatus command

Different reports according to Checksum parameter use.

The “basic” TEXT report only shows a small set of information:

* Success/failure
* Base flocat (if successful)
* Error description (if failed)

XML and JSON reports are more verbose and include checksum match information in case this feature has been required.

**Sample A: Archive of a single file, no checksum operation required**

Request syntax

Command=WriteFile&FileName=/data/LTFStest/sample.file&PoolName=demopool

Assigned taskid:

200 a77e29d3-1c4a-45c9-93db-2144de54fab6

GetResult Command (XML):

Command=GetResult&TaskID=a77e29d3-1c4a-45c9-93db-2144de54fab6&Output=XML

GetResult Output (XML):

<LTFSArchiver jobid="a77e29d3-1c4a-45c9-93db-2144de54fab6" command="WriteFile" exitcode="200">

<message>success</message>

<FLocat xlink:href="lto-ltfs:LTFSA001:a77e29d3-1c4a-45c9-93db-2144de54fab6/sample.file"/>

</LTFSArchiver>

GetResult Output (JSON):

Command=GetResult&TaskID=a77e29d3-1c4a-45c9-93db-2144de54fab6&Output=JSON

GetResult Output (JSON):

{

"jobid":"a77e29d3-1c4a-45c9-93db-2144de54fab6",

"service":"WriteToLTO",

"command":"WriteFile",

"exitcode":"200",

"message":"success",

"FLocats":[

{

"Flocat":"lto-ltfs:LTFSA001:a77e29d3-1c4a-45c9-93db-2144de54fab6/sample.file"

}

]

}

**Sample B: Archive of a single file, MD5 checksum calculation required**

Request syntax

Command=WriteFile&FileName=/data/LTFStest/sample.file&PoolName=demopool&Checksum=MD5

Assigned taskid:

200 9b171424-1257-43ea-912d-1b3a7261327f

GetResult Command (XML):

Command=GetResult&TaskID=9b171424-1257-43ea-912d-1b3a7261327f&Output=XML

GetResult Output (XML):

<LTFSArchiver jobid="9b171424-1257-43ea-912d-1b3a7261327f" command="WriteFile" exitcode="200">

<message>success</message>

<FLocat xlink:href="lto-ltfs:LTFSA001:9b171424-1257-43ea-912d-1b3a7261327f/sample.file">

<checksum type="MD5" value="ff7e1a2e531c857d46e68f97de99ce48" />

</FLocat>

</LTFSArchiver>

GetResult Output (JSON):

Command=GetResult&TaskID=9b171424-1257-43ea-912d-1b3a7261327f&Output=JSON

GetResult Output (JSON):

{

"jobid":"9b171424-1257-43ea-912d-1b3a7261327f",

"service":"WriteToLTO",

"command":"WriteFile",

"exitcode":"200",

"message":"success",

"FLocats":[

{

"Flocat":"lto-ltfs:LTFSA001:9b171424-1257-43ea-912d-1b3a7261327f/sample.file",

"checksum":{

"type":"MD5",

"value":"ff7e1a2e531c857d46e68f97de99ce48"

}

}

]

}

**Sample C: Archive of a single file, SHA1 checksum calculation and verification against source required**

Request syntax

Command=WriteFile&FileName=/data/LTFStest/sample.file&PoolName=demopool&Checksum=SHA1\_both

Assigned taskid:

200 ee0efe2e-4ca3-4344-aa60-df08531b7932

GetResult Command (XML):

Command=GetResult&TaskID=ee0efe2e-4ca3-4344-aa60-df08531b7932&Output=XML

GetResult Output (XML):

<LTFSArchiver jobid="ee0efe2e-4ca3-4344-aa60-df08531b7932" command="WriteFile" exitcode="200">

<message>success</message>

<FLocat xlink:href="lto-ltfs:LTFSA001:ee0efe2e-4ca3-4344-aa60-df08531b7932/sample.file">

<checksum type="SHA1" expectedvalue="f4b3bd12d724d552c10b4e18a82097952052e8a2" value="f4b3bd12d724d552c10b4e18a82097952052e8a2" match="true"/>

</FLocat>

</LTFSArchiver>

GetResult Output (JSON):

Command=GetResult&TaskID=ee0efe2e-4ca3-4344-aa60-df08531b7932&Output=JSON

GetResult Output (JSON):

{

"jobid":"ee0efe2e-4ca3-4344-aa60-df08531b7932",

"service":"WriteToLTO",

"command":"WriteFile",

"exitcode":"200",

"message":"success",

"FLocats":[

{

"Flocat":"lto-ltfs:LTFSA001:ee0efe2e-4ca3-4344-aa60-df08531b7932/sample.file",

"checksum":{

"type":"SHA1",

"expectedvalue":"f4b3bd12d724d552c10b4e18a82097952052e8a2",

"value":"f4b3bd12d724d552c10b4e18a82097952052e8a2",

"match":"true"

}

}

]

}

Where expectedvalue is the checksum calculated on the source file and value is the checksum of the SAVED one

**Sample D: Archive of a single file, MD5 checksum calculation and verification against supplied MD5 value**

This example assumes that the user has already a text file containing the pre-calculated checksum value. This file would be like:

#MD5

ff7e1a2e531c857d46e68f97de99ce48 \*/data/LTFStest/sample.file

Request syntax

Command=WriteFile&FileName=/data/LTFStest/sample.file&PoolName=demopool&Checksum=FILE&ChecksumFile=/data/LTFStest/checksumfile1.txt

Assigned taskid:

200 a965f20c-d8e3-4a47-897d-618f790a637a

GetResult Command (XML):

Command=GetResult&TaskID=a965f20c-d8e3-4a47-897d-618f790a637a&Output=XML

GetResult Output (XML):

<LTFSArchiver jobid="a965f20c-d8e3-4a47-897d-618f790a637a" command="WriteFile" exitcode="200">

<message>success</message>

<FLocat xlink:href="lto-ltfs:LTFSA001:a965f20c-d8e3-4a47-897d-618f790a637a/sample.file">

<checksum type="MD5" expectedvalue="ff7e1a2e531c857d46e68f97de99ce48" value="ff7e1a2e531c857d46e68f97de99ce48" match="true"/>

</FLocat>

</LTFSArchiver>

GetResult Command (JSON):

Command=GetResult&TaskID= a965f20c-d8e3-4a47-897d-618f790a637a&Output=JSON

GetResult Output (JSON):

{

"jobid":"a965f20c-d8e3-4a47-897d-618f790a637a",

"service":"WriteToLTO",

"command":"WriteFile",

"exitcode":"200",

"message":"success",

"FLocats":[

{

"Flocat":"lto-ltfs:LTFSA001:a965f20c-d8e3-4a47-897d-618f790a637a/sample.file",

"checksum":{

"type":"MD5",

"expectedvalue":"ff7e1a2e531c857d46e68f97de99ce48",

"value":"ff7e1a2e531c857d46e68f97de99ce48",

"match":"true"

}

}

]

}

Where expectedvalue is the checksum of the file just written on the tape and re-read on purpose and value is the value supplied through the pre-calculated checksum file.

**Sample E: Archive of a whole directory, no checksum operation required**

Request syntax

Command=WriteFolder&FileName=/data/LTFStest/sampledir&PoolName=demopool

Assigned taskid:

200 c7da6356-a50f-4acb-a5fc-e1782fcf0679

GetResult Command (XML):

Command=GetResult&TaskID= c7da6356-a50f-4acb-a5fc-e1782fcf0679&Output=XML

GetResult Output (XML):

<LTFSArchiver jobid="c7da6356-a50f-4acb-a5fc-e1782fcf0679" command="WriteFolder" exitcode="200">

<message>success</message>

<FLocat xlink:href="lto-ltfs:LTFSA001:c7da6356-a50f-4acb-a5fc-e1782fcf0679/sampledir/dir\_to\_archive/sample.txt"/>

<FLocat xlink:href="lto-ltfs:LTFSA001:c7da6356-a50f-4acb-a5fc-e1782fcf0679/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt"/>

<FLocat xlink:href="lto-ltfs:LTFSA001:c7da6356-a50f-4acb-a5fc-e1782fcf0679/sampledir/dir\_to\_archive/sample.bin"/>

<FLocat xlink:href="lto-ltfs:LTFSA001:c7da6356-a50f-4acb-a5fc-e1782fcf0679/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin"/>

</LTFSArchiver>

GetResult Command (JSON):

Command=GetResult&TaskID=c7da6356-a50f-4acb-a5fc-e1782fcf0679&Output=JSON

GetResult Output (JSON):

{  
   **"jobid"**:"c7da6356-a50f-4acb-a5fc-e1782fcf0679",  
   **"service"**:"WriteToLTO",  
   **"command"**:"WriteFolder",  
   **"exitcode"**:"200",  
   **"message"**:"success",  
   **"FLocats"**:[  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:c7da6356-a50f-4acb-a5fc-e1782fcf0679/sampledir/dir\_to\_archive/sample.txt"  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:c7da6356-a50f-4acb-a5fc-e1782fcf0679/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt"  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:c7da6356-a50f-4acb-a5fc-e1782fcf0679/sampledir/dir\_to\_archive/sample.bin"  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:c7da6356-a50f-4acb-a5fc-e1782fcf0679/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin"  
      }  
   ]  
}

**Sample F: Archive of a whole directory, only SHA1 checksum calculation required**

Request syntax

Command=WriteFolder&FileName=/data/LTFStest/sampledir&PoolName=demopool&Checksum=SHA1

Assigned taskid:

200 dddfd308-9a74-43f5-8313-da3b69a945df

GetResult Command (XML):

Command=GetResult&TaskID=dddfd308-9a74-43f5-8313-da3b69a945df&Output=XML

GetResult Output (XML):

<LTFSArchiver jobid="dddfd308-9a74-43f5-8313-da3b69a945df" command="WriteFolder" exitcode="200">

<message>success</message>

<FLocat xlink:href="lto-ltfs:LTFSA001:dddfd308-9a74-43f5-8313-da3b69a945df/sampledir/dir\_to\_archive/sample.txt">

<checksum type="SHA1" value="32127a6cd22cd0dfceb996efdfc843e285f0f825" />

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:dddfd308-9a74-43f5-8313-da3b69a945df/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt">

<checksum type="SHA1" value="32127a6cd22cd0dfceb996efdfc843e285f0f825" />

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:dddfd308-9a74-43f5-8313-da3b69a945df/sampledir/dir\_to\_archive/sample.bin">

<checksum type="SHA1" value="f4b3bd12d724d552c10b4e18a82097952052e8a2" />

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:dddfd308-9a74-43f5-8313-da3b69a945df/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin">

<checksum type="SHA1" value="f4b3bd12d724d552c10b4e18a82097952052e8a2" />

</FLocat>

</LTFSArchiver>

GetResult Command (JSON):

Command=GetResult&TaskID=dddfd308-9a74-43f5-8313-da3b69a945df&Output=JSON

GetResult Output (JSON):

{  
   **"jobid"**:"dddfd308-9a74-43f5-8313-da3b69a945df",  
   **"service"**:"WriteToLTO",  
   **"command"**:"WriteFolder",  
   **"exitcode"**:"200",  
   **"message"**:"success",  
   **"FLocats"**:[  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:dddfd308-9a74-43f5-8313-da3b69a945df/sampledir/dir\_to\_archive/sample.txt",  
         **"checksum"**:{  
            **"type"**:"SHA1",  
            **"value"**:"32127a6cd22cd0dfceb996efdfc843e285f0f825"  
         }  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:dddfd308-9a74-43f5-8313-da3b69a945df/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt",  
         **"checksum"**:{  
            **"type"**:"SHA1",  
            **"value"**:"32127a6cd22cd0dfceb996efdfc843e285f0f825"  
         }  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:dddfd308-9a74-43f5-8313-da3b69a945df/sampledir/dir\_to\_archive/sample.bin",  
         **"checksum"**:{  
            **"type"**:"SHA1",  
            **"value"**:"f4b3bd12d724d552c10b4e18a82097952052e8a2"  
         }  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:dddfd308-9a74-43f5-8313-da3b69a945df/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin",  
         **"checksum"**:{  
            **"type"**:"SHA1",  
            **"value"**:"f4b3bd12d724d552c10b4e18a82097952052e8a2"  
         }  
      }  
   ]  
}

Where value is the SHA1 checksum of the SAVED files that are re-read from tape on purpose.

**Sample G: Archive of a whole directory, SHA1 checksum and check against source required**

Request syntax

Command=WriteFolder&FileName=/data/LTFStest/sampledir&PoolName=demopool&Checksum=SHA1

Assigned taskid:

200 58debbee-d51e-493f-bd8e-6dab689afe05

GetResult Command (XML):

Command=GetResult&TaskID=58debbee-d51e-493f-bd8e-6dab689afe05&Output=XML

GetResult Output (XML):

<LTFSArchiver jobid="58debbee-d51e-493f-bd8e-6dab689afe05" command="WriteFolder" exitcode="200">

<message>success</message>

<FLocat xlink:href="lto-ltfs:LTFSA001:58debbee-d51e-493f-bd8e-6dab689afe05/sampledir/dir\_to\_archive/sample.txt">

<checksum type="MD5" expectedvalue="385f6d7c0ef34f5a8ad40c34110afedb" value="385f6d7c0ef34f5a8ad40c34110afedb" match="true"/>

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:58debbee-d51e-493f-bd8e-6dab689afe05/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt">

<checksum type="MD5" expectedvalue="385f6d7c0ef34f5a8ad40c34110afedb" value="385f6d7c0ef34f5a8ad40c34110afedb" match="true"/>

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:58debbee-d51e-493f-bd8e-6dab689afe05/sampledir/dir\_to\_archive/sample.bin">

<checksum type="MD5" expectedvalue="ff7e1a2e531c857d46e68f97de99ce48" value="ff7e1a2e531c857d46e68f97de99ce48" match="true"/>

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:58debbee-d51e-493f-bd8e-6dab689afe05/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin">

<checksum type="MD5" expectedvalue="ff7e1a2e531c857d46e68f97de99ce48" value="ff7e1a2e531c857d46e68f97de99ce48" match="true"/>

</FLocat>

</LTFSArchiver>

GetResult Command (JSON):

Command=GetResult&TaskID=58debbee-d51e-493f-bd8e-6dab689afe05&Output=JSON

GetResult Output (JSON):

{  
   **"jobid"**:"58debbee-d51e-493f-bd8e-6dab689afe05",  
   **"service"**:"WriteToLTO",  
   **"command"**:"WriteFolder",  
   **"exitcode"**:"200",  
   **"message"**:"success",  
   **"FLocats"**:[  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:58debbee-d51e-493f-bd8e-6dab689afe05/sampledir/dir\_to\_archive/sample.txt",  
         **"checksum"**:{  
            **"type"**:"MD5",  
            **"expectedvalue"**:"385f6d7c0ef34f5a8ad40c34110afedb",  
            **"value"**:"385f6d7c0ef34f5a8ad40c34110afedb",  
            **"match"**:"true"  
         }  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:58debbee-d51e-493f-bd8e-6dab689afe05/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt",  
         **"checksum"**:{  
            **"type"**:"MD5",  
            **"expectedvalue"**:"385f6d7c0ef34f5a8ad40c34110afedb",  
            **"value"**:"385f6d7c0ef34f5a8ad40c34110afedb",  
            **"match"**:"true"  
         }  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:58debbee-d51e-493f-bd8e-6dab689afe05/sampledir/dir\_to\_archive/sample.bin",  
         **"checksum"**:{  
            **"type"**:"MD5",  
            **"expectedvalue"**:"ff7e1a2e531c857d46e68f97de99ce48",  
            **"value"**:"ff7e1a2e531c857d46e68f97de99ce48",  
            **"match"**:"true"  
         }  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:58debbee-d51e-493f-bd8e-6dab689afe05/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin",  
         **"checksum"**:{  
            **"type"**:"MD5",  
            **"expectedvalue"**:"ff7e1a2e531c857d46e68f97de99ce48",  
            **"value"**:"ff7e1a2e531c857d46e68f97de99ce48",  
            **"match"**:"true"  
         }  
      }  
   ]  
}

**Sample H: Archive of whole directory, SHA1 checksum calculation and verification against supplied SHA1 values**

This sample case assume that user has already generated a file containing the pre-calculated checksum value. The content of the file follows:

#SHA1

f4b3bd12d724d552c10b4e18a82097952052e8a2 \*/data/LTFStest/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin

32127a6cd22cd0dfceb996efdfc843e285f0f825 \*/data/LTFStest/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt

f4b3bd12d724d552c10b4e18a82097952052e8a2 \*/data/LTFStest/sampledir/dir\_to\_archive/sample.bin

32127a6cd22cd0dfceb996efdfc843e285f0f825 \*/data/LTFStest/sampledir/dir\_to\_archive/sample.txt

Request syntax

Command=WriteFolder&FileName=/data/LTFStest/sampledir&PoolName=demopool&Checksum=FILE&ChecksumFile=/data/LTFStest/checksumfile2.txt

Assigned taskid:

200 acfb3935-0302-4b70-b2a0-27205b6dd949

GetResult Command (XML):

Command=GetResult&TaskID= acfb3935-0302-4b70-b2a0-27205b6dd949&Output=XML

GetResult Output (XML):

<LTFSArchiver jobid="acfb3935-0302-4b70-b2a0-27205b6dd949" command="WriteFolder" exitcode="200">

<message>success</message>

<FLocat xlink:href="lto-ltfs:LTFSA001:acfb3935-0302-4b70-b2a0-27205b6dd949/sampledir/dir\_to\_archive/sample.txt">

<checksum type="SHA1" expectedvalue="32127a6cd22cd0dfceb996efdfc843e285f0f825" value="32127a6cd22cd0dfceb996efdfc843e285f0f825" match="true"/>

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:acfb3935-0302-4b70-b2a0-27205b6dd949/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt">

<checksum type="SHA1" expectedvalue="32127a6cd22cd0dfceb996efdfc843e285f0f825" value="32127a6cd22cd0dfceb996efdfc843e285f0f825" match="true"/>

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:acfb3935-0302-4b70-b2a0-27205b6dd949/sampledir/dir\_to\_archive/sample.bin">

<checksum type="SHA1" expectedvalue="f4b3bd12d724d552c10b4e18a82097952052e8a2" value="f4b3bd12d724d552c10b4e18a82097952052e8a2" match="true"/>

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:acfb3935-0302-4b70-b2a0-27205b6dd949/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin">

<checksum type="SHA1" expectedvalue="f4b3bd12d724d552c10b4e18a82097952052e8a2" value="f4b3bd12d724d552c10b4e18a82097952052e8a2" match="true"/>

</FLocat>

</LTFSArchiver>

GetResult Command (JSON):

Command=GetResult&TaskID= acfb3935-0302-4b70-b2a0-27205b6dd949&Output=JSON

{  
   **"jobid"**:"acfb3935-0302-4b70-b2a0-27205b6dd949",  
   **"service"**:"WriteToLTO",  
   **"command"**:"WriteFolder",  
   **"exitcode"**:"200",  
   **"message"**:"success",  
   **"FLocats"**:[  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:acfb3935-0302-4b70-b2a0-27205b6dd949/sampledir/dir\_to\_archive/sample.txt",  
         **"checksum"**:{  
            **"type"**:"SHA1",  
            **"expectedvalue"**:"32127a6cd22cd0dfceb996efdfc843e285f0f825",  
            **"value"**:"32127a6cd22cd0dfceb996efdfc843e285f0f825",  
            **"match"**:"true"  
         }  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:acfb3935-0302-4b70-b2a0-27205b6dd949/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt",  
         **"checksum"**:{  
            **"type"**:"SHA1",  
            **"expectedvalue"**:"32127a6cd22cd0dfceb996efdfc843e285f0f825",  
            **"value"**:"32127a6cd22cd0dfceb996efdfc843e285f0f825",  
            **"match"**:"true"  
         }  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:acfb3935-0302-4b70-b2a0-27205b6dd949/sampledir/dir\_to\_archive/sample.bin",  
         **"checksum"**:{  
            **"type"**:"SHA1",  
            **"expectedvalue"**:"f4b3bd12d724d552c10b4e18a82097952052e8a2",  
            **"value"**:"f4b3bd12d724d552c10b4e18a82097952052e8a2",  
            **"match"**:"true"  
         }  
      },  
      {  
         **"Flocat"**:"lto-ltfs:LTFSA001:acfb3935-0302-4b70-b2a0-27205b6dd949/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin",  
         **"checksum"**:{  
            **"type"**:"SHA1",  
            **"expectedvalue"**:"f4b3bd12d724d552c10b4e18a82097952052e8a2",  
            **"value"**:"f4b3bd12d724d552c10b4e18a82097952052e8a2",  
            **"match"**:"true"  
         }  
      }  
   ]  
}

Where expectedvalue is the checksum of files calculated after writing on tape and value is the value supplied through the pre-calculated checksum file.

**Sample I: Archive of a whole directory, checksumfile supplied but with missing entry**

Let’s suppose the following directory tree to be archived:

dir\_to\_archive

├── sample.bin

├── sample.txt

└── sub\_dir with\_space

├── sample.bin

└── sample.txt

and the following checksum file is supplied:

#SHA1

f4b3bd12d724d552c10b4e18a82097952052e8a2 \*/data/LTFStest/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin

f4b3bd12d724d552c10b4e18a82097952052e8a2 \*/data/LTFStest/sampledir/dir\_to\_archive/sample.bin

32127a6cd22cd0dfceb996efdfc843e285f0f825 \*/data/LTFStest/sampledir/dir\_to\_archive/sample.txt

Request syntax

Command=WriteFolder&FileName=/data/LTFStest/sampledir&PoolName=demopool&Checksum=FILE&ChecksumFile=/data/LTFStest/badchecksumfile.txt

Assigned taskid:

200 eed50a0b-2e01-4c7a-bb93-45da8bbe0e09

GetResult Command (TEXT):

400 bad\_request Mismatch found between checksum file and file system (number of file(s) differs)

GetResult Command (JSON):

<LTFSArchiver jobid="eed50a0b-2e01-4c7a-bb93-45da8bbe0e09" command="WriteFolder" exitcode="400">

<message> Mismatch found between checksum file and file system (number of file(s) differs)</message>

</LTFSArchiver>

GetResult Command (JSON):

{  
   **"jobid"**:"eed50a0b-2e01-4c7a-bb93-45da8bbe0e09",  
   **"service"**:"WriteToLTO",  
   **"command"**:"WriteFolder",  
   **"exitcode"**:"400",  
   **"message"**:" Mismatch found between checksum file and file system (number of file(s) differs)"  
}

This error is due to the fact that the supplied checksum file contains only three entries (with checksum ad filename) instead of the four expected. This check is performed before trying to archive any data in order to avoid wasting space on tape(s).

The task CANNOT be resubmitted. It has to be deleted (using the Cancel command) and a brand new request must be performed after having fixed the checksum file.

**Sample J: Archive of a whole directory, checksumfile supplied but with some wrong value**

Let’s suppose the following directory tree to be archived:

dir\_to\_archive

├── sample.bin

├── sample.txt

└── sub\_dir with\_space

├── sample.bin

└── sample.txt

and the following checksum file is supplied:

#SHA1

f4b3bd12d724d552c10b4e18a82097952052e8a2 \*/data/LTFStest/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin

32127a6cd22cd0dfceb996efdfc843e285f0f825 \*/data/LTFStest/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt

f4b3bd12d724d552c10b4e18a82097952052e8af \*/data/LTFStest/sampledir/dir\_to\_archive/sample.bin

32127a6cd22cd0dfceb996efdfc843e285f0f825 \*/data/LTFStest/sampledir/dir\_to\_archive/sample.txt

Request syntax

Command=WriteFolder&FileName=/data/LTFStest/sampledir&PoolName=demopool&Checksum=FILE&ChecksumFile=/data/LTFStest/badchecksumfile.txt

Assigned taskid:

200 5778a231-dc8d-4685-9537-0f3d184a9099

GetResult Command (TEXT):

500 failure lto-ltfs:LTFSA001:5778a231-dc8d-4685-9537-0f3d184a9099/sampledir Some file didn not pass checksum verification

GetResult Command (XML):

<LTFSArchiver jobid="5778a231-dc8d-4685-9537-0f3d184a9099" command="WriteFolder" exitcode="500">

<message> Some file didn not pass checksum verification</message>

<FLocat xlink:href="lto-ltfs:LTFSA001:5778a231-dc8d-4685-9537-0f3d184a9099/sampledir/dir\_to\_archive/sample.txt">

<checksum type="SHA1" expectedvalue="32127a6cd22cd0dfceb996efdfc843e285f0f825" value="32127a6cd22cd0dfceb996efdfc843e285f0f825" match="true"/>

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:5778a231-dc8d-4685-9537-0f3d184a9099/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt">

<checksum type="SHA1" expectedvalue="32127a6cd22cd0dfceb996efdfc843e285f0f825" value="32127a6cd22cd0dfceb996efdfc843e285f0f825" match="true"/>

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:5778a231-dc8d-4685-9537-0f3d184a9099/sampledir/dir\_to\_archive/sample.bin.corrupted">

<checksum type="SHA1" expectedvalue="f4b3bd12d724d552c10b4e18a82097952052e8af" value="f4b3bd12d724d552c10b4e18a82097952052e8a2" match="false"/>

</FLocat>

<FLocat xlink:href="lto-ltfs:LTFSA001:5778a231-dc8d-4685-9537-0f3d184a9099/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin">

<checksum type="SHA1" expectedvalue="f4b3bd12d724d552c10b4e18a82097952052e8a2" value="f4b3bd12d724d552c10b4e18a82097952052e8a2" match="true"/>

</FLocat>

</LTFSArchiver>

GetResult Command (JSON):

{

"jobid":"5778a231-dc8d-4685-9537-0f3d184a9099",

"service":"WriteToLTO",

"command":"WriteFolder",

"exitcode":"500",

"message":" Some file didn not pass checksum verification",

"FLocats":[

{

"Flocat":"lto-ltfs:LTFSA001:5778a231-dc8d-4685-9537-0f3d184a9099/sampledir/dir\_to\_archive/sample.txt",

"checksum":{

"type":"SHA1",

"expectedvalue":"32127a6cd22cd0dfceb996efdfc843e285f0f825",

"value":"32127a6cd22cd0dfceb996efdfc843e285f0f825",

"match":"true"

}

},

{

"Flocat":"lto-ltfs:LTFSA001:5778a231-dc8d-4685-9537-0f3d184a9099/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.txt",

"checksum":{

"type":"SHA1",

"expectedvalue":"32127a6cd22cd0dfceb996efdfc843e285f0f825",

"value":"32127a6cd22cd0dfceb996efdfc843e285f0f825",

"match":"true"

}

},

{

"Flocat":"lto-ltfs:LTFSA001:5778a231-dc8d-4685-9537-0f3d184a9099/sampledir/dir\_to\_archive/sample.bin.corrupted",

"checksum":{

"type":"SHA1",

"expectedvalue":"f4b3bd12d724d552c10b4e18a82097952052e8af",

"value":"f4b3bd12d724d552c10b4e18a82097952052e8a2",

"match":"false"

}

},

{

"Flocat":"lto-ltfs:LTFSA001:5778a231-dc8d-4685-9537-0f3d184a9099/sampledir/dir\_to\_archive/sub\_dir with\_space/sample.bin",

"checksum":{

"type":"SHA1",

"expectedvalue":"f4b3bd12d724d552c10b4e18a82097952052e8a2",

"value":"f4b3bd12d724d552c10b4e18a82097952052e8a2",

"match":"true"

}

}

]

}

In this case the failing files are archived with a supplementary “.corrupted” extension, and the TaskID will be declared in failure status. In the report the errorcode assumes the value 500 and the mismatch of checksums is clearly declared.

The task CANNOT be resubmitted. It has to be deleted (using the Cancel command) and a brand new request must be performed after having fixed the checksum file.