

## **GRAU Archive Manager**

# **HSM-API**

## **HSM-API Specification Technical Reference**

---

Produced by	Eberhard Leba
Checked by	Ulrich Lechner
Version	1
Created on	07.Jan.2009
Updated on	9-Mar-09
Approved by	Ulrich Lechner
Archived on	D:\projects\cvshsl\GAM\SNAPPER\jvd- doc\Project\snapper\concepts\HSMAPISPEC.doc

---

### **Abstract:**

This HSM-API specification document describes the public interface to the GRAU Archive Manager software.



**Table of contents**

<b>1</b>	<b>HSM-API Versioning</b>	<b>5</b>
<b>2</b>	<b>API Definition</b>	<b>7</b>
	Overview	7
	Helper Functions 1.0	8
	HSM-API 1.0	8
	Class Properties .....	8
	Class Methods .....	8
	GetHsmApiVersion 1.0 .....	8
	GetHsmVersion 1.0.....	8
	Partition	9
	Structure definitions.....	10
	LocationData 1.0 .....	10
	MediaData 1.0 .....	11
	FielData 1.0.....	12
	Class Methods .....	14
	GetPathSeparator 1.0 .....	14
	GetMountPoint 1.0.....	14
	GetPartitionStatus 1.0.....	14
	LocalPath2HSM 1.0.....	14
	MigrateFile 1.0.....	14
	CreateMajorCollocationID 1.0 .....	15
	EnumMinorCollocatioID 1.0.....	16
	JobStatus 1.0 .....	16
	GetFileStatus 1.1 .....	16
	GetFileLocations 1.0 .....	16
	GetVolumeList 1.0, 1.1 .....	16
	GetVolumeFileList 1.1 .....	17
	ReorgScan 1.1 .....	17
	ReorganizeMedium 1.1.....	17
<b>3</b>	<b>References</b>	<b>20</b>

**Document history**

Date	Version	Author	Description
07.01.2009	Draft	Eberhard Leba	Fixed CI settings. Added additional parameters for HSM-API constructor

## 1 HSM-API Versioning

---

HSM-API development is done in several steps to get productive as soon as possible. Furthermore we requests for new member functions drop in and HSM-API versioning is provided to provide the information whether a particular function will be available or not.

An application should first check the HSM-API version before assuming a function to be available. Below we list the functions that will be available in a particular version.

Item/ Version	1.0	1.1
<b>HSM-API Class</b>	X	X
GetHsmApiVersion	X	X
GetHsmVersion	X	X
<b>Partition Class</b>	X	X
GetPathSeparator	X	X
GetMountPoint	X	X
GetPartitionStatus	X	X
LocalPath2HSM	X	X
MigrateFile	X	X
CreateMajorCollocationID	X	X
EnumMinorCollocationID	X	X
JobStatus	X	X
GetFileStatus		X
GetFileLocations	X	X
GetVolumeList		X
GetVolumeFileList		X
ReorgScan		X
ReorganizeMedium		X



## 2 API Definition

---

### Overview

---

The API is delivered on the base of C++ as **DLL** (Dynamic Link Library) in Windows environments and **so** (shared object) in linux environments. It contains the base HSM-API class encapsulating important configuration parameters as e.g the API version or the GAM version of the target system. The HSM-API class instantiates a GAM Server object. The name of the server is provided as a parameter to the class.

The x.x information provided in the topic reflects the API version providing the method or feature

All methods just returning a class attribute return the attribute itself as a literal value. A Returned uint64 value represents an error code if it is not zero.

#### IN, OUT, INOUT Definitions

IN :            Input parameter  
OUT :          Output parameter  
INOUT :       Input and Output parameter

Results are returned in the variables marked as OUT or INOUT.

Strings are always coded in UTF8 code set.

File names provided to and returned from HSM-API are full qualified local path including the mount point and based on the reference mount point of the application client system.

The reference mount point is provided at instantiation of the HSMpartition object together with its HSM reference path equivalent. The mount point of the HSM file system must be part of the HSM reference path.

This mount point could be obtained from the HSM-API. The file name must not have a leading path separator.

#### Example

```
GAM Client:  C:\hsm\hsmfs1\share1\dir1\dir11\file1.txt
Mount point: C:\hsm\hsmfs1
View on the Application server
              C:\app\dir1\dir11\file1.txt
Path provided to HSM-API:
              C:\app\dir1\dir11\file1.txt
Instantiation:
  Reference mount point: C:\hsm\app
  HSM Reference path: C:\hsm\hsmfs1\share1
```

The path separators used are handled by HSM-API.

#### Example

```
GAM Client:  /usr/local/hsm/hsmfs1/share1/dir1/dir11/file1.txt
Mount point: /usr/local/hsm/hsmfs1
View on the Application server
              C:\app\dir1\dir11\file1.txt
Path provided to HSM-API:
              C:\app\dir1\dir11\file1.txt
```

Partition methods that change or query file system related information need to check that the partition has the status „mounted“. Those methods return ie\_NOMOUNT if the partition is not in the mounted state.

## Helper Functions 1.0

---

Since instantiation of a class returns the class itself the HSM API provides a helper function to gather the returncode of the instantiation itself.

//Function returns the last error and clears the error variable

```
HSM API_API UInt64_t HsmGetLastError(void);
```

If this function returns a non zero value a class was not fully instantiated and is not usable. The class object however may exist in that case.

## HSM API 1.0

---

```
class hsmapi(IN const char *aServerAddress, IN const char *aHostname, IN const char *aLogDir )
```

aServerAddress: Host name identifying the server on the network. Note that HSM API uses CORBA (omniORB) to communicate to the server. If there is any omniORB.cfg profile on the system using the API, check that the profile provides a communication path to the GAM server.

aHostname: The name of the host HSM API is running on.

aLogDir: The directory where the GAM Logs should be put to. GAM creates files named "ivd.log" and "error.log" in this directory.

### Class Properties

There are no public class properties exported

### Class Methods

#### GetHsmApiVersion 1.0

```
string GetHsmApiVersion( void )
```

#### GetHsmVersion 1.0

```
string GetHsmVersion( void )
```



### Partition

---

```
class HsmPartition(IN hsmapi& hsm,  
                  IN const string& aPartitionName,  
                  IN const string& aRefPath,  
                  IN const string& aHSMRefPath)
```

hsm : Previously instantiated hsmapi object.

aPartitionName: GAM Partition name

aRefPath: Local reference mount point where the GAM partition share is mounted

aHSMRefPath: HSM reference path, the equivalent for aRefPath on the GAM system. The HSMFS mountpoint must be a subset of that path. HSM-API will otherwise return ie\_INVALID\_ARG on the mandatory HsmGetLastError() call.

**Note: If the HsmGetLastError() call returns a non zero value, the partition object is not usable. Please abort the operation and delete the instance in this case.**

## Structure definitions

### LocationData 1.0

```
typedef struct LocationData {  
  
    UInt64_t ui64MediumId;    // index of the medium in the GAM system  
  
    string aBarcode;         // Barcode label  
  
    UInt64_t ui64Volume;      // index of the media volume or virtual volume  
  
    UInt64_t ui64NominalCapacity; // native nominal capacity of the medium  
  
    UInt64_t usedCapacity;     // occupied capacity in mega byte  
  
    UInt64_t SlackSpace;       // capacity that could be gained in a reorganization  
  
    UInt64_t MediaStatus;      // Status flags of the whole medium vector  
  
    UInt64_t CollocationIdMajorList; // list of collocation groups held by the  
  
        // data on the volume  
  
    vector<UInt64_t> aui64CollocationIdMinorList; // list of collocation sets held by the  
  
        // data on one volume  
  
} LocationData_t; *LocationData_pt;
```

### MediaData 1.0

```
typedef struct MediaData {  
    UInt64_t ui64MediumId;    // index of the medium in the GAM system  
    string Barcode;          // Barcode label  
    UInt64_t ui64Volume;      // index of the media volume or virtual volume  
    UInt64_t ui64NominalCapacity; // native nominal capacity of the medium  
    UInt64_t ui64UsedCapacity; // occupied capacity in mega byte  
    UInt64_t ui64SlackSpace;  // capacity that could be gained in a reorganization  
    UInt64_t ui64MediaStatus; // Status flags of the whole medium  
    vector <UInt64_t> ui64CollocationIdMajorList; // list of collocation groups held by  
                                                // the data on the volume  
    vector <UInt64_t> ui64CollocationIdMinorList; // list of collocation sets held by the  
                                                // data on one volume  
} MediaData_t; *tMediaData_pt;
```

**FileData 1.0**

```
typedef struct FileData {  
    string Filename;           // full qualified path relative to the GAM mountpoint  
                                // without leading path separator  
    vector<LocationData_t> locationdata; // Positions of the file in the back end storage  
} FileData_t, FileData_pt;
```

## GRAU Archive Manager



HSM-API – HSM-API Specification Technical Reference -  
Specification

### Class Properties

There are no public properties exported.

## Class Methods

### GetPathSeparator 1.0

**string GetPathSeparator( void )**

get the path separator

### GetMountPoint 1.0

**string GetMountPoint( void )**

get the mountpoint on the GAM client system

### GetPartitionStatus 1.0

**string GetPartitionStatus( void )**

get the current status of a GAM partition

### LocalPath2HSM 1.0

**string LocalPath2HSM(const string localpath)**

returns the HSMFS representation of a local path.

### MigrateFile 1.0

**UInt64\_t MigrateFile( IN const string& localpath, OUT vector <string> JobIDList, UInt64\_t flags)**

**UInt64\_t MigrateFile(IN const string& localpath, IN UInt64\_t MajorCollocationID, IN UInt64\_t MinorCollocationID, IN UInt64\_t flags, OUT vector <string>& JobIDList)**

Migrates one or more files, matching the pattern of the given path to one or more volumes not being part of another collection set bound to the collocation set specified by the MinorCollocationID and the MajorCollocationID. Migration in terms of GAM means copying the data to the back end storage locations. Only files in unknown or dirty status are migrated. Online or offline files are ignored ( because the copies on the backend storage are already in place).

**path:** full qualified path based on the content of szMountPoint without leading path separator

**MinorCollocationID:** Identifier for a collocation set defined by the application. If the MinorCollocationID does not yet exist in the system it searches for a new free volume for each copy to place the migration there. If the MinorCollocationID exists, it uses the volume where the last data of that particular MinorCollocationID has been placed before. If there is no free volume available for one of the required copies the EXCLUSIVE\_VOLUME\_USAGE flags are checked and based on that the operation is rejected with E\_NO\_VOLUME\_AVAILABLE error code. No migration will take place at all in this case. Otherwise the system places the new collocation set on the volume where the last data for members of this MajorCollocationID has been placed. The MinorCollocationID then becomes automatically a member of the group specified by the MajorCollocationID. If members of the MajorCollocationID exist on more than one volume it checks the occupation of those volumes and uses the volume with the most free space. If the system is allowed to place the new collocation set anywhere it checks the occupation of the volumes and uses the volume with the most free space.

**MajorCollocationID: The mandatory** Identifier for a collocation group. The parameter has an effect only for the new invented MinorCollocationID. Use create MajorCollocationID method to create a new MajorCollocationID.

**JobIDList:** List of job IDs generated as a result of this migration call. The migration jobs are running asynchronously. Multiple calls using the same pattern in a row while previous migration jobs are not

finished yet may lead to multiple migrations of the same Data. The application needs to make sure that all corresponding jobs have finished before issuing such a call.

**Flags:** Options to be set for the migration call

HSM\_RECURSIVE :

A tree walk should take place to collect all matching files below the given directory.

HSM\_EXCLUSIVE\_VOLUME\_USAGE\_MINOR\_ALLMEDIA :

In case a new MinorCollocation ID is invented and there is no new volume of any type available to exclusively hold the items of a migration of that collocation the method returns E\_NO\_VOLUME\_AVAILABLE error code immediately. **This flag is a superset of any EXCLUSIVE\_VOLUME\_USAGE\_MINOR flag.** The application may then decide how to proceed. The flag has an effect in that particular case only. If a medium volume runs out of space during migration and there is no new volume available the job will be waiting for resources. Once the administrator has added a new free volume the job will write the data and finish.

HSM\_EXCLUSIVE\_VOLUME\_USAGE\_MAJOR\_ALLMEDIA :

In case a new MinorCollocation ID is invented and there is no new volume of any type available to exclusively hold the items of a migration of that collocation in the group specified by the MajorCollocationId the method returns E\_NO\_VOLUME\_AVAILABLE error code immediately. **This flag is a superset of any EXCLUSIVE\_VOLUME\_USAGE\_MAJOR flag.** The application may then decide how to proceed. The flag has an effect in that particular case only. If a medium volume runs out of space during migration and there is no new volume available the job will be waiting for resources. Once the administrator has added a new free volume the job will write the data and finish.

HSM\_EXCLUSIVE\_VOLUME\_USAGE\_MINOR\_DISKMEDIA :

In case a new MinorCollocation ID is invented and there is no new volume of **type disk media** in the group specified by the MajorCollocationId available to exclusively hold the items of a migration of that Minor collocation ID the method returns E\_NO\_VOLUME\_AVAILABLE error code immediately. The application may then decide how to proceed. The flag has an effect in that particular case only. For additional copies the system is allowed to place the collocation set anywhere on a media type other than disk but near members of the MajorCollocationID group. If a medium volume runs out of space during migration and there is no new volume available the job will be waiting for resources. Once the administrator has added a new free volume the job will write the data and finish.

HSM\_EXCLUSIVE\_VOLUME\_USAGE\_MAJOR\_DISKMEDIA :

In case a new MinorCollocation ID is invented and there is no new volume of **type disk media** available to exclusively hold the items of a migration of that Minor collocation ID the method returns E\_NO\_VOLUME\_AVAILABLE error code immediately. The application may then decide how to proceed. The flag has an effect in that particular case only. For additional copies the system is allowed to place the collocation set anywhere on a media type other than disk but near members of the MajorCollocationID group. If a medium volume runs out of space during migration and there is no new volume available the job will be waiting for resources. Once the administrator has added a new free volume the job will write the data and finish.

## CreateMajorCollocationID 1.0

**UInt64\_t CreateMajorCollocationID(UInt64\_t MajorCollocationID, UInt64\_t MinimumGroupSize )**

This method creates a MajorCollocationID to build up a collocation group. If the MajorCollocationID already exist an error code E\_GROUP\_EXISTS is returned. If the MinimumGroupSize Parameter is non zero it specifies the minimum Size of a Group, it uses a default size defined during GAM setup

otherwise. The system virtually allocates enough volumes on each media pool of a GAM partition so that the given size fits in. Note that the size of the group may grow as long as there are enough free non assigned volumes. If the system runs out of media in one of the pools during migration the system will wait for resources. The system writes a warning into its log each time it allocates a new free volume for the group, reminding the administrator to provide new media.

**MajorCollocationID:** Identifier for a new collocation group.

**MinimumGroupSize:** Minimum .size of a collocation group in Megabyte

**EnumMinorCollocationID 1.0**

**UInt64\_t EnumMinorCollocationID( IN UInt64\_t MajorCollocationID,  
OUT vector <UInt64\_t>& aMinorCollocationIDList )**

Enumerates all MinorCollocationIDs grouped in a MajorCollocationID.

**MajorCollocationID:** Identifier for a collocation group.

**MinorCollocationIDList:** List of identifiers for collocation sets.

**JobStatus 1.0**

**UInt64\_t JobStatus( IN vector <string> JobIDList, OUT vector <UInt64\_t>& aStatusList )**

Returns the status of the jobs in the given list ( pending, waiting for resources, in progress, finished, unknown ) The status values are ordered in the same sequence as the job IDs in the input list. If a job does not yet exist or has vanished in the meantime a status unknown is returned in the list.

**JobIDList:** A list of job IDs to be queried.

**StatusList:** A list of job status ordered in the same sequence as the JobIDList.

**GetFileStatus 1.1**

**UInt64\_t GetFileStatus( IN const string& localFilePath, OUT UInt64\_t & status )**

Returns the status of a given file ( unknown, dirty, online, offline )

**LocalFilePath:** full qualified local path.

**status :** a literal reflecting the status of the given file.

**GetFileLocations 1.0**

**UInt64\_t GetFileLocations( IN const string LocalFilePath&, OUT vector <LocationData\_t>& aLocationData )**

Returns all the locations of the last generation of a given File on the backend storage.

**LocalFilePath:** full qualified local path.

**LocationData:** structure holding the location data of a file

**GetVolumeList 1.0, 1.1**

Available with 1.0 :

**UInt64\_t GetVolumeList( OUT vector <MediaData\_t>& aMediaData )**

**UInt64\_t GetVolumeListMinor( IN UInt64\_t MinorCollocationID, OUT vector <MediaData\_t>& aMediaData )**

Available with 1.1 :

**uint64 GetVolumeListMajor( IN uint64 MajorCollocationID, OUT vector <MediaData\_t>& aMediaData )**



GetVolumeList enumerates all media volumes belonging to a partition. The GetVolumeListMinor returns the list of volumes just containing data of that collocation ID. The GetVolumeListMajor returns the list of all volumes containing data in the group defined by the MajorCollectionID. If the collocation ID is unknown to the system an empty list is returned. There is no error code returned in this particular case.

**aMediaData:** List of structures holding the data of a media volumes including the status and meta data of the media the volume is located in.

**MajorCollocationID:** Identifier for a collocation group.

**MinorCollocationID:** Identifier for a collocation set.

#### GetVolumeFileList 1.1

**UInt64\_t GetVolumeFileList( IN uint64 MediumID, IN uint64 volumeID, OUT vector <fileData\_t>& a fileData )**

Returns all files currently located on the given volume. Note that the file in the list may have only an older generation or a split located on this volume. A generation is hereby an older version of a file that has been once overwritten. A split is a part of a file, because the file did not fit into the volume as a whole at all or into the remaining space of a volume. The list returned may be rather huge holding millions of files.

**MediumID :** index of the medium the volume is located on.

**volumeID :** index of the media volume of interest.

**aFileData:** List of structures holding the file related information

#### ReorgScan 1.1

**UInt64\_t ReorgScan( void )**

Scans the GAM internal database (FSC) to get reorganisation statistics for each media volume located in a partition. The result could be enumerated using the GetVolumeList methods. A call of this method is mandatory for a later call of the ReorganizeMedium method. It prepares internal lists required for the reorganization process for each particular volume.

#### ReorganizeMedium 1.1

**UInt64\_t ReorganizeMedium( IN UInt64\_t MediumId )**

**UInt64\_t ReorganizeMedium( IN UInt64\_t MediumId, IN UInt64\_t volumeID )**

Reorganizes a medium as a whole including all media volumes located on that medium if the media volume is not provided. In this case as a result the medium is initialized and marked free. In case a VolumeID was given, just that volume will be reorganized and finally initialized. During the reorganization process all data that has been identified as still valid will be migrated to other volumes. The call of this method requires that the ReorgStat method has been called upfront and that there was no append to the medium data in the mean time. If this condition is not met an error code will be returned.

**MediumID :** index of the medium to be reorganized.

**volumeID :** index of the media volume to be reorganized.





### 3 References

---

- [1] **GAM user documentation 3.5.0**
- [2] **Collocation FURPS Document**
- [3] **Collocation ERS Document**