



Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2.2.2

Specification of AXMEDIS Command Manager, first update of DE3.1.2 part B

Version: 1.6

Date: 09-05-2006

Responsible: DSI (Rogai) (verified and approved by coordinator)

Project Number: IST-2-511299

Project Title: AXMEDIS

Deliverable Type: report

Visible to User Groups: yes

Visible to Affiliated: yes

Visible to the Public: yes

Deliverable Number: DE3.1.2.2.2

Contractual Date of Delivery: M18

Actual Date of Delivery: 10/05/2006

Title of Deliverable: DE3.1.2.2.2 Specification of AXMEDIS Command Manager, first update of DE3.1.2 part B

Work-Package contributing to the Deliverable: WP3.1

Task contributing to the Deliverable: WP3, WP2

Nature of the Deliverable: report

Author(s): DSI

Abstract: this part includes the specification of components, formats, databases and protocol related to the AXMEDIS Framework area AXMEDIS Object Model including only details on Command Manager and its usage, the usage of the AXOM

Keyword List: AXOM, AXMEDIS Command Manager, MPEG-21 models, authoring tools and players.

AXMEDIS Copyright Notice

The following terms (including future possible amendments) set out the rights and obligations licensee will be requested to accept on entering into possession of any official AXMEDIS document either by downloading it from the web site or by any other means.

Any relevant AXMEDIS document includes this license. PLEASE READ THE FOLLOWING TERMS CAREFULLY AS THEY HAVE TO BE ACCEPTED PRIOR TO READING/USE OF THE DOCUMENT.

1. DEFINITIONS

- i. **"Acceptance Date"** is the date on which these terms and conditions for entering into possession of the document have been accepted.
- ii. **"Copyright"** stands for any content, document or portion of it that is covered by the copyright disclaimer in a Document.
- iii. **"Licensor"** is AXMEDIS Consortium as a de-facto consortium of the EC project and any of its derivations in terms of companies and/or associations, see www.axmedis.org
- iv. **"Document"** means the information contained in any electronic file, which has been published by the Licensor's as AXMEDIS official document and listed in the web site mentioned above or available by any other means.
- v. **"Works"** means any works created by the licensee, which reproduce a Document or any of its part.

2. LICENCE

1. The Licensor grants a non-exclusive royalty free licence to reproduce and use the Documents subject to present terms and conditions (the **Licence**) for the parts that are own and proprietary property the of AXMEDIS consortium or its members.
2. In consideration of the Licensor granting the Licence, licensee agrees to adhere to the following terms and conditions.

3. TERM AND TERMINATION

1. Granted Licence shall commence on Acceptance Date.
2. Granted Licence will terminate automatically if licensee fails to comply with any of the terms and conditions of this Licence.
3. Termination of this Licence does not affect either party's accrued rights and obligations as at the date of termination.
4. Upon termination of this Licence for whatever reason, licensee shall cease to make any use of the accessed Copyright.
5. All provisions of this Licence, which are necessary for the interpretation or enforcement of a party's rights or obligations, shall survive termination of this Licence and shall continue in full force and effect.
6. Notwithstanding License termination, confidentiality clauses related to any content, document or part of it as stated in the document itself will remain in force for a period of 5 years after license issue date or the period stated in the document whichever is the longer.

4. USE

1. Licensee shall not breach or denigrate the integrity of the Copyright Notice and in particular shall not:
 - i. remove this Copyright Notice on a Document or any of its reproduction in any form in which those may be achieved;
 - ii. change or remove the title of a Document;
 - iii. use all or any part of a Document as part of a specification or standard not emanating from the Licensor without the prior written consent of the Licensor; or
 - iv. do or permit others to do any act or omission in relation to a Document which is contrary to the rights and obligations as stated in the present license and agreed with the Licensor

5. COPYRIGHT NOTICES

1. All Works shall bear a clear notice asserting the Licensor's Copyright. The notice shall use the wording employed by the Licensor in its own copyright notice unless the Licensor otherwise instructs licensees.

6. WARRANTY

1. The Licensor warrants the licensee that the present licence is issued on the basis of full Copyright ownership or re-licensing agreements granting the Licensor full licensing and enforcement power.
2. For the avoidance of doubt the licensee should be aware that although the Copyright in the documents is given under warranty this warranty does not extend to the content of any document which may contain references or specifications or technologies that are covered by patents (also of third parties) or that refer to other standards. AXMEDIS is not responsible and does not guarantee that the information contained in the document is fully proprietary of AXMEDIS consortium and/or partners.
3. Licensee hereby undertakes to the Licensor that he will, without prejudice to any other right of action which the Licensor may have, at all times keep the Licensor fully and effectively indemnified against all and any liability (which liability shall include, without limitation, all losses, costs, claims, expenses, demands, actions, damages, legal and other professional fees and expenses on a full indemnity basis) which the Licensor may suffer or incur as a result of, or by reason of, any breach or non-fulfillment of any of his obligations in respect of this License.

7. INFRINGEMENT

1. Licensee undertakes to notify promptly the Licensor of any threatened or actual infringement of the Copyright which comes to licensee notice and shall, at the Licensor's request and expense, do all such things as are reasonably necessary to defend and enforce the Licensor's rights in the Copyright.

8. GOVERNING LAW AND JURISDICTION

1. This Licence shall be subject to, and construed and interpreted in accordance with Italian law.
2. The parties irrevocably submit to the exclusive jurisdiction of the Italian Courts.

Please note that:

- You can become affiliated with AXMEDIS. This will give you the access to a huge amount of knowledge, information and source code related to the AXMEDIS Framework. If you are interested please contact P. Nesi at nesi@dsi.unifi.it. Once affiliated with AXMEDIS you will have the possibility of using the AXMEDIS specification and technology for your business.
- You can contribute to the improvement of AXMEDIS documents and specification by sending the contribution to P. Nesi at nesi@dsi.unifi.it
- You can attend AXMEDIS meetings that are open to public, for additional information see WWW.axmedis.org or contact P. Nesi at nesi@dsi.unifi.it

Table of Content

1	EXECUTIVE SUMMARY AND REPORT SCOPE	5
1.1	THIS DOCUMENT CONCERNS (DSI)	6
1.2	LIST OF MODULES OR EXECUTABLE TOOLS SPECIFIED IN THIS DOCUMENT	6
2	GENERAL ARCHITECTURE AND RELATIONSHIPS AMONG THE MODULES PRODUCED.....	7
3	AXMEDIS OBJECT MANAGER (DSI)	8
3.1	GENERAL DESCRIPTION OF THE MODULE.....	9
3.1.1	AXMEDIS Object loading.....	10
3.2	MODULE DESIGN IN TERMS OF CLASSES	10
3.2.1	AxObjectManager Capabilities Overview	11
3.2.2	Class Methods Overview	12
3.3	AXOBJECTMANAGER AS EVENTMANAGER.....	25
3.4	EXAMPLES OF USAGE	27
3.5	ERRORS REPORTED AND THAT MAY OCCUR	28
4	AXOID ASSIGNMENT (DSI)	28
5	OBJECT REGISTRATION (DSI)	28

1 Executive Summary and Report Scope

The full AXMEDIS specification document has been decomposed in the following parts:

DE number	Deliverable title	responsible
DE3.1.2.2.1	Specification of General Aspects of AXMEDIS framework, first update of DE3.1.2 part A AXMEDIS-DE3-1-2-2-1-Spec-of-AX-Gen-Asp-of-AXMEDIS-framework-upA-v1-0.doc	DSI
DE3.1.2.2.2	Specification of AXMEDIS Command Manager, first update of DE3.1.2 part B AXMEDIS- DE3-1-2-2-2-Spec-of-AX-Cmd-Man-upB-v1-0.doc	DSI
DE3.1.2.2.3	Specification of AXMEDIS Object Manager and Protection Processor, first update of DE3.1.2 part B AXMEDIS-DE3-1-2-2-3-Spec-of-AXOM-and-ProtProc-upB-v1-0.doc	DSI
DE3.1.2.2.4	Specification of AXMEDIS Editors and Viewers, first update of DE3.1.2 part B AXMEDIS-DE3-1-2-2-4-Spec-of-AX-Editors-and-Viewers-upB-v1-0.doc	DSI
DE3.1.2.2.5	Specification of External AXMEDIS Editors/Viewers and Players, first update of DE3.1.2 part B AXMEDIS-DE3-1-2-2-5-Spec-of-External-Editors-Viewers-Players-upB-v1-0.doc	EPFL
DE3.1.2.2.6	Specification of AXMEDIS Content Processing, first update of DE3.1.2 part C AXMEDIS-DE3-1-2-2-6-Spec-of-AX-Content-Processing-upC-v1-0.doc	DSI
DE3.1.2.2.7	Specification of AXMEDIS External Processing Algorithms AXMEDIS-DE3-1-2-2-7-Spec-of-AX-External-Processing-Algorithms-v1-0.doc	FHGIGD
DE3.1.2.2.8	Specification of AXMEDIS CMS Crawling Capabilities, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-8-Spec-of-AX-CMS-Crawling-Capab-v1-0.doc	DSI
DE3.1.2.2.9	Specification of AXMEDIS database and query support, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-9-Spec-of-AX-database-and-query-support-v1-0.doc	EXITECH
DE3.1.2.2.10	Specification of AXMEDIS P2P tools, AXEPTTool and AXMEDIS, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-10-Spec-of-AXEPTTool-and-AXMEDIA-tools-v1-0.doc	CRS4
DE3.1.2.2.11	Specification of AXMEDIS Programme and Publication tools, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-11-Spec-of-AX-Progr-and-Pub-tool-v1-0.doc	UNIVLEEDS
DE3.1.2.2.12	Specification of AXMEDIS Workflow Tools, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-12-Spec-of-AX-Workflow-Tools-v1-0.doc	IRC
DE3.1.2.2.13	Specification of AXMEDIS Certifier and Supervisor and networks of AXCS, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-13-Spec-of-AXCS-and-networks-v1-0.doc	DSI
DE3.1.2.2.14	Specification of AXMEDIS Protection Support, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-14-Spec-of-AX-Protection-Support-v1-0.doc	FUPF
DE3.1.2.2.15	Specification of AXMEDIS accounting and reporting, first update of part of DE3.1.2 AXMEDIS-DE3-1-2-2-15-Spec-of-AX-Accounting-and-Reporting-v1-0.doc	EXITECH

1.1 This document concerns (DSI)

AXMEDIS Object Manager, so called AXOM, is the outer module exposing functionalities in order to manipulate AXMEDIS Object (or MPEG-21 Digital Items). It hides all the underlying model for representing loading, saving object content and metadata. This module is the keystone to build any AXMEDIS compliant tools since it grants the developer to correctly manages the underlying content model, while also respecting DRM constraints on the AXMEDIS Object. AXMEDIS Object Manager guarantees DRM rules respect on AXMEDIS object manipulations according to the issued licences. AXMEDIS Object Manager is the sole responsible of command execution (i.e. to command an execution of a desired manipulation), because completion of this task requires features of all other modules in specification.

1.2 List of Modules or Executable Tools Specified in this document

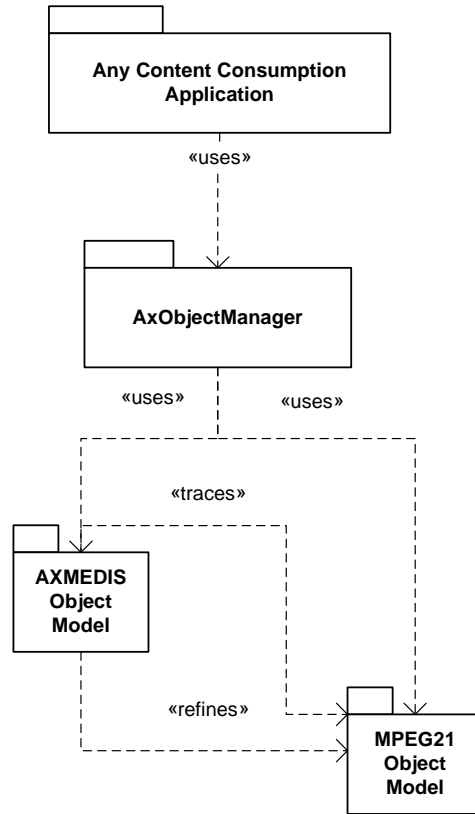
A module is a component that can be or it is reused in other cases or points of the AXMEDIS framework or of other AXMEDIS based solutions.

The modules/tools have to include effective components and/or tools and also testing components and tools.

Module/tool Name	Module/Tool Description and purpose, state also in which other AXMEDIS area is used	Standards exploited if any
AXMEDIS Object Manager	In AXMEDIS Editor, AXMEDIS players, AXMEDIS Content Processing tools, and all tools that use the AXMEDIS object model	MPEG-21 REL/RDD

2 General architecture and relationships among the modules produced

AxObjectManager architecture work in cooperation with many modules involved in manipulating AXMEDIS Objects and to provide at upper level applications useful methods to accomplish all needed tasks in order to manage, modify, and even create, new objects. These interfaces are build in accordance to DRM guidelines and accomplishes all operations enforcing DRM .



This module include several classes . The core is Axmedis Object Manager, that coordinates all other classes and expose methods to upper level applications.

AxIndexManager supports indexing of managed elements, providing retrival functionalities for the entire module.

AxCommand provide a common interface for all the defined commands in the module.

Another concept such as the “event” paradigm as explained in the Observer design pattern has been considered an implemented in the AxObjectManager in order to allow application to effectively manage their rendering of the multimedia package

An infrastructure of classes is related to AxObjectManager allowing event description. The AxObjectManager is responsible for registering the event handler interested to different kind of events: those which notify changes in the package structure and those that notify modification to the embedded digital resources or the related metadata. Both AXMEDIS and MPEG-21 events have been provided.

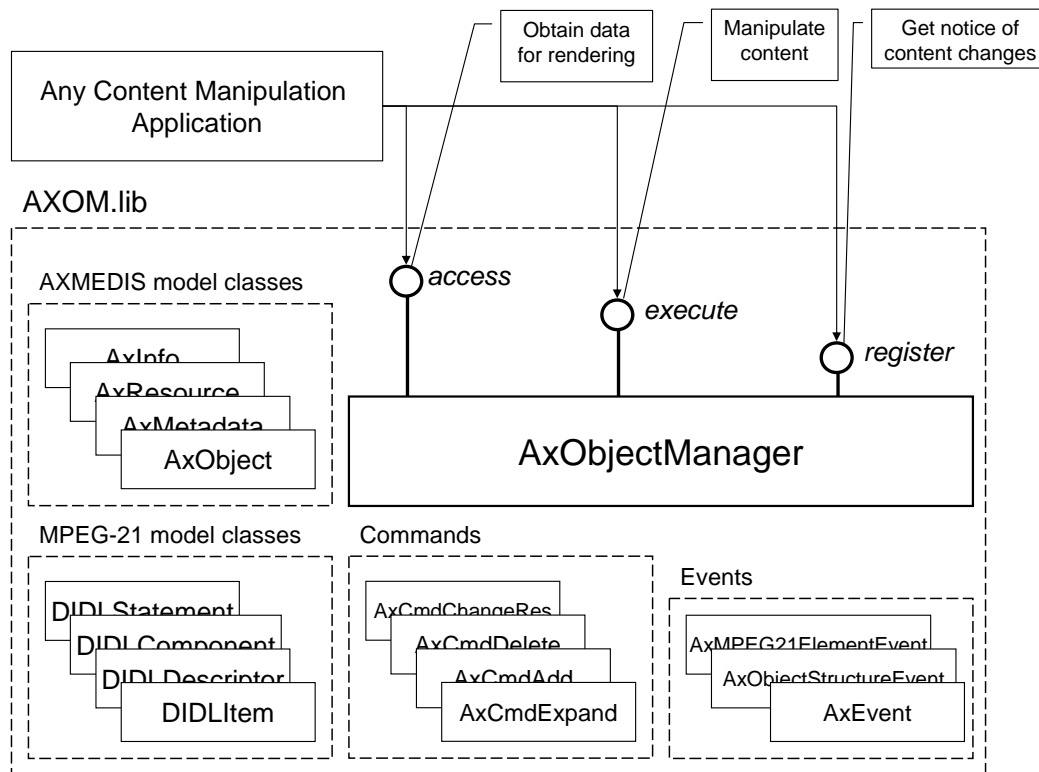
3 AXMEDIS Object Manager (DSI)

Module/Tool Profile		
AXMEDIS Object Manager		
Responsible Name	DAvide Rogai	
Responsible Partner	DSI	
Status (proposed/approved)	Proposed	
Implemented/not implemented	Implemented	
Status of the implementation	90%	
Executable or Library/module (Support)		
Single Thread or Multithread	Multithread	
Language of Development	C++	
Platforms supported	Windows/Unix-Linux	
Reference to the AXFW location of the source code demonstrator	https://cvs.axmedis.org/repos/framework/source/axom	
Reference to the AXFW location of the demonstrator executable tool for internal download	https://cvs.	
Reference to the AXFW location of the demonstrator executable tool for public download		
Address for accessing to WebServices if any, add accession information (user and Passwd) if any		
Test cases (present/absent)		
Test cases location	http://	
Usage of the AXMEDIS configuration manager (yes/no)	Yes	
Usage of the AXMEDIS Error Manager (yes/no)	No	
Major Problems not solved	-- --	
Major pending requirements	-Right Enforcement logic --	
Interfaces API with other tools, named as	Name of the communicating tools References to other major components needed	Communication model and format (protected or not, etc.)
Formats Used	Shared with	format name or reference to a section

Protocol Used	Shared with	Protocol name or reference to a section
Object ID Generation	AXCS	
Object Registration	AXCS	
AXDB Loader	AXDB	
AXDB Saver	AXDB	
Used Database name		
User Interface	Development model, language, etc.	Library used for the development, platform, etc.
Used Libraries	Name of the library and version	License status: GPL. LGPL. PEK, proprietary, authorized or not
libcurl	libcurl-7.15.0	LGPL
gsoap	Gsoap 2.7	LGPL

3.1 General Description of the Module

Object Manager is the interface among objects representation and all others data-manipulation AXMEDIS Editor modules (e.g. View Modules, plug-ins, etc...). Object Manager will provide all base operations (add, change, delete, etc...) which will be needed to manipulate AXMEDIS objects, at all with elements index management and coordination. Moreover, it will invoke, through the use of Protection Processor , Protection Manager Support to verify each operation.



- Object Manager works in respect of DRM model, i.e. on every user action it shall invoke the control of user grants on the involved items. That should be possible through the invocation of Protection Processor (see AXMEDIS-DE3-1-2-2-3)
- Object Manager stores information about taken actions, in particular the following information shall be stored:
 - Kind of action and entities involved;
 - Who takes the action;
 - Where the action have been taken (AXMEDIS Editor installation identifier);
 - When the action have been taken (timestamp);

Object Manger provides an interface to permit development of data-manipulation plug-ins by third party developer. This functionality is implemented in ProtectionProcessor. (see AXMEDIS-DE3-1-2-2-3) and exposed in Object Manager

3.1.1 AXMEDIS Object loading

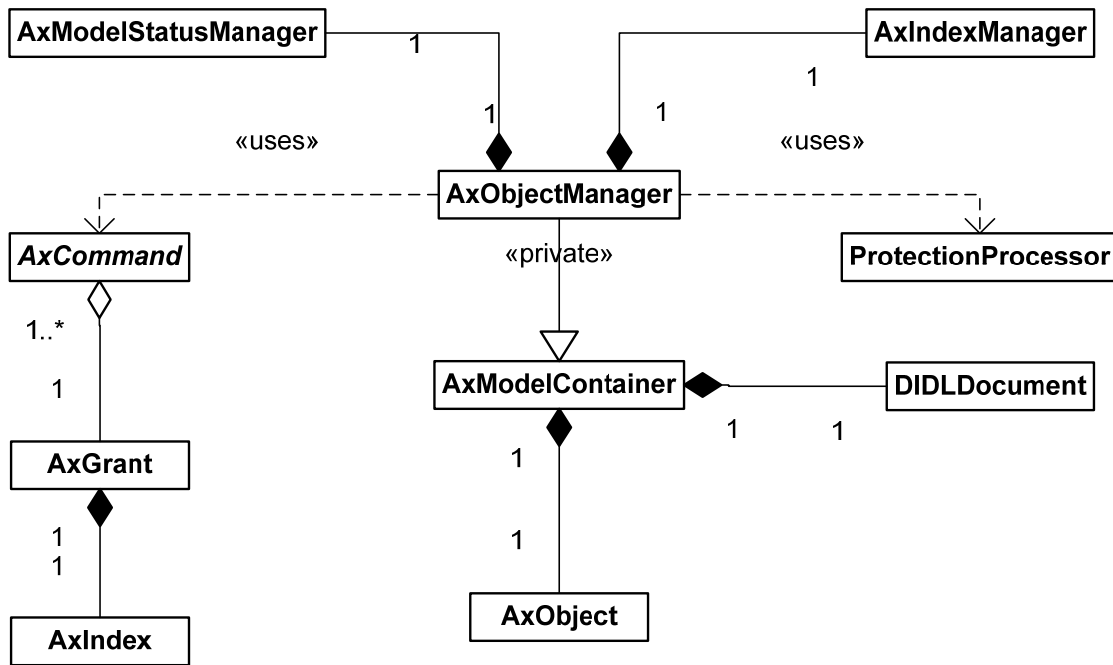
AXMEDIS Object Manager can be created for managing new and existing objects. In case of existing object they can be retrieved by means of different URIs. The supported protocols are:

- **File System:** [file://](#) protocol or a path can be used to locate an object to be loaded an manipulated via AXOM
- **HTTP download:** [http://](#) is used when AXMEDIS object have to be retrieved from the Web
- **AXDB checkout:** a special database protocol as been defined withcorresponding URI type. The syntax is [axdb://<user>:<passwd>@<host>:<port>/<endpoint>?axoid=<axoid>&ver=<version>](#).

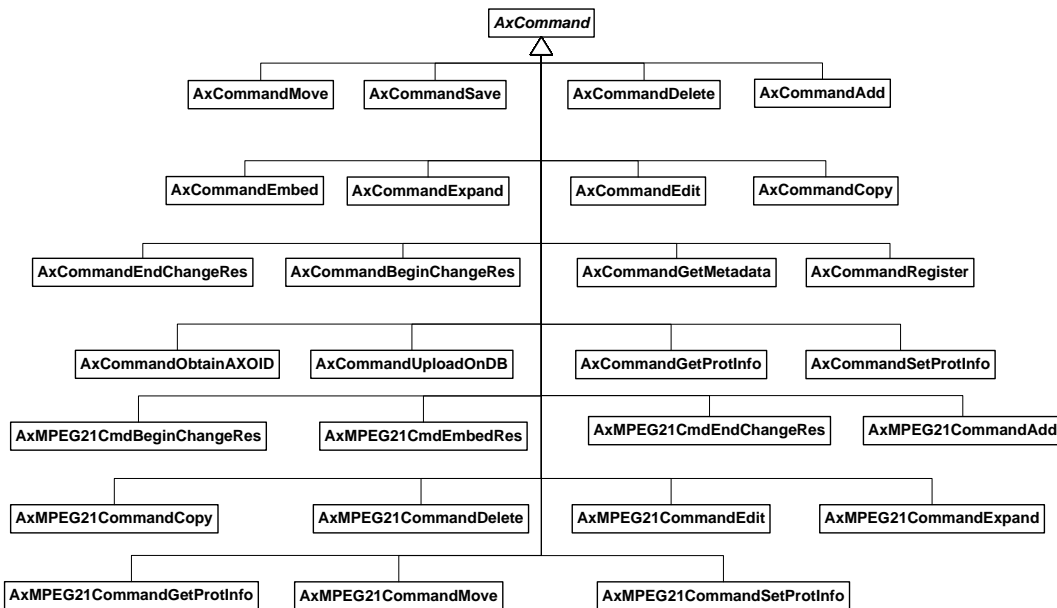
To save an AXMEDIS object or to upload on the AXDB specific command have been designed, for this actions are governed by DRM rules.

3.2 Module Design in terms of Classes

AxObjectManager class is composed and is hard linked to a range of classes that implements needed functionalities. Next Class Diagram shows relations between these classes. AxObjectManager is the core class, this class derives from AxModelContainer that offers functionalities to hold AxObjects and Mpeg21 elements. AxIndexManager is in charge to maintain indexing throughout the object model. AxCommand class and his derived children represent allowed commands exposed to the outer environment. AxModelStatusManager controls status of the elements locking and unlocking Objects



Going deeply we can see how **AxCommand** class is used. From this class indeed derives all allowed commands that could be requested at **AxObjectManager**. These commands share the common interface provided by **AxCommand**, featuring specific methods and data structures to accomplish their tasks.



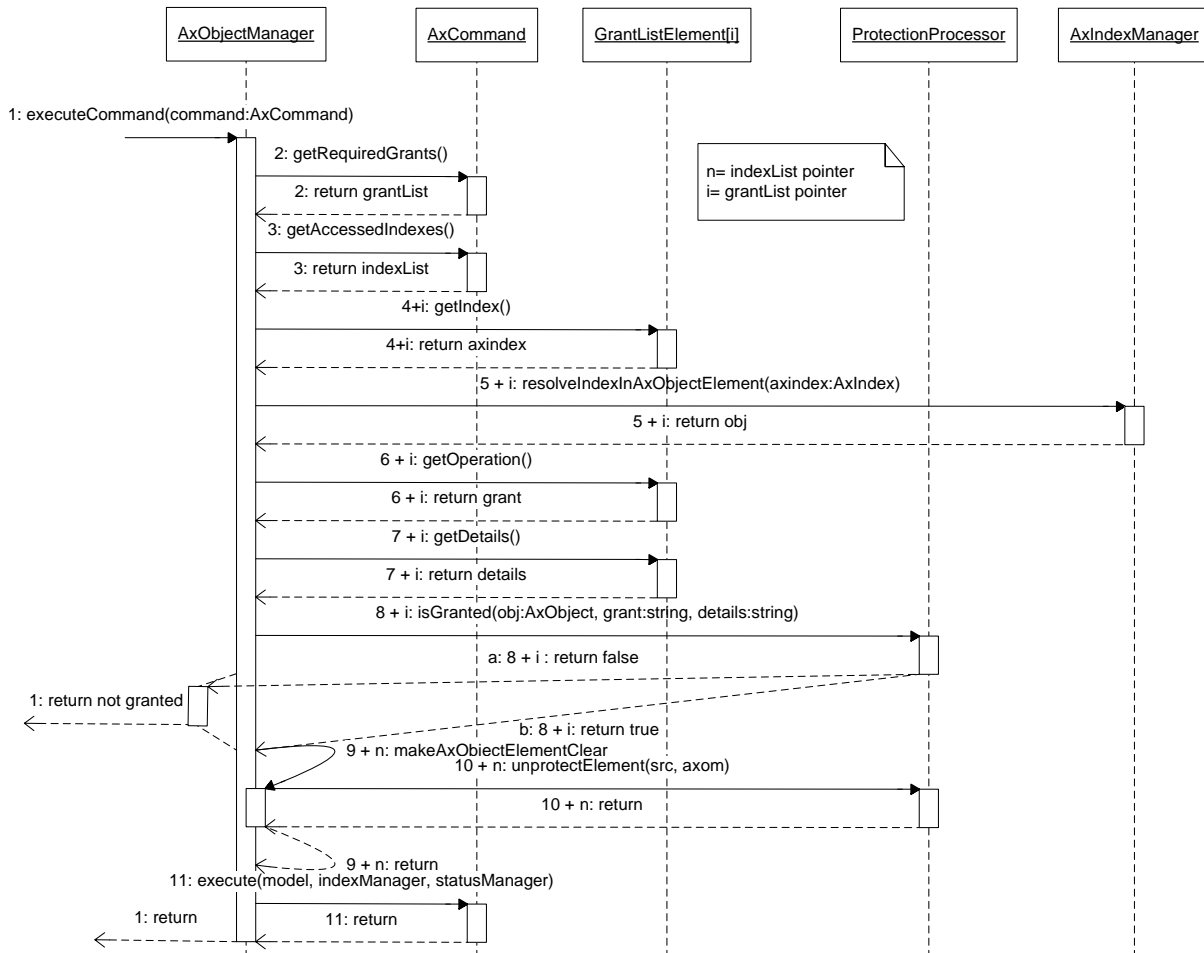
A typical command will override **AxCommand::execute** method to implement the operation sequence needed to perform the task that command models. Other accessory methods could be implemented in the derived command classes.

3.2.1 AxObjectManager Capabilities Overview

The following pictures shows how the entire module works in terms of sequence diagrams. In the first picture an example of command execution is taken. The command, represented by AxCommand class, is passed at AxObjectManager by mean of executeCommand method.

Next operations involves :

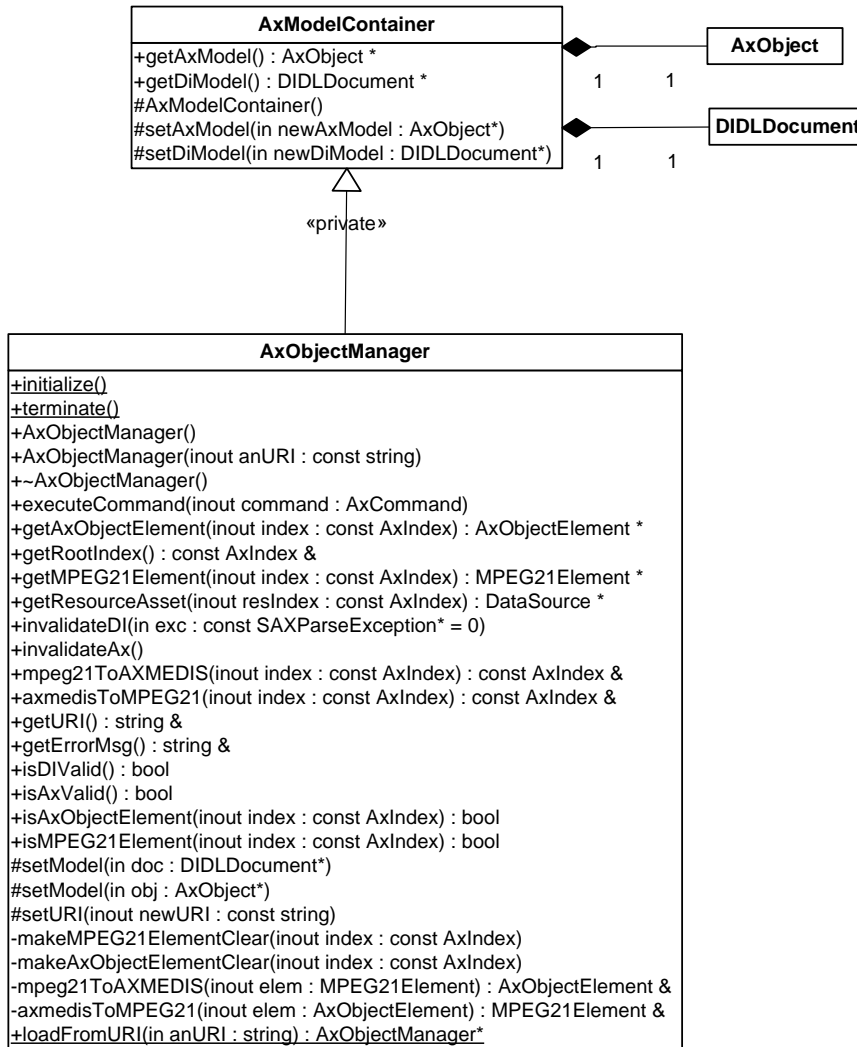
- Getting all grants elements, maintained in AxCommand, for command execution
- Getting all indexes of model elements, maintained in AxCommand, to unprotect for command execution
- Checking all grants for command execution through ProtectionProcessor
- Unprotecting all elements needed for command execution
- Command execution.



3.2.2 Class Methods Overview

AxObjectManager - AxModelContainer

Core class of this module, expose methods to interface outer applications to inner classes of the module. Through AxObjectManager indeed AxIndexManager, AxCommand and AxModelStatusManager are used.



AxObjectManager – Class methods

AxObjectManager - ~AxObjectManager

Class constructor and destructor

Initialize – terminate

These two static methods setup and dismiss all needed information and data structures in order to allow usage of AxObjectManager and linked modules. Initialize has to be called as first step when an application has to use Axmedis Object Model or MPEG-21 object model in any way. Multiple calls of initialize don't cause changes in initialized items. Terminate has to be called whenever an application stops using Axmedis Object Model and MPEG-21 Object Model

executeCommand

Performs operations to allow execution of the input AxCommand., then execute the command

getAxObjectElement – getMPEG21Element - getResourceAsset

Interface to AxIndexManager. Retrives element related to input AxIndex . Search path is chosen in AxObject or DIDLDocument trees respectively. It returns a clone of the encapsulated object, it has to be destroyed by client code.

getRootIndex

Return index of the root element of AxIndexManager. This index points both AxObject and

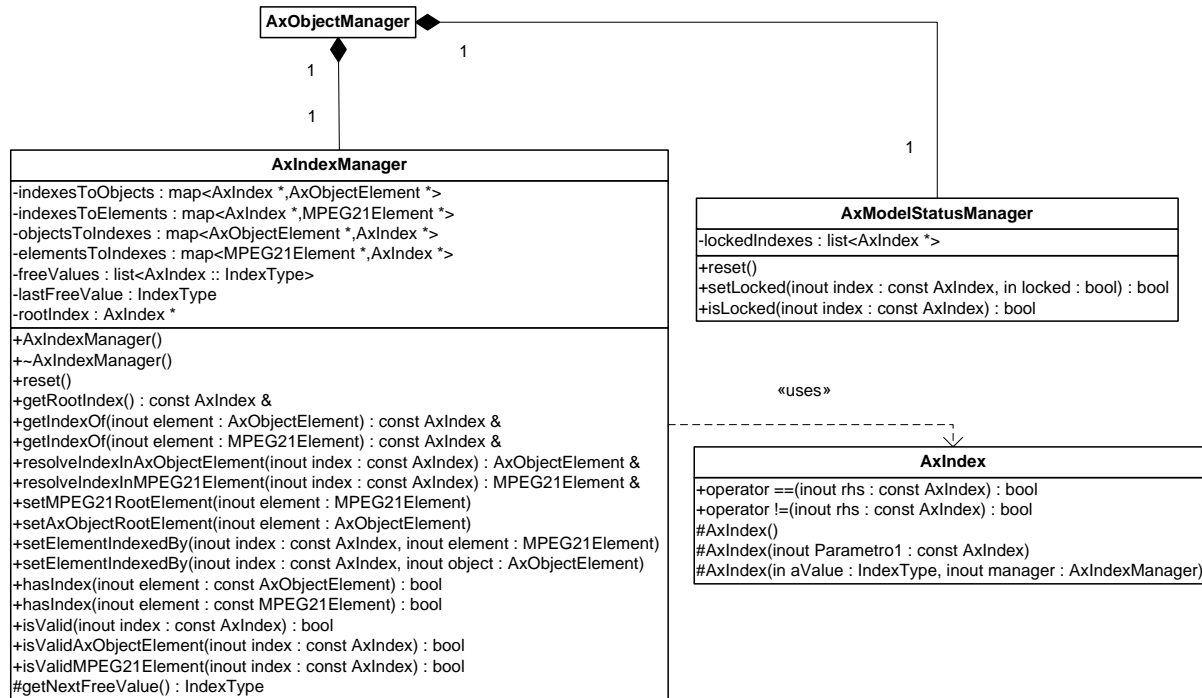
DIDLDocument roots
invalidateDI, invalidateAX
Make Digital Item tree or AxObject tree invalid
mpeg21ToAXMEDIS – axmedisToMPEG21
Convert MPEG21 element to AxObject element and vice versa. Return index of the converted element
getURI – setURI
Returns an set URI for the manager
isDIValid – isAxValid
Check validity status of the models
isMPEG21Element – isAxObjectElement
Checks if the given index refers an MPEG21Element or an AxObjectElement respectively
setModel
Sets the model root index in AxIndexManager
makeMPEG21ElementClear – makeAxObjectElementClear
Unprotect referred elements through the use of ProtectionProcessor
mpeg21ToAXMEDIS – axmedisToMPEG21
loadFromURI
a static methods for loading objects from multiple URI has been provided, in order to avoid a constructor which can fail. By calling this static method a pointer to AxObjectManager ready to managed the loaded document.

AxModelContainer – Class methods
AxModelContainer
Class constructor
getAxModel – getDIModel
Returns AxModel and DIDLDocument root pointers
setAxModel – setDIModel
Sets AxModel root and DIDLDocument root

AxIndexManager – AxIndex - AxModelStatusManager

These classes support AxObjectManager. AxIndexManager is demanded to manage access to data models maintaining indexes for all the elements. The class maintains two different indexes, one for MPEG21Elements and the other for AxObjectElements.

AxModelStatusManager provide functionalities to control the status of the model.

**AxIndex Manager – Class methods****AxIndexManager - ~AxIndexManager**

Class constructor and destructor

reset

Reset all the element index to an empty value. Delete all AxIndex elements in the index

getRootIndex

Return index of the root element of AxIndexManager.

getIndexOf

Get the AxIndex of input element.

resolveIndexInAxObjectElement – resolveIndexInMPEG21Element

Return element associated with input AxIndex

setMPEG21RootElement – setAxObjectRootElement

Sets root for MPEG21Element index and AxObjectElement index

setElementIndexedBy

Add a new entry in one of two indexes chosen by input element

hasIndex

Checks if the given element has an associated AxIndex

isValid – isValidAxObjectElement – isValidMPEG21Element

Check validity status of the element

getNextFreeValue

Returns next AxIndex number free from element associations

AxIndex– Class methods**AxIndex**

Class constructor

operator ==

Checks if two AxIndex are equal

operator !=

Checks if two AxIndex are different

AxModelStatusManager– Class methods**reset**

Reset the class to initial state cleaning all locked indexes

setLocked

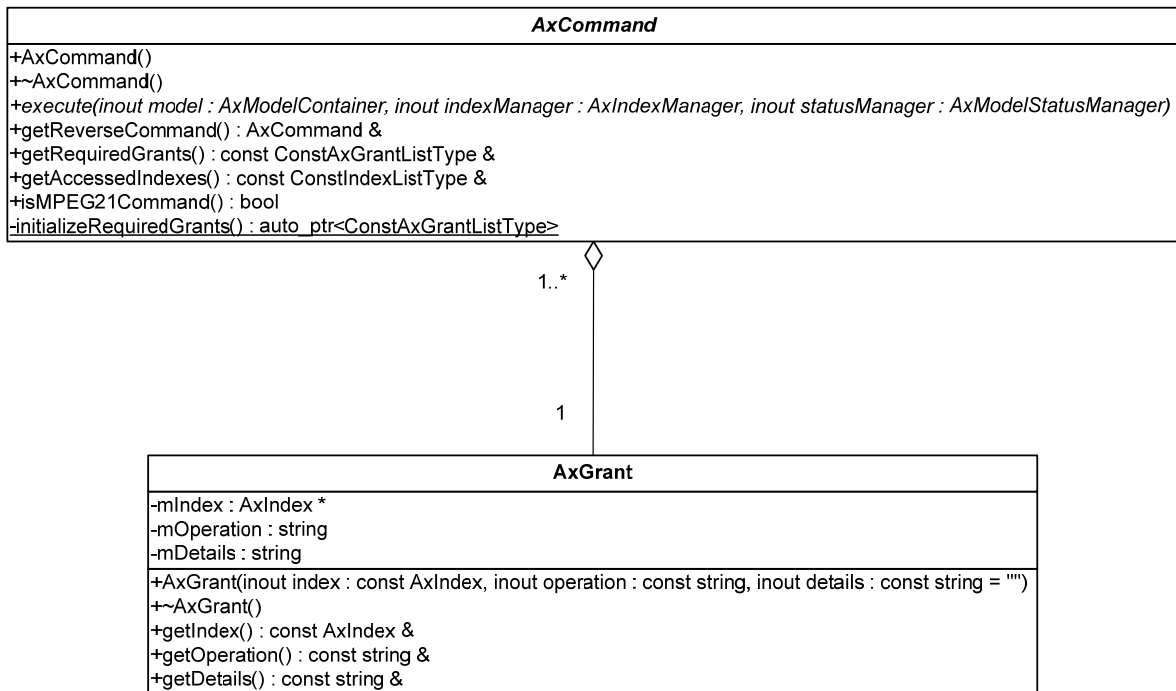
Lock target element

isLocked

Checks if target element is locked

AxCommand – AxGrant

These classes support commands definitions and execution. AxCommand represent the common interface for all the command defined for the models. AxGrant models grants required for command executions. These grants will be checked by Protection Processor (see DE- 3-1-2-2-3- ProtectionProcessor) .

**AxCommand – Class methods****AxCommand - ~AxCommand**

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getReverseCommand

Return the reverse version of this command if a reverse command is supported by the command itself

getRequiredGrants

Returns a list of grants needed for the execution of this command.

getAccessedIndexes

Return a list of AxIndex that refers to all the elements used by the execution of this command

isMPEG21Command

Checksif the command operates on an MPEG21Element

initializeRequiredGrants
Initialize the grant list for the command

AxGrant – Class methods
AxGrant - ~AxGrant
Class constructor and destructor
getIndex – getOperation –getDetails
Returns index , operation name and details for the grant

AxCommands

A list of all implemented command classes is now showed

AxCommandAdd: add a new AxObject to the tree. The object which is passed as an argument to the addition is cloned with deep option set to true.

AxCommandAdd
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxCommandAdd(inout theNewElement : AxObjectElement, inout theParentIndex : const AxIndex)
+AxCommandAdd(inout theNewElement : AxObjectElement, inout theParentIndex : const AxIndex, inout theReferenceIndex : const AxIndex, in insertbefore : bool)
+getIndexOfAddedElement() : const AxIndex &
+getRequiredGrants() : constConstAxGrantListType&

AxCommandAdd – Class methods
AxCommandAdd - ~AxCommandAdd
Class constructor and destructor
Execute
Execute the command in environment defined by input model, indexMannager, statusManager
getIndexeOfAddedElement
Return the index of added element
getRequiredGrants
Return grants needed to command execution

AxCommandBeginChangeRes: Change the AxResource content. This command returns an output stream where the modified resource can be written. The action has to be finalized with AxCommandEndChangeRes.

AxCommandBeginChangeRes
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxCommandBeginChangeRes(inout contentContainerIndex : const AxIndex)
+~AxCommandBeginChangeRes()
+getOutputStream() : ostream &
+getRequiredGrants() : constConstAxGrantListType&

AxCommandBeginChangeRes– Class methods
AxCommandBeginChangeRes - ~AxCommandBeginChangeRes
Class constructor and destructor
Execute
Execute the command in environment defined by input model, indexMannager, statusManager
getOuputStream
Return the output stream of Resource to be changed
getRequiredGrants
Return grants needed to command execution

AxCommandCopy: copy a target AxObject element to a destination

AxMPEG21CommandCopy
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager) +AxMPEG21CommandCopy(inout theSourceIndex : const AxIndex, inout theDestParentIndex : const AxIndex) +getNewElementIndex() : const AxIndex & +getRequiredGrants() : constConstAxGrantListType &

AxCommandCopy – Class methods

AxCommandCopy - ~AxCommandCopy

Class constructor and destructor

Execute

Execute the command in environment defined by input model, indexMannager, statusManager

getNewElementIndex

Return the index of new element

getRequiredGrants

Return grants needed to command execution

AxCommandDelete: delete a target AxObject element

AxCommandDelete
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager) +AxCommandDelete(inout toDeleteIndex : const AxIndex) +getRequiredGrants() : constConstAxGrantListType&

AxCommandDelete – Class methods

AxCommandDelete

Class constructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getRequiredGrants

Return grants needed to command execution

AxCommandEdit: edit an element of the AxObject, this command can be used to modify the attribute of any element in the AxObject (e.g. to modify the mime-type of a resource).

AxCommandEdit
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager) +AxCommandEdit(inout theEditIndex : const AxIndex, inout theDataElement : AxObjectElement) +getRequiredGrants() : constConstAxGrantListType&

AxCommandEdit – Class methods

AxCommandEdit

Class constructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getRequiredGrants

Return grants needed to command execution

AxCommandEmbed: embed a new asset in an AxResource. Similar to the command for changing a resource, this command modify at once all the resource asset. The main difference is that allows to pass an input stream where the command will extract the content.

AxCommandEmbed
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager) +AxCommandEmbed(inout axResourceIndex : const AxIndex, inout assetFileName : const string) +AxCommandEmbed(inout axResourceIndex : const AxIndex, inout assetStream : istream) +~AxCommandEmbed() +getRequiredGrants() : constConstAxGrantListType&

AxCommandEmbed – Class methods
AxCommandEmbed - ~AxCommandEmbed
Class constructor and destructor
execute
Execute the command in environment defined by input model, indexMannager, statusManager
getRequiredGrants
Return grants needed to command execution

AxCommandEndChangeRes: terminate an AxResource change operation. Finalize the changes of a given resource. It has to be call when a resource modification process (beginning with a AxCommandEndChangeRes) is terminates. After its execution the new resource will be embedded as an asset.

AxCommandEndChangeRes
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxCommandEndChangeRes(inout contentContainerIndex : const AxIndex)
+~AxCommandEndChangeRes()
+getRequiredGrants() : constConstAxGrantListType&

AxCommandEndChangeRes – Class methods
AxCommandEndChangeRes - ~AxCommandEndChangeRes
Class constructor and destructor
execute
Execute the command in environment defined by input model, indexMannager, statusManager
getRequiredGrants
Return grants needed to command execution

AxCommandExpand: Returns indexes of target element's children. This command is used to browse the AxObject level by level.

AxCommandExpand
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxCommandExpand(inout expandIndex : const AxIndex)
+~AxCommandExpand()
+getChildrenIndexes() : const vector<AxIndex *> &
+getRequiredGrants() : constConstAxGrantListType&

AxCommandExpand – Class methods
AxCommandExpand - ~AxCommandExpand
Class constructor and destructor
execute
Execute the command in environment defined by input model, indexMannager, statusManager
getChildrenIndexes
Return indexes to children of expanded node
getRequiredGrants
Return grants needed to command execution

AxCommandGetMetadata: return metadata indexes. This command is used to obtain the list of metadata, which are associated to a given AxObject.

AxCommandGetMetadata
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager) +AxCommandGetMetadata(inout getMetadataIndex : const AxIndex, inout ns : const string = "") +~AxCommandGetMetadata() +getMetadataIndexes() : const vector<AxIndex *> & +getRequiredGrants() : constConstAxGrantListType&

AxCommandGetMetadata – Class methods

AxCommandGetMetadata - ~AxCommandGetMetadata

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getMetadataIndexes

Return the indexes of retrived metadata

getRequiredGrants

Return grants needed to command execution

AxCommandGetProtInfo: returns Protection Information for target AxObject

AxCommandGetProtInfo
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager) +AxCommandGetProtInfo(inout index : const AxIndex) +~AxCommandGetProtInfo() +getToolList() : const ConstToolListType & +getRequiredGrants() : constConstAxGrantListType&

AxCommandGetProtInfo – Class methods

AxCommandGetProtInfo - ~AxCommandGetProtInfo

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getToolList

Return a list ot tool types used to process element's protection information

getRequiredGrants

Return grants needed to command execution

AxCommandMove: Move an AxObject element to a destination

AxCommandMove
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager) +AxCommandMove(inout theSourceIndex : const AxIndex, inout theDestParentIndex : const AxIndex) +AxCommandMove(inout theSourceIndex : const AxIndex, inout theDestParentIndex : const AxIndex, inout theDestRefIndex : const AxIndex, in beforeafter : bool) +getRequiredGrants() : constConstAxGrantListType&

AxCommandMove – Class methods

AxCommandMove - ~AxCommandMove

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getRequiredGrants

Return grants needed to command execution

AxCommandObtainAxoid: It contact the suitable service in orded to obtain an AXOID. Inthis way it can be uniquely identified in the AXMEDIS. This is a mandatory step before the publication/distribution.

AxCommandObtainAXOID
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxCommandObtainAXOID(inout index : const AxIndex)
+~AxCommandObtainAXOID()
+getAXOID() : const string &
+getRequiredGrants() : constConstAxGrantListType&

AxCommandObtainAXOID – Class methods

AxCommandObtainAXOID - ~AxCommandObtainAXOID

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getAXOID

Return Axmedis Object ID (AXOID)

getRequiredGrants

Return grants needed to command execution

AxCommandRegister: Register the object. This is a mandatory step before the publication/distribution.

AxCommandRegister
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxCommandRegister()
+~AxCommandRegister()
+getRequiredGrants() : constConstAxGrantListType&

AxCommandRegister – Class methods

AxCommandRegister - ~AxCommandRegister

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getRequiredGrants

Return grants needed to command execution

AxCommandSave: save the object in a output file

AxCommandSave
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxCommandSave(inout destFileName : const string = "")
+getState() : SaveStateType
+getMessage() : const string &
-generateTmpFileName() : string
+getRequiredGrants() : constConstAxGrantListType&

AxCommandSave – Class methods

AxCommandSave

Class constructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getState

Return state of saving process

getMessage

Return information about command execution errors

getRequiredGrants

Return grants needed to command execution

AxCommandSetProtInfo: set protection info for the object

AxCommandSetProtInfo
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxCommandSetProtInfo(inout tbp : const AxIndex, inout tools : const ConstToolListType)
+~AxCommandSetProtInfo()
+getRequiredGrants() : constConstAxGrantListType&

AxCommandSetProtInfo – Class methods

AxCommandSetProtInfo - ~AxCommandSetProtInfo

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getRequiredGrants

Return grants needed to command execution

AxCommandUploadOnDB: save the object in a AXDB

AxCommandUploadOnDB
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxCommandUploadOnDB()
+AxCommandUploadOnDB(in saverEndPoint : string, in ftpPath : string, in user : string, in passwd : string)
+getState() : UploadStateType
+getMessage() : const string &
+getRequiredGrants() : constConstAxGrantListType&

AxCommandUploadOnDB– Class methods

AxCommandUploadOnDB

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getState

Return state of saving process

getMessage

Return information about command execution errors

getRequiredGrants

Return grants needed to command execution

Note: the default constructor will target the “default database” (location will be retrieved by the current configuration). If the extended constructor will be used, the target database is located by the proper information.

AxMPEG21CmdBeginChangeRes: Changes a Resource asset in the MPEG-21 DI. See corresponding command on the AXMEDIS Object.

AxMPEG21CmdBeginChangeRes
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxMPEG21CmdBeginChangeRes(inout contentContainerIndex : const AxIndex)
+~AxMPEG21CmdBeginChangeRes()
+getOutputStream() : ostream &
+getRequiredGrants() : const ConstAxGrantListType &

AxMPEG21CmdBeginChangeRes – Class methods

AxMPEG21CmdBeginChangeRes - ~AxMPEG21CmdBeginChangeRes

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getOutputStream

Return output stream related to new Resource

getRequiredGrants

Return grants needed to command execution

AxMPEG21CmdEmbedRes: load resource asset content in an MPEG-21 DI

AxMPEG21CmdEmbedRes
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxMPEG21CmdEmbedRes(inout contentContainerIndex : const AxIndex, inout streamToBeEmbedded : istream)
+~AxMPEG21CmdEmbedRes()
+getRequiredGrants() : const ConstAxGrantListType &

AxMPEG21CmdEmbedRes – Class methods

AxMPEG21CmdEmbedRes- ~AxMPEG21CmdEmbedRes

Class constructor and destructor

Execute

Execute the command in environment defined by input model, indexMannager, statusManager

getIndexeOfAddedElement

Return the index of added element

AxMPEG21CmdEndChangeRes: End changes of the resource. To be called at the end of a Resource editing which has been started by AxMPEG21CmdBeginChangeRes.

AxMPEG21CmdEndChangeRes
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxMPEG21CmdEndChangeRes(inout contentContainerIndex : const AxIndex)
+~AxMPEG21CmdEndChangeRes()
+getRequiredGrants() : const ConstAxGrantListType &

AxCmdEndChangeRes – Class methods

AxMPEG21CmdEndChangeRes - ~AxMPEG21CmdEndChangeRes

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getRequiredGrants

Return grants needed to command execution

AxMPEG21CommandAdd: add a target MPEG-21 element to an MPEG-21 DI

AxMPEG21CommandAdd
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxMPEG21CommandAdd(inout theNewElement : MPEG21Element, inout theParentIndex : const AxIndex)
+AxMPEG21CommandAdd(inout theNewElement : MPEG21Element, inout theParentIndex : const AxIndex, inout theReferenceIndex : const AxIndex, in beforeafter : bool)
+getIndexOfAddedElement() : const AxIndex &
+getRequiredGrants() : constConstAxGrantListType &

AxMPEG21CommandAdd – Class methods

AxMPEG21CommandAdd - ~AxMPEG21CommandAdd

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getIndexeOfAddedElement

Return the index of added element

getRequiredGrants

Return grants needed to command execution

AxMPEG21CommandCopy: Copy target MPEG-21 Element

AxMPEG21CommandCopy
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxMPEG21CommandCopy(inout theSourceIndex : const AxIndex, inout theDestParentIndex : const AxIndex)
+getNewElementIndex() : const AxIndex &
+getRequiredGrants() : constConstAxGrantListType &

AxMPEG21CommandCopy – Class methods

AxMPEG21CommandCopy - ~AxMPEG21CommandCopy

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getNewElementIndex

Return the index of copy element

getRequiredGrants

Return grants needed to command execution

AxMPEG21CommandDelete: Delete target MPEG-21 Element

AxMPEG21CommandDelete

+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)

+AxMPEG21CommandDelete(inout toDeleteIndex : const AxIndex)

+getRequiredGrants() : constConstAxGrantListType &

AxMPEG21CommandDelete – Class methods

AxMPEG21CommandDelete

Class constructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getRequiredGrants

Return grants needed to command execution

AxMPEG21CommandEdit: Edit target MPEG-21 Element. It changes the attribute of the element with no impact on the structure.

AxMPEG21CommandEdit

+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)

+AxMPEG21CommandEdit(inout theEditIndex : const AxIndex, inout theDataElement : const MPEG21Element)

+getRequiredGrants() : constConstAxGrantListType &

AxMPEG21CommandEdit – Class methods

AxMPEG21CommandEdit

Class constructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getRequiredGrants

Return grants needed to command execution

AxMPEG21CommandExpand: Return indexes of target element's children

AxMPEG21CommandExpand

+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)

+AxMPEG21CommandExpand(inout expandIndex : const AxIndex)

+~AxMPEG21CommandExpand()

+getChildrenIndexes() : const vector<AxIndex *> &

+getRequiredGrants() : constConstAxGrantListType &

AxMPEG21CommandExpand – Class methods

AxMPEG21CommandExpand- ~ AxMPEG21CommandExpand

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getChildrenIndexes

Return indexes to children of expanded node

getRequiredGrants

Return grants needed to command execution

AxMPEG21CommandGetProtInfo: Returns protection info for target MPEG-21 element

AxMPEG21CommandGetProtInfo
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxMPEG21CommandGetProtInfo(inout index : const AxIndex)
+~AxMPEG21CommandGetProtInfo()
+getToolList() : const ConstToolListType &
+getRequiredGrants() : constConstAxGrantListType &

AxMPEG21CommandGetProtInfo – Class methods

AxMPEG21CommandGetProtInfo - ~AxMPEG21CommandGetProtInfo
--

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getToolList

Return a list of tool types used to process element's protection information
--

getRequiredGrants

Return grants needed to command execution

AxMPEG21CommandMove: Move target MPEG-21 element to destination

AxMPEG21CommandMove
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxMPEG21CommandMove(inout theSourceIndex : const AxIndex, inout theDestParentIndex : const AxIndex)
+AxMPEG21CommandMove(inout theSourceIndex : const AxIndex, inout theDestParentIndex : const AxIndex, inout theDestRefIndex : const AxIndex, in beforeafter : bool)
+getRequiredGrants() : constConstAxGrantListType &

AxMPEG21CommandMove – Class methods

AxMPEG21CommandMove - ~AxMPEG21CommandMove
--

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

getRequiredGrants

Return grants needed to command execution

AxMPEG21CommandSetProtInfo: Set protection informations for MPEG-21 target element

AxMPEG21CommandSetProtInfo
+execute(inout model : AxModelContainer, inout indexManager : AxIndexManager, inout statusManager : AxModelStatusManager)
+AxMPEG21CommandSetProtInfo(inout ttp : const AxIndex, inout tools : const ConstToolListType)
+~AxMPEG21CommandSetProtInfo()
+getRequiredGrants() : constConstAxGrantListType&

AxMPEG21CommandSetProtInfo – Class methods

AxMPEG21CommandSetProtInfo - ~AxMPEG21CommandSetProtInfo
--

Class constructor and destructor

execute

Execute the command in environment defined by input model, indexMannager, statusManager

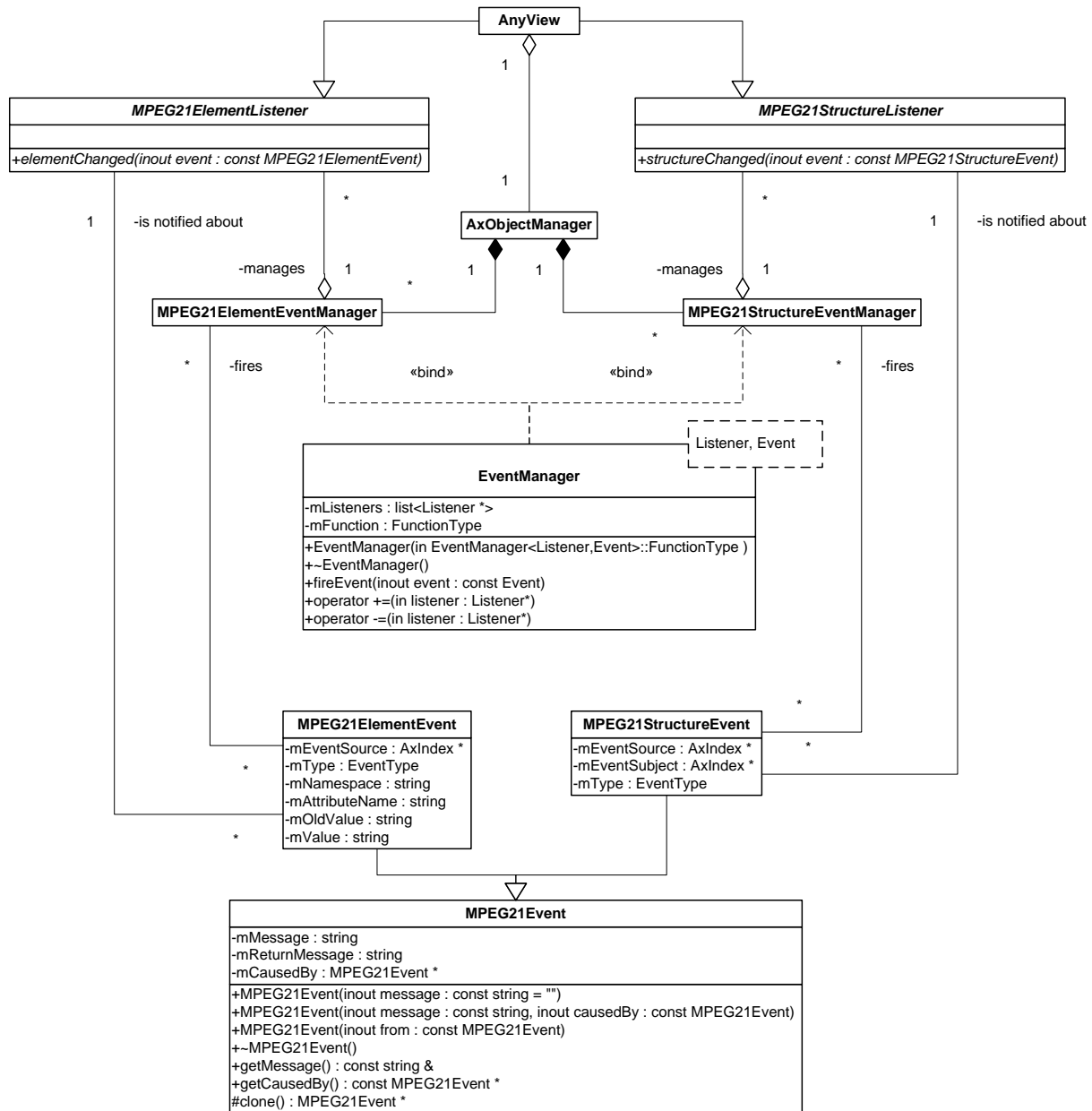
getRequiredGrants

Return grants needed to command execution

3.3 AxObjectManager as EventManager

The observer pattern has been implemented in the AxObjectManager (is not the sole case). In the following diagram the main relationships among classes which build the event-driven enabling infrastructure.

The fundamental class is EventManager, since it implements management of event listener once for all in a general manner. In fact it models in a template the common functionality of storing a list of event listeners and firing a certain event on all of them. This template can model, as depicted in the diagram, all the listener/event types.



Please note that MPEG21ElementEvent and MPEG21StructureEvent are examples of the event nature that can be model and it is effectively implemented by AxObjectManager. With this feature AxObjectManager can accept the registration of classes which are conformant to the listener for the event that can be fired.

Event are modelled by the following classes:

- **MPEG21Event**;
 - **MPEG21ElementEvent**: it models a change which has occurred in an MPEG21Element element (e.g. attributes, text);
 - **MPEG21StructureEvent**: it models a change occurred in the sub-tree starting from a certain MPEG21Element (e.g. a child has been added/removed);
- **AxObjectEvent**

- `AxObjectElementEvent`: it is similar to that defined for MPEG21, while it targets an `AxObjectElement`
- `AxObjectStructureEvent`: it is similar to that defined for MPEG21, while it targets an `AxObjectElement`

A view has to implement a specific method defined by the listener class and it has to register itself to `AxObjectManager` in order to react to specific changes of the underlying content model.

3.4 Examples of usage

Is important to specify that any operation call in `AxObjectManager` requires the class initialized. Two static methods, `initialize` and `terminate`, have to be called at the start and at the end of any chunk of code that involve use of `AxObjectManager`. Classes initialized by this methods are lower model static factories, loaders and writers.(see DE-3-1-2-2-3)

```
AxObjectManager::initialize();
...Any Code...
AxObjectManager::terminate();
```

This chunk of code shows an example of `Object Manager's` command execution.. We suppose that `initialize` is already executed.

```
AxObjectManager myAXOM;
//build a new object
AxObject *myObject=new AxObject;
myObject->setContentID("example_object_id");
AxMetadata *myMetadata = new AxMetadata();
myMetadata->setMetadataID("example_metadata");
myObject->addMetadata(myMetadata);
//build a command to add the object
AxCommandAdd *myCommand=new AxCommandAdd(*myObject,myAXOM.getRootIndex());
myAXOM.executeCommand(*myCommand);
AxObject *theSameObject =
    dynamic_cast<AxObject *> (myAXOM.getAxObjectElement(myCommand->getIndexOfAddedElement()));
delete theSameObject;
//build a new resource
AxResource *myResource=new AxResource();
myResource->setMimeType("video/mp4");
myResource->setRef("http://myvideos.com/test.mp4");
myResource->setContentID("mp4");
//build a command to add the resource before the object
AxCommandAdd* cmdAddResource=new AxCommandAdd(*myResource, myAXOM.getRootIndex());
myAXOM.executeCommand(*cmdAddResource);
AxResource *theSameResource =
    dynamic_cast<AxResource *> (myAXOM.getAxObjectElement(cmdAddResource->getIndexOfAddedElement()));
delete theSameResource;
AxObject *theObject = dynamic_cast<AxObject *> (myAXOM.getAxObjectElement(myAXOM.getRootIndex()));
AxCommandExpand *checkExpand = new AxCommandExpand(myAXOM.getRootIndex());
myAXOM.executeCommand(*checkExpand);
AxResource *againResource =
    dynamic_cast<AxResource *> (myAXOM.getAxObjectElement(* (checkExpand->getChildrenIndexes() [1])));
delete againResource;
AxCommandDelete* cmdDelete=new AxCommandDelete(cmdAddResource->getIndexOfAddedElement());
myAXOM.executeCommand(*cmdDelete);
theObject = dynamic_cast<AxObject *> (myAXOM.getAxObjectElement(myAXOM.getRootIndex()));
delete theObject;
AxCommandExpand *checkExpand2 = new AxCommandExpand(myAXOM.getRootIndex());
myAXOM.executeCommand(*checkExpand2);
theObject = dynamic_cast<AxObject *> (myAXOM.getAxObjectElement(* (checkExpand2->getChildrenIndexes() [0])));
delete theObject;
```

Please note that after obtaining an object from the `AxObjectManager` (e.g. an `AxResource`) it has to be destroyed, since it is a clone of the node (and only it) which is inside the object model.

In the following is also reported a simple example on how is possible to open a digital resource which has been embedded in an AXMEDIS object, for rendering.

```

DataSource*   loadedAsset=axom->getResourceAsset(index);
std::istream& embeddedStream = loadedAsset->getInputStream();
ResourceDecoder *decoder = new ResourceDecoder(embeddedStream,
encoding=="base64");
load(decoder->getInputStream(), mimetype);
delete decoder;
delete loadedAsset;

```

In this example the load function model the action of extracting the digital asset file and process them w.r.t. the suitable format (based on mime-type information).

3.5 Errors reported and that may occur

Error code	Description and rationales
0	Invalid Index: input index don't refers expected element
1	Invalid input resource
2	Unable to unprotect the input element

4 AXOID Assignment (DSI)

See AXMEDIS – DE – 3-1-2-2-13 section 7

5 Object Registration (DSI)

See AXMEDIS – DE – 3-1-2-2-13 section 7