Automating Production of Cross Media Content for Multi-channel Distribution

www.AXMEDIS.org

DE3.1.2.3.2
Specification of AXMEDIS Command Manager, update of DE3.1.2.2.2

Version: 3.1
Date: 30-09-2007
Responsible: DSI (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: DE3.1.2.3.2 Contractual Date of Delivery: M36 Actual Date of Delivery: 30/09/2007 Title of Deliverable: DE3.1.2.3.2 Specification of AXMEDIS Command Manager 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

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

DE number	Deliverable title	responsible
DE3.1.2.3.1	Specification of General Aspects of AXMEDIS framework AXMEDIS-DE3-1-2-3-1-Spec-of-AX-Gen-Asp-of-AXMEDIS-framework	DSI
DE3.1.2.3.2	Specification of AXMEDIS Command Manager AXMEDIS- DE3-1-2-3-2-Spec-of-AX-Cmd-Man	DSI
DE3.1.2.3.3	Specification of AXMEDIS Object Manager and Protection Processor AXMEDIS-DE3-1-2-3-3-Spec-of-AXOM-and-ProtProc	DSI
DE3.1.2.3.4	Specification of AXMEDIS Editors and Viewers AXMEDIS-DE3-1-2-3-4-Spec-of-AX-Editors-and-Viewers	DSI
DE3.1.2.3.5	Specification of External AXMEDIS Editors/Viewers and Players AXMEDIS-DE3-1-2-3-5-Spec-of-External-Editors-Viewers-Players	DSI
DE3.1.2.3.6	Specification of AXMEDIS Content Processing AXMEDIS-DE3-1-2-3-6-Spec-of-AX-Content-Processing	DSI
DE3.1.2.3.7	Specification of AXMEDIS External Processing Algorithms AXMEDIS-DE3-1-2-3-7-Spec-of-AX-External-Processing-Algorithms	FHGIGD
DE3.1.2.3.8	Specification of AXMEDIS CMS Crawling Capabilities AXMEDIS-DE3-1-2-3-8-Spec-of-AX-CMS-Crawling-Capab	DSI
DE3.1.2.3.9	Specification of AXMEDIS database and query support AXMEDIS-DE3-1-2-3-9-Spec-of-AX-database-and-query-support	EXITECH
DE3.1.2.3.10	Specification of AXMEDIS P2P tools, AXEPTool and AXMEDIS AXMEDIS-DE3-1-2-3-10-Spec-of-AXEPTool-and-AXMEDIA-tools	DSI
DE3.1.2.3.11	Specification of AXMEDIS Programme and Publication tools AXMEDIS-DE3-1-2-3-11-Spec-of-AX-Progr-and-Pub-tool	UNIVLEEDS
DE3.1.2.3.12	Specification of AXMEDIS Workflow Tools AXMEDIS-DE3-1-2-3-12-Spec-of-AX-Workflow-Tools	UR
DE3.1.2.3.13	Specification of AXMEDIS Certifier and Supervisor and networks of AXCS AXMEDIS-DE3-1-2-3-13-Spec-of-AXCS-and-networks	DSI
DE3.1.2.3.14	Specification of AXMEDIS Protection Support AXMEDIS-DE3-1-2-3-14-Spec-of-AX-Protection-Support	UPC
DE3.1.2.3.15	Specification of AXMEDIS accounting and reporting AXMEDIS-DE3-1-2-3-15-Spec-of-AX-Accounting-and-Reporting	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.

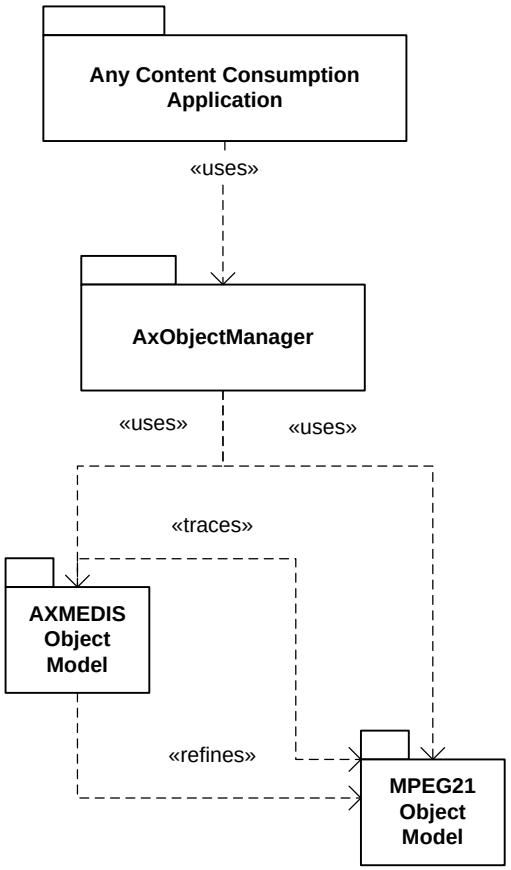
1.2 List of Modules or Executable Tools Specified in this document

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 retrieval 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 and implemented as a part of AxObjectManager package, in order to allow upper layer applications to effectively manage their rendering of the multimedia package Events managed in AxObjectManager provides information about changes in the package structure and notifies modifications in the embedded digital resources or the related metadata. Both AXMEDIS and MPEG-21 models may be subject of those changes , thus two distinct sets of 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

The AXMEDIS Object Manager (AxObjectManager from now on) has to be used to develop software tools which will handle contents following DRM rules. This module has three main tasks :

- To Provide API to upper layer tools in order to allow AXMEDIS and MPEG-21 DIs object manipulation, also maintaining indexes for the object
- To Assure DRM compliance on any object model related operation, even avoiding direct control from upper layer tools to AXMEDIS and MPEG-21 DIs objects.
- To Establish an Event-Driven communication between Model Layer and Application Layer to cope whit objects structure and content changes.

AXMEDIS Object Model (AXOM from now on) is used to cope with content objects and it is compliant with MPEG-21, extending it with additional features and data structures.
AXOM can be used as foundation to design and develop applications which are able of manipulating both MPEG-21 DIs and AXMEDIS Objects, editing/authoring and playing contents and supporting manipulation/access of a hinged package structure (containing nested levels of packaging, hierarchies organization).
On the other hand one of the most important features of this model is to provide the means to create a trusted environment,, capable,to enforce DRM .
AXOM is capable to guarantee that a tool built on its accessible commands will provide digital right observance Commands are designed to allow only execution of license permitted actions..; different actions on the content model should require different grants (i.e., authorizations). Actions can target the content structure, the resources and the metadata; a unique flow to handle verification of any action performed on the

content has been conceived. Finally a single command may require multiple authorisation to be executed (almost one).

AxObjectManager is the interface among AXOM and upper application layer. Object Manager will provide all base operations (add, change, delete, etc…) which will be needed to manipulate AXMEDIS and MPEG-21 DIs objects,. Moreover, it will maintain the trustness of the object model avoiding direct manipulation of its components from application.

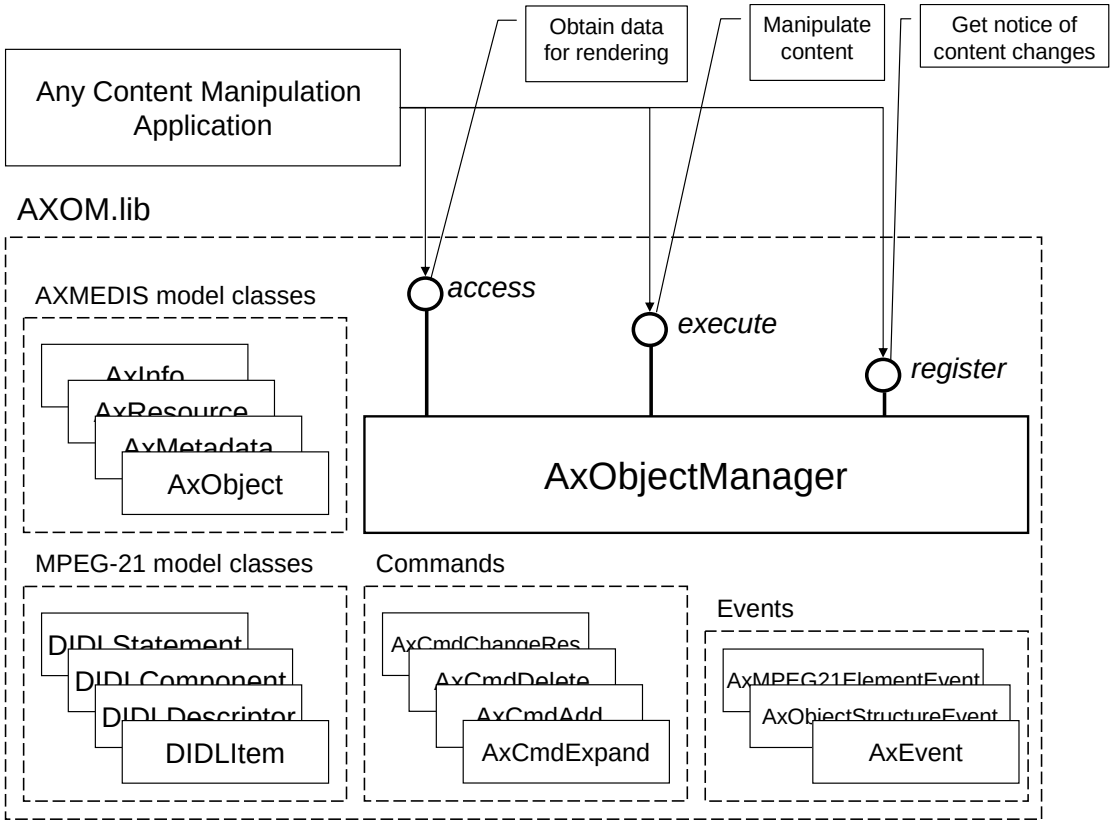
As stated, the AXOM can be used to build a large set of content manipulation applications/systems obtaining trusted behavior encapsulated in a clear use of AXMEDIS/MPEG-21 Objects. The developer can use the API exposed by the AXOM for object loading, browsing and navigation in the object structure, obtaining streams for embedded digital resources according to the rights defined in the licenses. Every action performed on the content is related to a set of rights, which have to be owned at consumption time to enable the action. The result is a transparent DRM-including manipulation of the underlying content package.
The API is organized in two main subsets:

- Object Model Access – functionalities for browsing and reading content elements; i.e. getting metadata, components, reading content element attributes such as resource descriptors, technical information: MIME-type;
- Object Model Manipulation – functionalities for modifying content package according to DRM rules by using a set of executable commands to add, remove, copy, move content elements such as metadata or resources, to change content elements attribute.

The Object Model Access API covers the basic functionalities to explore and retrieve information from the content package; some of them are listed in the following:

- *getAxObjectElement* – to obtain information about a content element, it provides a clone of the element included in the package;
- *getAxChildIndexes* – to get the list of indexes of the package component when applied to the root level or to an inner sub-package;
- *getAxMetadataIndexes* – to get the list of indexes of the package descriptors;
- *getResourceAsset* – to realize the digital resource extraction, providing a byte stream that can be used for different purposes such as rendering, printing, content processing. This method can used for different purposes: call arguments allow to impose the action to be performed on the resource in terms of rights and related details — e.g., play for 15 minutes;
- *getPublicMetadataTree* – to obtain the complete tree of “public” metadata of the underlying content – i.e. obtaining a full description of it without the need of unprotecting.

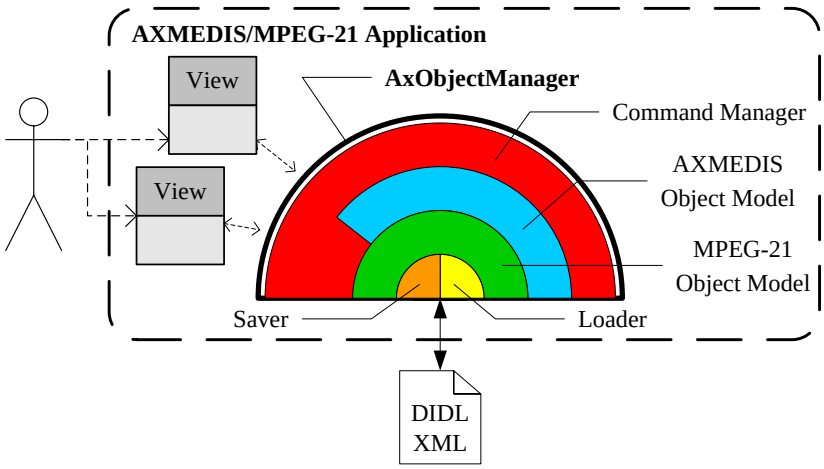
The Object Model Manipulation API has been kept minimal; while an extensible set of elementary command classes have been provided. Thus the content manipulation looks like a sequence of commands targeting the AXMEDIS Object.



- 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.

In order to build a reusable and flexible infrastructure, the AXMEDIS Object Manager has been decomposed in five parts, separating responsibilities of the several elements: MPEG-21 Object Model responsible of managing the MPEG-21 DI allowing content access and manipulation; AXMEDIS Object Model a refinement and an extension of MPEG21 DI, targeting the mentioned requirements and realizing on top of the MPEG-21, the concept of AXMEDIS Data Model.



The part which is described with full details in this document is Command Manager, it is places at the top of the layers and allows to access and manipulate high-level features in the underlying MPEG-21 structure;
The Command Manager allows transparent manipulation of content in respect to DRM rule and directly accessing to the Servers for the acquisition of the License and of Protection Information. It allows to access or to manipulate content with a built-in authorization check on the basis of MPEG-21 REL.

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
- **FTP download**: *ftp://* is used when AXMEDIS object have to be retrieved by using File Tranfer Protocol.
- **AXDB checkout**: a special database protocol as been defined with corresponding 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.

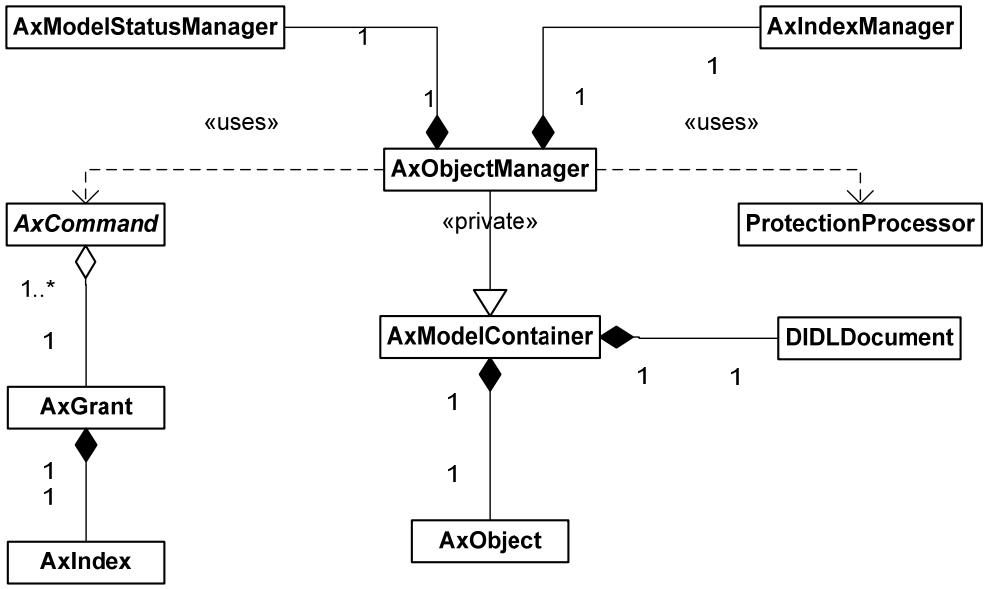
For any of these protocol option it can be chosen which loading mode has to be exploited: two file formats are supported:

- Simple XML document
- MPEG-21 FileFormat (ISOMedia compliant)

If nothing is specified, the loader automatically detect the file type by the prologue and select the appropriate loading mode.

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 MPEG-21 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



The AXMEDIS Object Manager architecture supports extendibility, new types of objects based on MPEG-21 model can be easily managed in MPEG-21 Object Model without changes in present structure. Moreover, only minor changes are needed to process content models that extend MPEG-21 or AXMEDIS elements definitions. Hooks to perform operations on content packages are provided by the upper level layer by means of AxObjectManager and AxCommand set.
These classes offer:

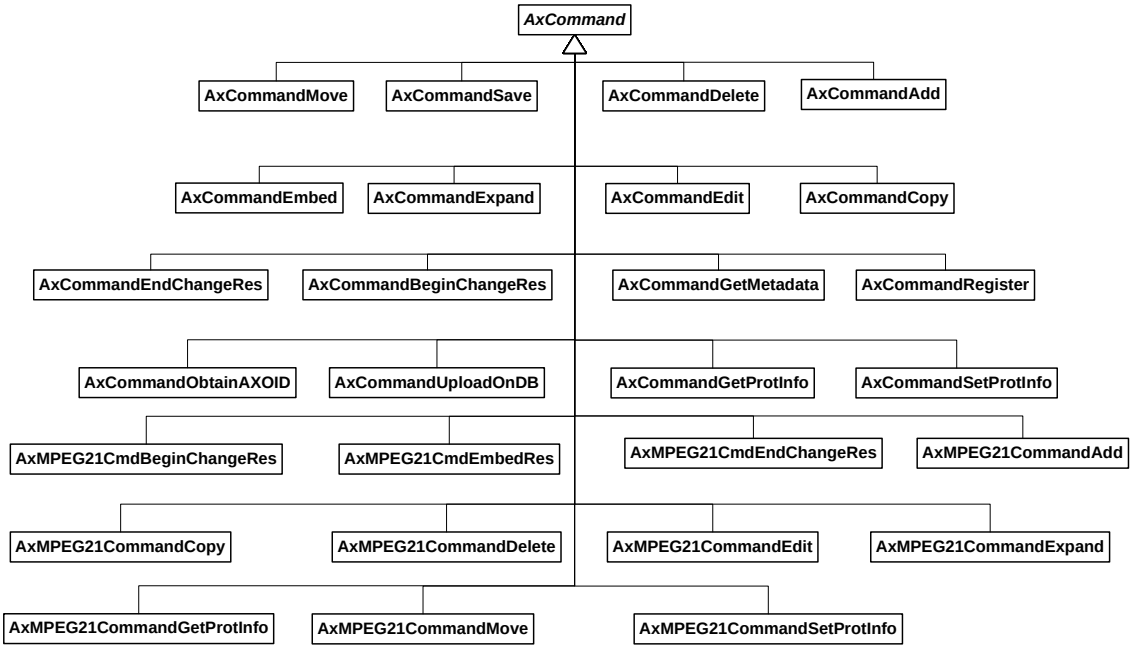
- a range of executable commands targeting the manipulation of both AXMEDIS and MPEG-21 object models,
- coordination of functionalities of lower levels modules to organize a reliable and protected execution flow.

The set of commands is easily extendable to manage future needs of upper level applications as shown in the sequel. Execution of defined commands is performed using class AxObjectManager as shown in the sequel

The “Command Pattern” has been applied to the design of AxObjectManager. The controller is the AxObjectManager which exposes the AXOM API. AxObjectManager is an intermediate layer between the application views and the object models (i.e., the object/package model with resources and descriptors). A view cannot directly manipulate the object model, while it has to issue commands to the AxObjectManager class. The latter is in charge of performing the requested actions on the model. Specifically, each conceivable command has been realized as a class which implements the interface AxCommand. It exposes two main methods: execute and getRequiredGrants. The execute method of AxCommand has to be implemented by each specialized class to actually perform the action on the object model. The command execution method is able to directly access to the data model without any restriction, since the command is executed only if the authorization is obtained. The getRequiredGrants method of AxCommand allows the verification of the requested grants. Thus, the AxObjectManager class is able to handle the request received by the user and the set of conceivable commands can be augmented without any impact on the current architecture.

AxObjectManager class has few control points and the code to verify the commands can be easily inserted in. Indeed AxObjectManager class is not in charge of verifying the grants, while it has been designed in order to provide hooks to easily create control mechanisms. The DRM enforcement is modeled by the declaration of the requested rights which has to be stated by each AxCommand inherited class. In fact, to each command is associated a list of AxGrant, which are the basic arguments for issuing the request to the authorization service. In this way, AxObjectManager is able to generically handle any request issued by the user, respecting governance, delegating authorization service the task of determining if the user has been authorized or not on the basis of the license.

Going deeply we can see how AxCommand class is used. From this class derives all allowed commands that could be requested to 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 represented command task . Other accessory methods could be implemented in the derived command classes.

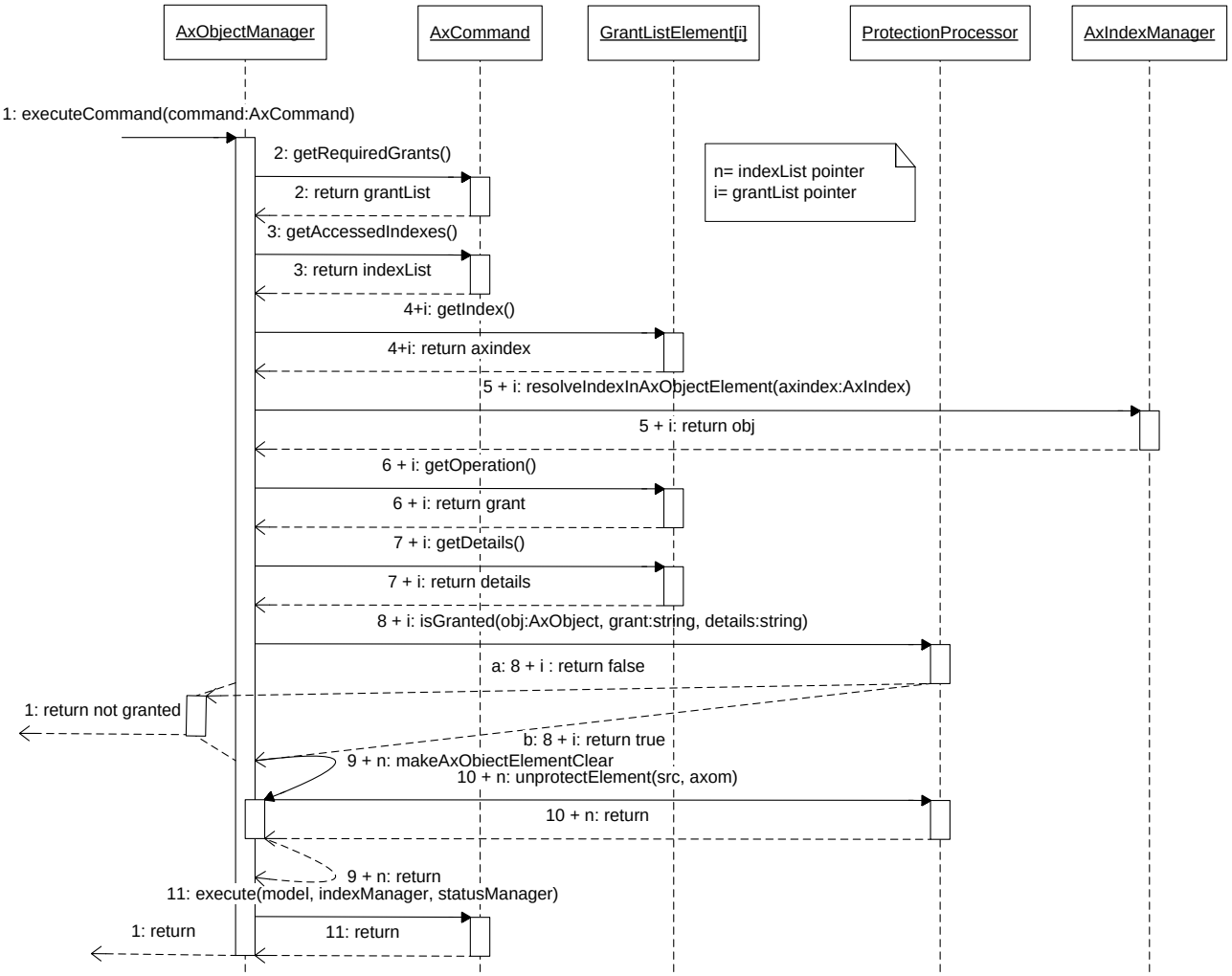
3.2.1 AxObjectManager Capabilities Overview

The AxObjectManager exposes functionalities for: creating new content; opening existing content by indicating a URI; browsing content structure; accessing metadata and resources embedded or referred to by the content; manipulating content structure, metadata and resources; saving the content; adding/modifying protection information associated to any content element.

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.



By using model encapsulation, a good level of security has been achieved in terms of robustness against developer's malicious content handling preventing direct manipulation from the views. Thus, views are not allowed to target content elements by using pointers. Command targets have to be indicated using logical references that prevent the access to the physical addresses referring to the digital resources. Thus, the view interacts with the model using instances of the class *AxIndex*. These are used like pointers, while they can be actually resolved only by the *AxObjectManager* which generated them.

The Command Pattern allows to easily extend the set of actions which AXOM is capable to execute (and control)
The proposed solution has been designed to speed-up the creation of AXMEDIS-compliant tools, which can easily exploit the provided functionalities and can also extend the command set to their specific need. A custom new command can be defined by taking into account its fundamental aspects: authorization, un-protection and behavior implementation. By specializing from AxCommand class the custom command class presents *getRequiredGrants*, *getAccessedIndexes* and *execute* that have to be implemented according to the desired semantics and manipulation logic. The custom action commands classes have to define one or more constructors to give arguments to the manipulation (e.g., target elements) and to introduce specific methods in order to obtain back information after a performed execution.

An example of an extension set of manipulation commands could be one for resource processing.
Let us consider a command for processing an image performing a mirror transformation (left to right) and replace the resource with the result of the processing.
Command *AxCommandImageMirror* defines a constructor which accepts the target image as an argument.
The constructor can be defined as *AxCommandImageMirror(AxIndex imageIndex)*.

The following pseudo-code provides and example about the definition of the command, including the declaration of the required rights and what has to be unprotected.
The method *execute*() sketching the command implementation is also reported.

Each proposed custom command has to be certified and approved to be compliant with the semantics of the rights. This means that the enforcement of the rights into the authoring and player tools has to be performed in according to a rights data dictionary – e.g., MPEG-21 RDD and that defined by Mi3P.

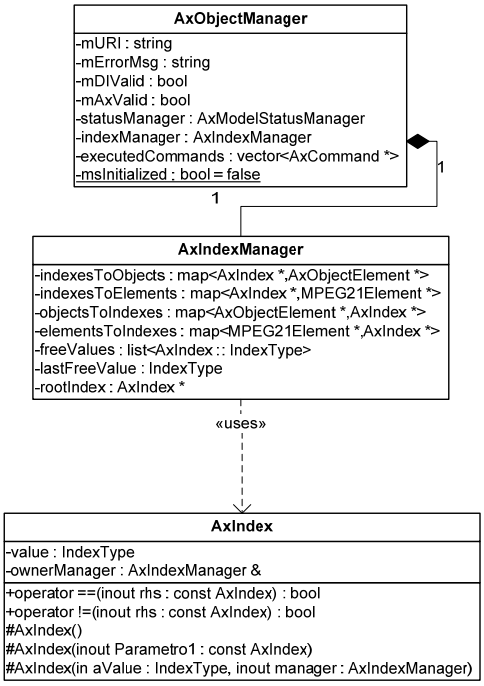
```
class AxCommandImageMirror : public AxCommand
{  private AxIndex targetIndex;
   //constructor
   AxCommandImageMirror(AxIndex imageIndex)
   {     targetIndex = imageIndex;
   }
   //declaration of required granted rights the operation “modify” has to
   // be authorized with details “apply mirror filter”
   vector AxGrant getRequiredGrants()
   {     return new vector { new AxGrant(targetIndex, “modify”,
                              “apply mirror filter”) };
   }
   //declaration of content elements to be unprotected
   //only the target resource has to be unprotected
   vector AxIndex getAccessedIndexes()
   {     return new vector { targetIndex };
   }
   void execute(AxModelContainer model,
                  AxIndexManager indexManager, AxStatusManager s)
   {     //since the model has been already unprotected it is possible to
         //have direct access to the target resource (located by targetIndex)
         AxResource res = (AxResource) indexManager.resolveIndex(targetIndex);
         string mime res.getMIMEType();
         //processing
         inputstream bytes, res.getAsset().getInputStream();
         inputstream mirrorbytes = applyMirrorFilter(mime, bytes);
         res.getAsset().setInputStream(mirrorbytes);
   }
};
```

Please note that *targetIndex* is used to store the location of the target resource, which is set only at construction time. This index is returned as the unique “accessed index” of the grant authorization. The index is used at command execution time to retrieve the resource in the object model.

3.2.2 AxObjectManager as IndexManager

One of the main tasks AxObjectManager is responsible is to give support to upper layer application for a direct, in terms of indirection , access to the element managed by AXOM. To maintain separation of the functionalities the class AxIndexManager has been designed to wrap indexing functionalities then any AxObjectManager includes an instance of this class.

AxIndexManager maintains two separated indexes tables: one that refers to MPEG-21 hyerarchy and the other refers to AxObjects hierarchy. Another class designed to implement indexing functionalities is AxIndex, representing instances of a single index and used in AxIndexManager as key for indexes tables.



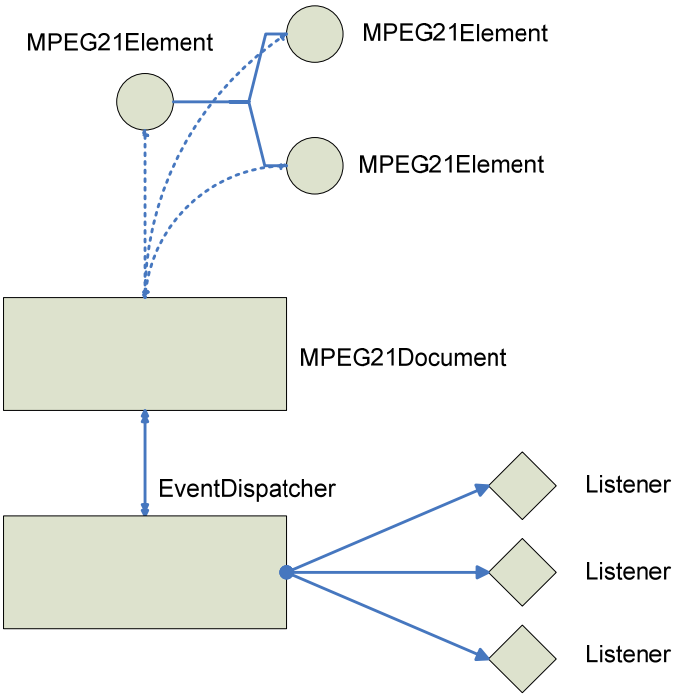
As shown in the class diagram AxIndexManager declares four private maps structure among its own structure. These maps establish legacy among objects and indexes and indexes and objects. Direct access to maps structures is hidden by the implementation through AxObjectManager and AxIndexManager avoiding direct access to referred objects and elements.

3.2.3 AxObjectManager as EventManager

The AxObjectManager, managing the life-cycle of the AXMEDIS Object manipulation need to efficiently handle the result of the modifications. The view that are attached to this manipulation controller has to render the changes after a proper notification of what it has been changed in the underlying model.

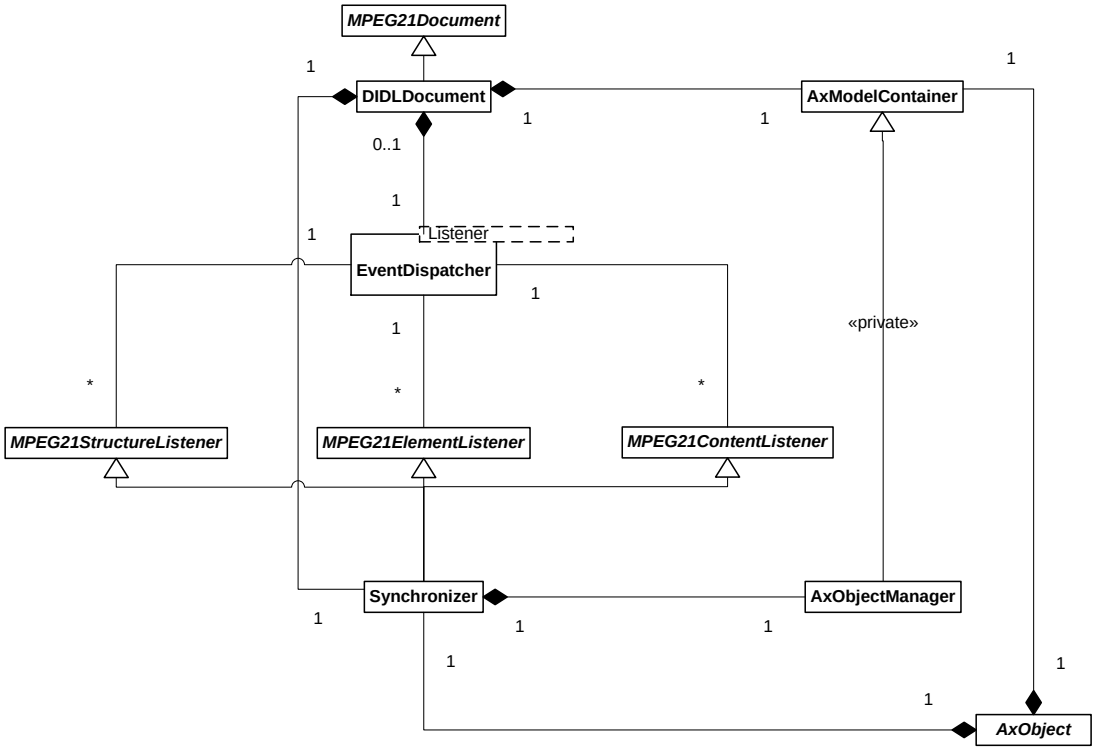
The observer pattern has been implemented in the AxObjectManager (is not the sole case). In the following diagram the main relationships among entities which build the event-driven enabling infrastructure:

- The MPEG- 21 Element that is where te modification could take place (e.g. a child is added)
- The MPEG-21 Document that is responsible of propagating any change notification to the proper handler
- The Event Dispatcher that after the notification from the Document can forward to all the registere listeners the received information.

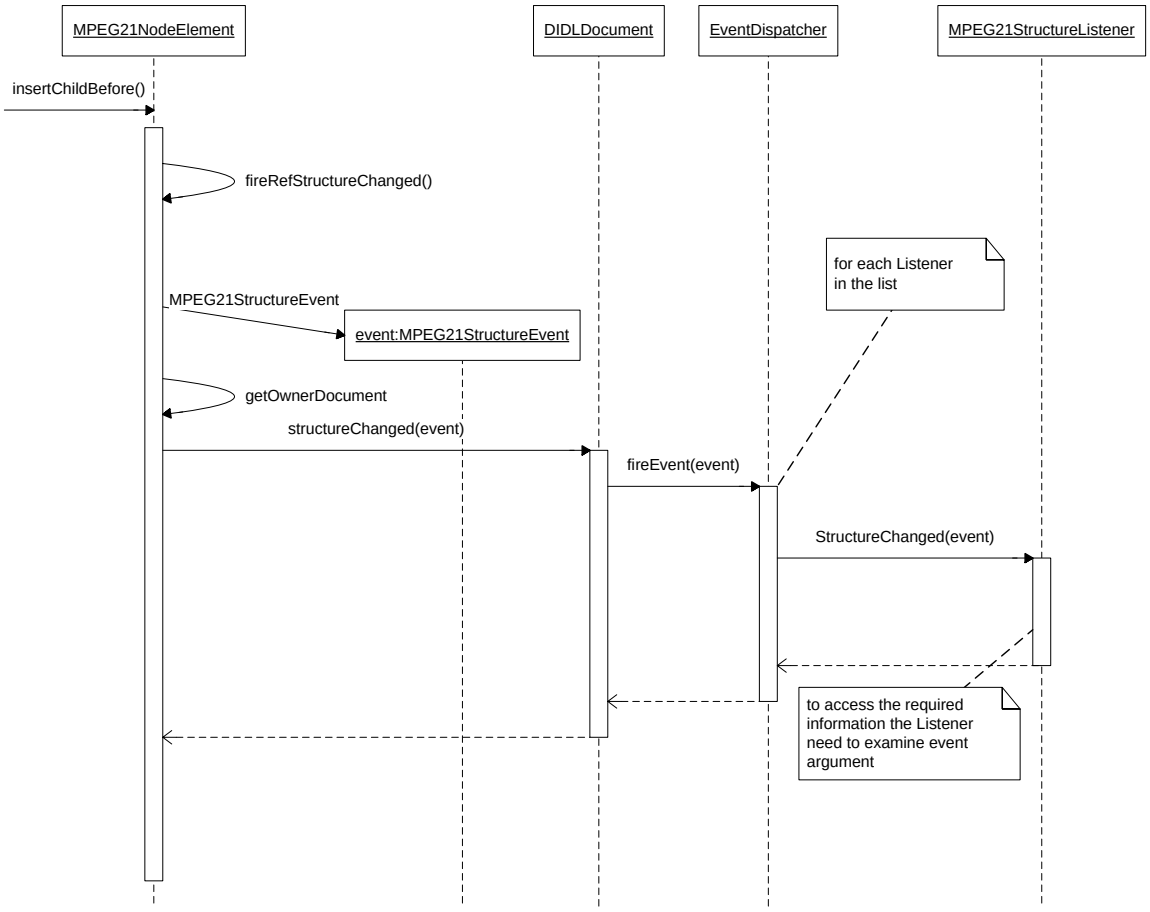


In the above diagram the design of the Event dispatcher is outlined. The relation with the DIDLDocument allow document changes being notified to the listeners.

The fundamental entity is EventDispatcher, 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.



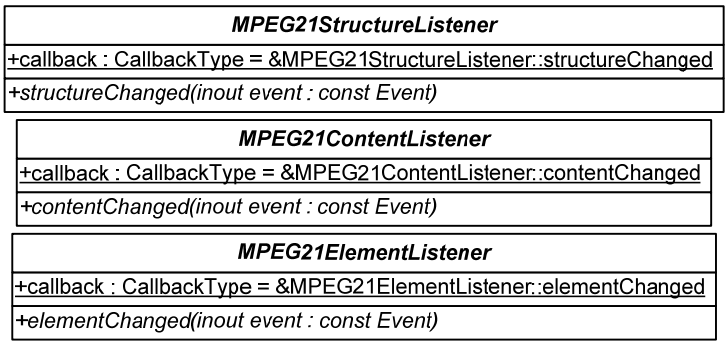
The following diagram depicts the mechanism that notifies any listener of an event, in this case the insertion of a child in an MPEG21Element.



The architecture is extensible, while three kinds of listener have been defined in order to handle typical events describing changes inside and MPEG-21 Document:

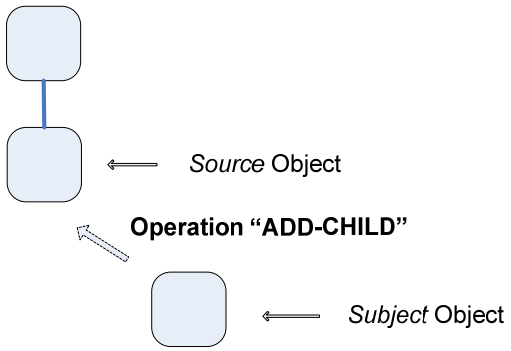
- MPEG21StructureListener that is able to receive all the nofications regarding changes in the hierarchical structure of MPEG-21 Document (e.g. a child has been added, another is removed).
- MPEG21ElementListener that is able to receive notification regarding the changes of the elements' attributes (e.g. id attribute has been changed).
- MPEG21ContentListener that is responsible of handling the modification regarding the digital assets or the XML content that are stored inside some MPEG-21 element (e.g. DIDLStatements, DIDLResource).

In the following the three listener classes are reported. Their responsibility is also to define the specific Event class and the specific CallBackType. With this two inner-classes can enforce how listener implementation can obtain information regarind the managed event.

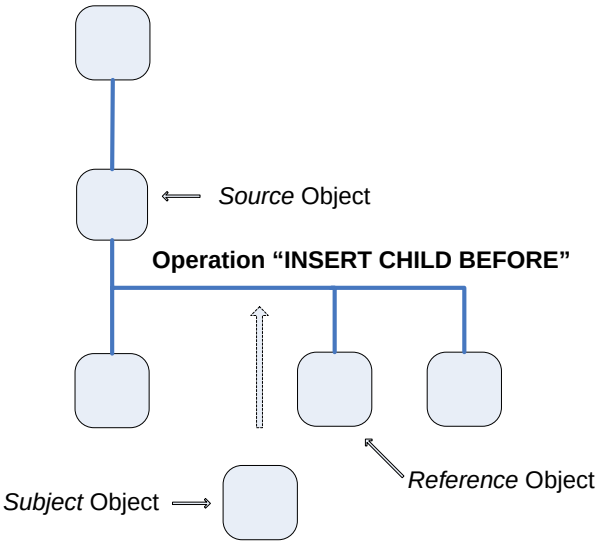


The MPEG21StructureEvent is the more complex event class since the structure of an MPEG-21 Document can change in different manners. In the following diagrams some of the structure events are listed.

When adding a new element inside an MPEG-21 Document, the event object carries the information of the element in which something has been added, and the element that is added. The first is indicated by "Source" and the second by Subject.

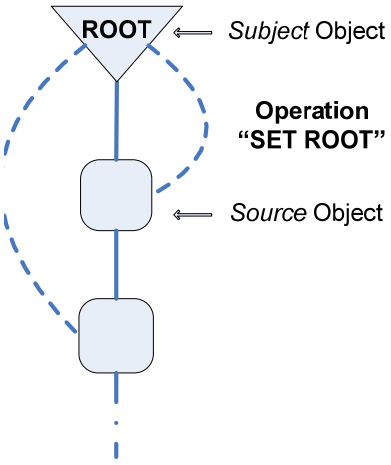


The addition can be performed not only to the end of an existing children list or to an empty list. In this case three object information is carried by the event: the Source and the Subject, like in the simple addition, and the Reference, that represent the position reference in order to retrieve "where" in the list the new element has been added.

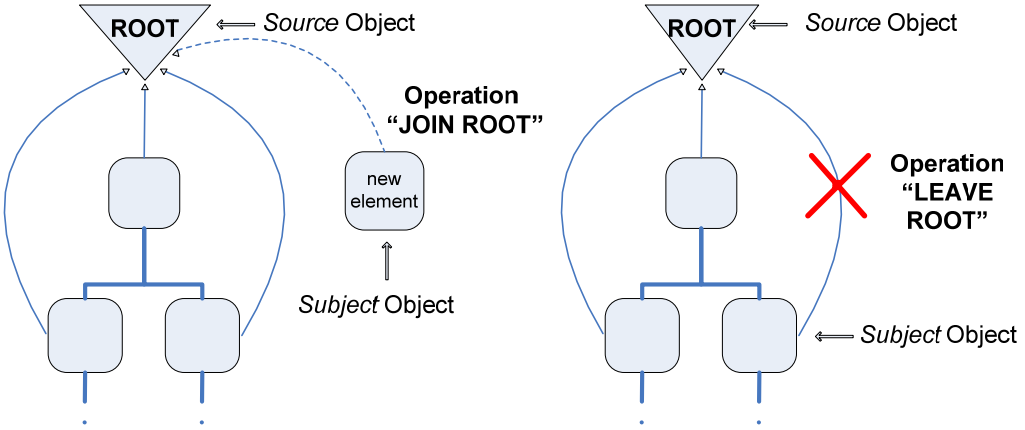


Some important changes in the structure of the MPEG-21 Document are related to the root element of the document. The MPEG21StructureEvent in this case define three possible condition.

– SET ROOT that means that a new root element has be set to the Document.



– JOIN ROOT and LEAVE ROOT that are respectively the action of adding a new element to the Document or to remove it.



The presented mechanism is simply the basis of the event dispatching inside the Model of the AxObjectManager. New entities are needed in order to notify MPEG-21 event to the outside views, but more important is to adjust the AXMEDIS model after the MPEG-21 modification, and to notify also the AXMEDIS Data Model events to the views.

The Synchronizer has bene created to keep AXMEDIS Model and MPEG-21 Model in a consisten relation, since AXMEDIS is a simplified view of the MPEG-21.

At any MPEG-21 Document change the AXMEDIS object model can be invalidate because a forbidden MPEG-21 Element has been inserted or some element that are mandatory for AXMEDIS Application Domain are removed.

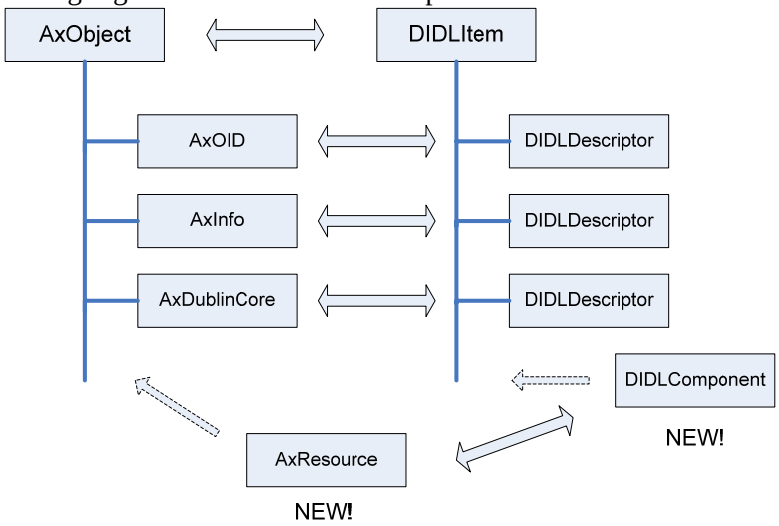
The Synchronizer acts as a listener for every kind of MPEG-21 Document event and contains the adjustment and validation logic.

In the following a table is reported to examine all the situation that are generated by MPEG21StructureEvent

CHILD(sbj)	PARENT(src)	EVENT TYPE	Required Sync. Operations
DIDLDescriptor	DIDLItem	CHILD_ADDED	New metadata has been attached to an AXMEDIS Object: • to verify the AxMetadata it is not already present • to load the DIDLDescriptor as an AxMetadata • to verify if the AxMetadata is not considered mandatory • to find the AxObject that is associated to the parent DIDLItem • to invoke `syncAddMetadata(new metadata)`
DIDLItem	DIDLItem	CHILD_ADDED	New object has been attached to the AXMEDIS Object as a sub-component: • to verify the AxObject it is not already present • to load the DIDLItem as an AxObject • to find the AxObject that is associated to the parent DIDLItem • to invoke `syncAddContent(new content)`
DIDLComponent	DIDLItem	CHILD_ADDED	New media resource has been attached to the AXMEDIS Object": • to verify the AxResource it is not already present • to load the DIDLComponent as an AxResource • to find the AxObject that is associated to the parent DIDLItem • invoke `syncAddContent(new content)`
DIDLStatement	DIDLDescriptor	CHILD_ADDED	No action
DIDLDescriptor	DIDLDescriptor	CHILD_ADDED	New descriptor has been added as attribute of an AxMetadata: • to verify Statement proper NameSpace and LocalName (if not AxObect is invalidated) • To find the AxMetadata that is associated to the parent DIDLDescriptor • invoke `refreshMetadataDescriptors()`
DIDLResource	DIDLComponent	CHILD_ADDED	New resource has been added to a DIDLComponent: the AXMEDIS model has to be invalidated
Other DIDL element	Other DIDL element	CHILD_ADDED	All cases that are not included in the above list cause the AXMEDIS model invalidation

DIDLDescriptor	DIDLItem	CHILD_REMOVED	A metadata has been removed from an AXMEDIS Object: • to verify the related AxMetadata it is not already removed • To find the AxMetadata that is associated to the DIDLDescriptor • to verify if the AxMetadata is not considered mandatory • to invoke `syncRemoveMetadata(metadata)`
DIDLItem	DIDLItem	CHILD_REMOVED	Object has been removed from a composite AXMEDIS object: • to verify the related AxObject it is not already removed • To find the AxObject that is associated to the DIDLItem • to invoke `syncRemoveContent(content)`
DIDLComponent	DIDLItem	CHILD_REMOVED	A resource has been cancelled • AxObject has to be invalidated
DIDLStatement	DIDLDescriptor	CHILD_REMOVED	The DIDL Statement has been removed by the Descriptor: to be invalidated
DIDLDescriptor	DIDLDescriptor	CHILD_REMOVED	Deleting a Descriptor in a Descriptor: • To find the AxMetadata that is associated to the DIDLDescriptor • invoke `refreshMetadataDescriptors()`
DIDLResource	DIDLComponent	CHILD_REMOVED	Resource is remove inside a DIDLComponent
Other DIDL element	Other DIDL element	CHILD_REMOVED	All cases that are not included in the above list cause the AXMEDIS model invalidation

In the following diagram the situation in which a new DIDLComponent is inserted as a child of a DIDLItem is depicted. The diagram highlights the sinchronization aspect of the related AXMEDIS object model.

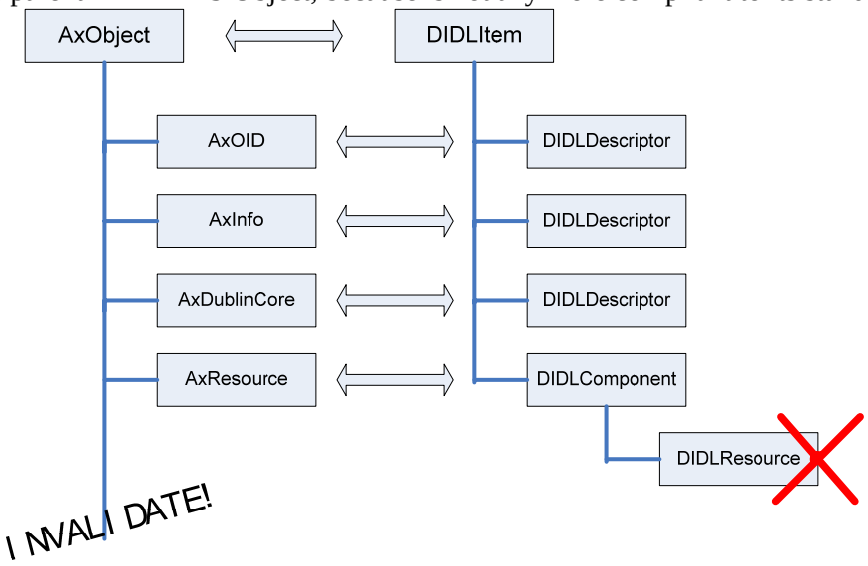


In the following a table is reported to examine the required operation when the Synchronizer is notified with a MPEG21ContentEvent.

Element	Event Type	Required Sync. Operations
DIDLStatement	CONTENT_CHANGED	

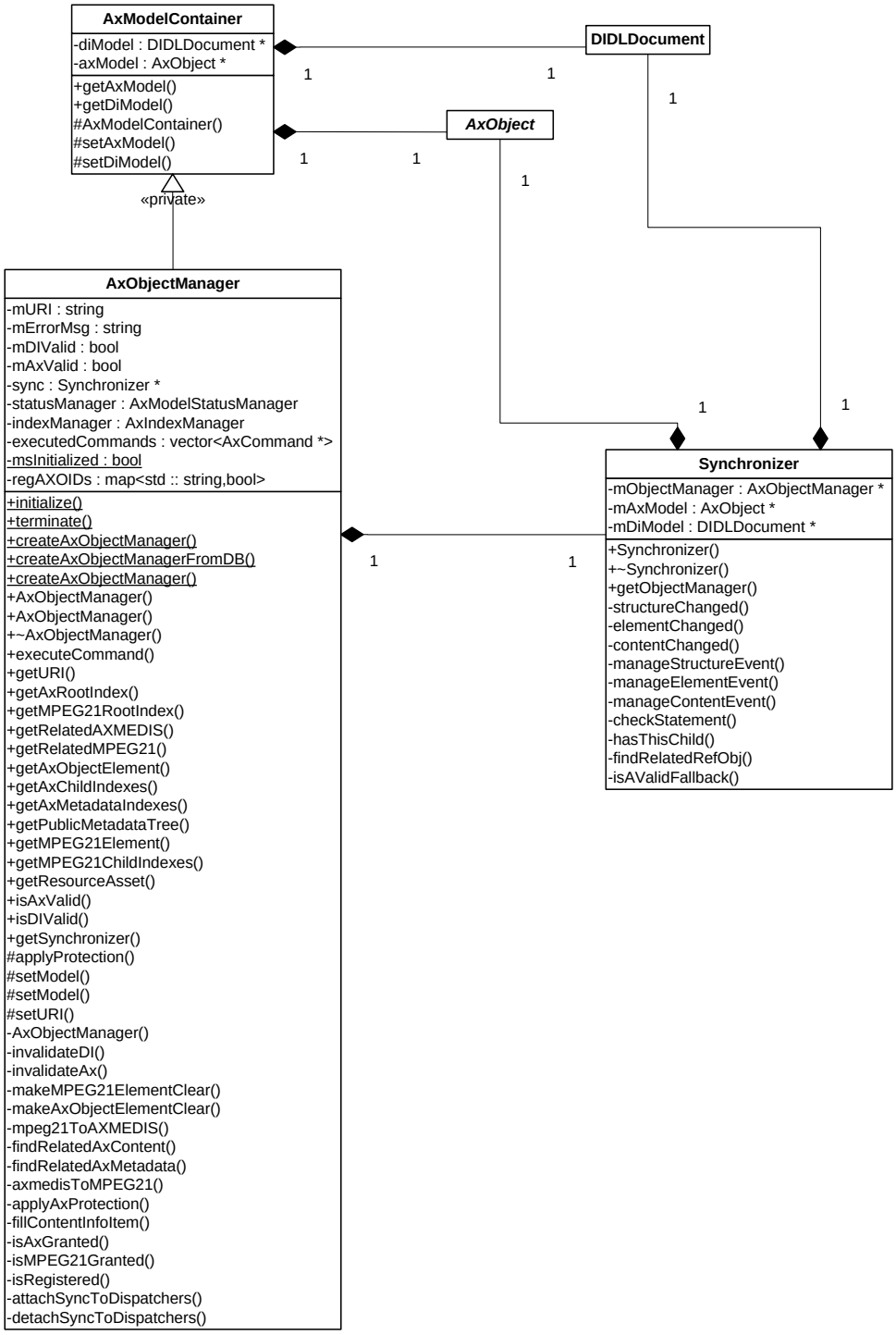
		The content has been changed: • to verify the parent element is a Descriptor • to find the AxMetadata that is associated to the DIDLDescriptor • to check if the modified metadata is mandatory and some required information have been removed.
DIDLResource	CONTENT_CHANGED	no action
Xinclude	CONTENT_CHANGED	The fallback is changed • to verify the new XInclude contains the reference to an AXMEDIS Object
Other DIDL element	CONTENT_CHANGED	All cases that are not included in the above list cause the AXMEDIS model invalidation
DIDLStatement	CONTENT_CLEARED	The metadata content has been deleted • to verify the parent element is a Descriptor • to find the AxMetadata that is associated to the DIDLDescriptor • to check if the modified metadata is mandatory and some required information have been removed.
DIDLResource	CONTENT_CLEARED	A resource has been emptied. AxObject has to be invalidated.
Xinclude	CONTENT_CLEARED	The Xinclude elementi is emptied thus, the AxObject has to be invalidated
Other DIDL element	CONTENT_CLEARED	All cases that are not included in the above list cause the AXMEDIS model invalidation

The following diagram depicts a situation in which the deletion of content inside a DIDLResource causes the invalidation of the parent AXMEDIS Object, because is not any more compliant to its standardization.



From the point of view of the AxObjectManager:

- it owns to data model MPEG21Document and the root AxObject, through the AxModelContainer class.
- it refers to a Synchronizer object in order to keep this two model in synchronization.
- it is responsible of providing means to allow client ciews registering to the interesting events related to the MPEG-21 and the AXMEDIS object models.



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.

Please note that the signature of the listener methods will change the Event object definition, in order to communicate not pointer any more, but AxIndex objects.

So AxEvent and AxMPEG21Event will be the classes of the objects that are received by the listener that are outside the controller (AxObjectManager).

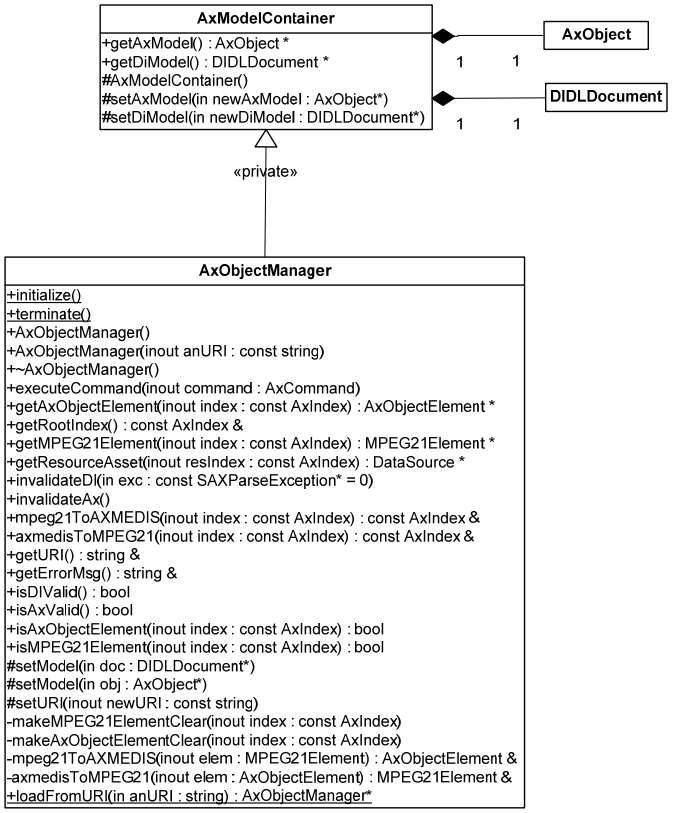
In the following is reported the classes' declarations.

```
class AxEvent
{
public:
      typedef enum {
                        INVALIDATE,
                        METADATA_ADDED,
                        OBJECT_ADDED,
                        RESOURCE_ADDED,
                        METHOD_ADDED,
                        RESOURCE_CHANGED,
                        OBJECT_CHANGED,
                        METHOD_CHANGED,
                        OBJECT_REMOVED,
                        METADATA_REMOVED,
                        RESOURCE_REMOVED,
                        METHOD_REMOVED,
                        METADATA_MODIFIED,
                        CONTENT_ATTRIBUTES_CHANGED,
                        METADATA_ATTRIBUTES_CHANGED,
                        METHOD_ATTRIBUTES_CHANGED,
                        RESTORE}
                        EventType;

      AxEvent(const AxIndex& parent, EventType Type,int position=0);
      ~AxEvent();

      inline const AxIndex&  getElementIndex()const;
      inline int getPosition() const;

      inline EventType getType()const;
};

class AxMPEG21Event
{

public:

      typedef enum {
            CONTENT_CHANGED,
            ELEMENT_ADDED,
            ELEMENT_REMOVED,
            ATTRIBUTES_CHANGED}
            EventType;

      AxMPEG21Event(const AxIndex* parenIndex,const AxIndex* target,const
AxIndex* reference, EventType Type);
      ~AxMPEG21Event();

      inline const AxIndex* getParentIndex()const;
      inline const AxIndex* getTargetElement()const;
      inline const AxIndex* getReferenceElement()const;

      inline EventType getType()const;
};
```

3.2.4 Class Methods Overview

AxObjectManager - AxModelContainer

Core class of this module, expose methods to interface outer applications to inner classes of the module. As exposed in previous parts of this specification through AxObjectManager ,AxIndexManager, AxCommand and AxModelStatusManager are accessed.



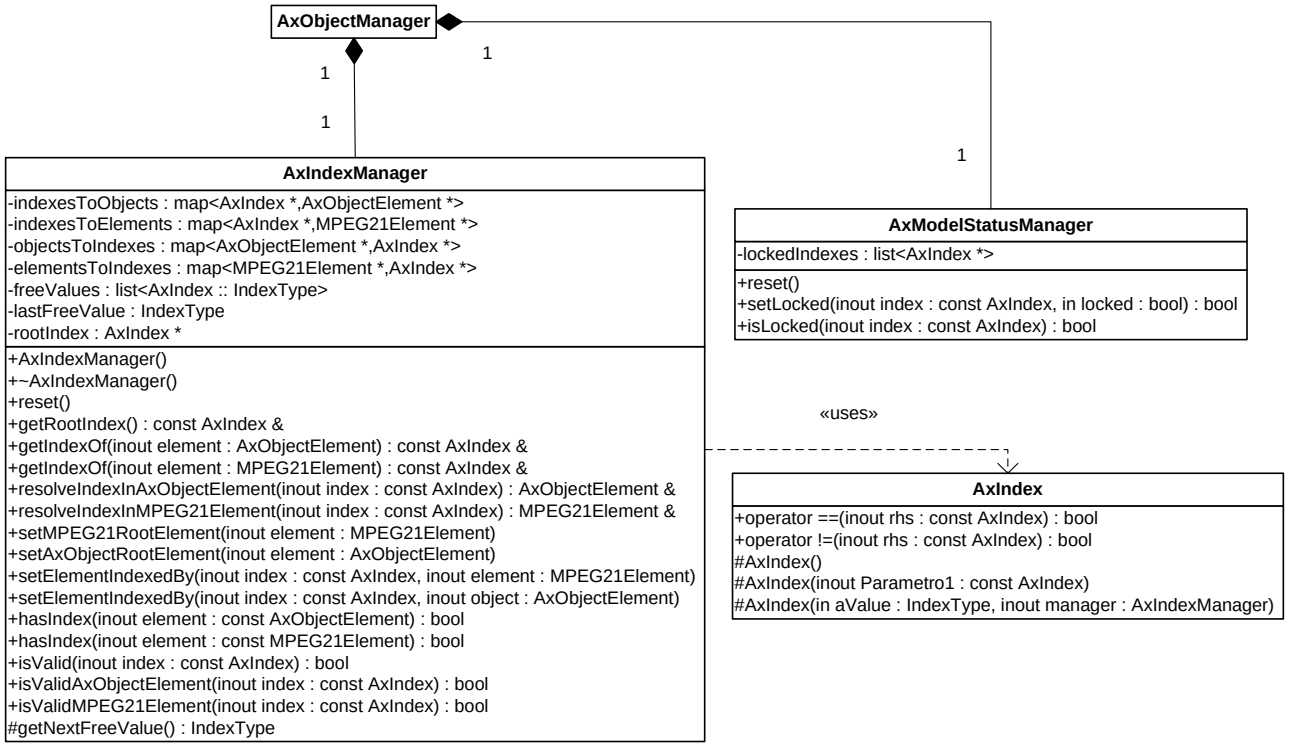
AxObjectManager – Class methods
AxObjectManager -AxObjectManager(const string anURI) - ~AxObjectManager
Class constructor and destructor. AxObjectManager constructors acts in a two way behaviour. If a class user wants to create a new manager for a target existing AXMEDIS or DIDL object , an input URI string has to be specified in order to retrieve it. Elsewhere a new empty clear object is created along with AxObjectManager new instance.
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
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.



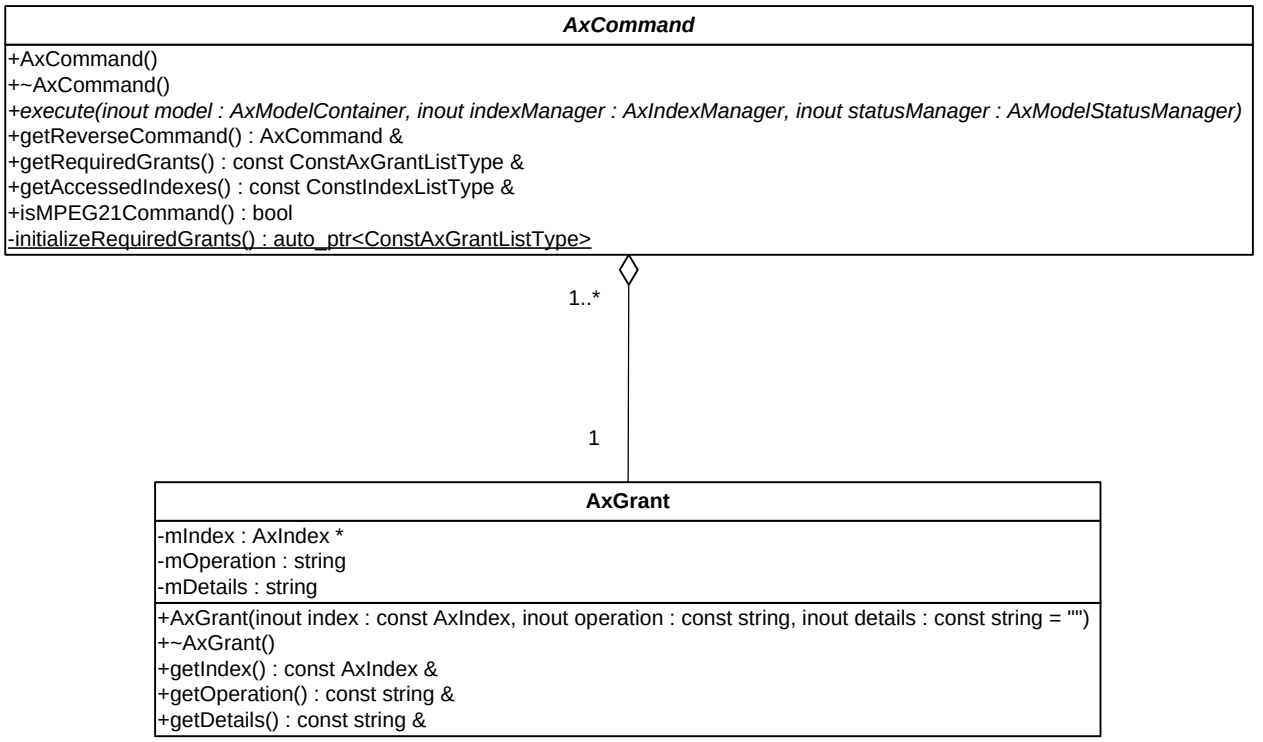
AxIndex Manager – Class methods
AxIndexManager - ~AxIndexManager
Class constructor and destructor
reset
Reset all the element index to an empty value. Delete all AxIndex elements in the index
getRootIndex
Return index of the root element of AxIndexManager.
getIndexOf
Get the AxIndex of input element.
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 modofy 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() : cosntConstAxGrantListType &

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 ot 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 tbp : 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 Examples of usage

In the following, a subset of the commands which are at disposal in the AXOM is presented. Each command models a specific manipulation on the content package. The important aspects of each class, inherited by *AxCommand*, are the constructors, which are needed for setting the operation parameters, and additional methods defined in order to get the execution results. The results typically may contain additional information not directly accessible in the modified package.

- *AxCommandAdd*– this command class has been defined to add AxObject elements. It presents constructors to impose the entry point of the addition*: AxCommandAdd(AxObjectElement newElement, AxIndex parentIndex*) to add an element at the end of the list of descriptors or components*; AxCommandAdd(AxObjectElement newElement, AxIndex parentIndex, AxIndex referenceIndex, bool insertBefore*) to insert a new content element before or after a given element in the list *AxIndex getIndexOfAddedElement()* to obtain after the execution the logical reference of an added new element
- *AxCommandDelete* – command class defined to reduce the content package by deleting descriptions, digital resources or inner packages. A constructor is needed in order to select the target element to be removed: *AxCommandDelete(AxIndex deleteIndex)*
- *AxCommandEdit* – command class defined to edit the attributes included in the content elements of the package:. *AxCommandEdit(AxObjectElement dataElement, AxIndex editIndex*) change the attributes of the target element by copying them from *dataElement;*

The usage of AXOM functionalities is quite simple. For the manipulation of content package the creation of a new command and its execution are needed. The AXOM is responsible of providing indexes for the root level or for any sub-tree of the package hierarchy.
In the following a simplified version of AXOM usage is reported by using a pseudo-code and adding comments, this highlight the semantic meaning of the performed actions.

```
// creating a manager to manipulate a new AXMEDIS object
AxObjectManager myEmptyObject = new AxObjectManager();
// creating a resource element targeting to a digital resource URL (a jpeg image)
AxResource myDigitalRes = new AxResource();
myDigitalRes.load(“bar.jpg”);
// performing the addition of the created resource in the empty object
// step1: creation of the suitable command object
AxCommandAdd addCmd = new AxCommandAdd(myDigitalRes, myEmptyObject.getRootIndex());
// step2: execution of the command
myEmptyObject.executeCommand(addCmd);
// step3: (optional) gathering of the results: the index of the added element
AxIndex addedResIndex = addcmd.getIndexOfAddedElement();
```

The usage of the AXOM hides the DRM verification of grants which can be needed to authorize the manipulations. The AXOM is capable of controlling the execution of the above mentioned commands, delegating to them the specification of the grants needed and the accesses to the content elements. On the other hand, only the AXOM can (i) request the grant authorization from the Authorization Service, and can (ii) unprotect the content package.

It 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).

3.4 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 – 3-1-2-3-13

5 Object Registration (DSI)

The Registration of the AXMEDIS Object put the object in a state that it can be distributed outside the factory. This registered object could reach the user through the Web, the P2P networks, the physical media.
The Registration is the guarantee for the final/business user that the package he receives is actually what is intended to be, the metadata and the embedded media resource has to be certified by a trusted authority.

So the Registration process include two sub-processes:

- The calculation of the hash
- The sending of the calculated hash, plus the metadata and the protection information, obtaining the signed hash.

Some pieces of information like the metadata and the protection parameters, are simply extracted from the AXMEDIS Object Model. The calculation of the Object Hash for the Object Signature requires some steps, due to the potential complexity of the hierarchy. The next section will examine in details these aspects.

5.1 Calculating AXMEDIS Object hash

AXMEDIS Object are MPEG-21 Document that is mainly an XML document. So the basis of that process, is the XML Signature standard.
The AXMEDIS object can be published in two formats:

- Simple XML format
- MPEG-21 FileFormat

In the first case the media resources are embedded inside the XML document by encoding them in base64.
In the second case the media resources are attached in the ISOMedia format outside the XML document that is referencing by specific URI.

In both cases the presence of media resource arises a problem in the signature calculation process.
The hash of the MPEG-21 document and the hash of the media resource that are included/connected to the model, need to be done separately.

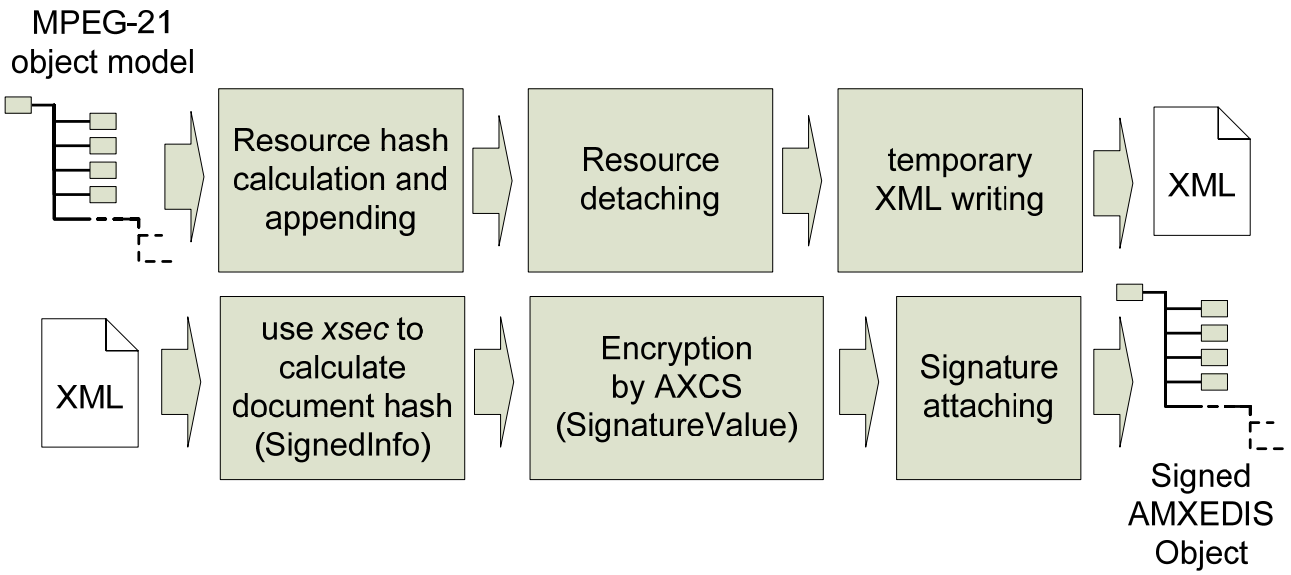
The integrity of MPEG-21 Document and media resource is assured by the following algorithm:

1. for each media resource calculate a hash of the resource bytes and write a descriptor associated to the resource that explicitly states the hash value of the resource
2. detach all the media resource from the Document (it is not needed in the file format)
3. calculate the hash of the resulting document

After these steps it is not possibleto change the resources without detection, because of the associated hash value. And the latter cannot be changed because of the document’s signature (encrypted hash).

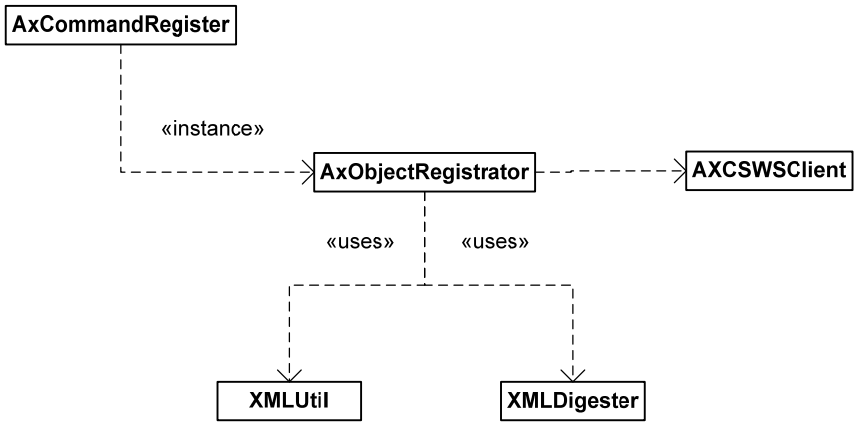
From the technological point of view the XMLSignature library (xsec) by Apache has been used.
That library implements the W3C XMLSignature standard by using Xerces and Xalan libraries. The only problem is that xsec is capable of apply a signature on a XML Document which has been loaded by DOM standard.

The DOM Structure of the document is not the basis of the MPEG-21 Object Model, so the process can be sketched as in the following.



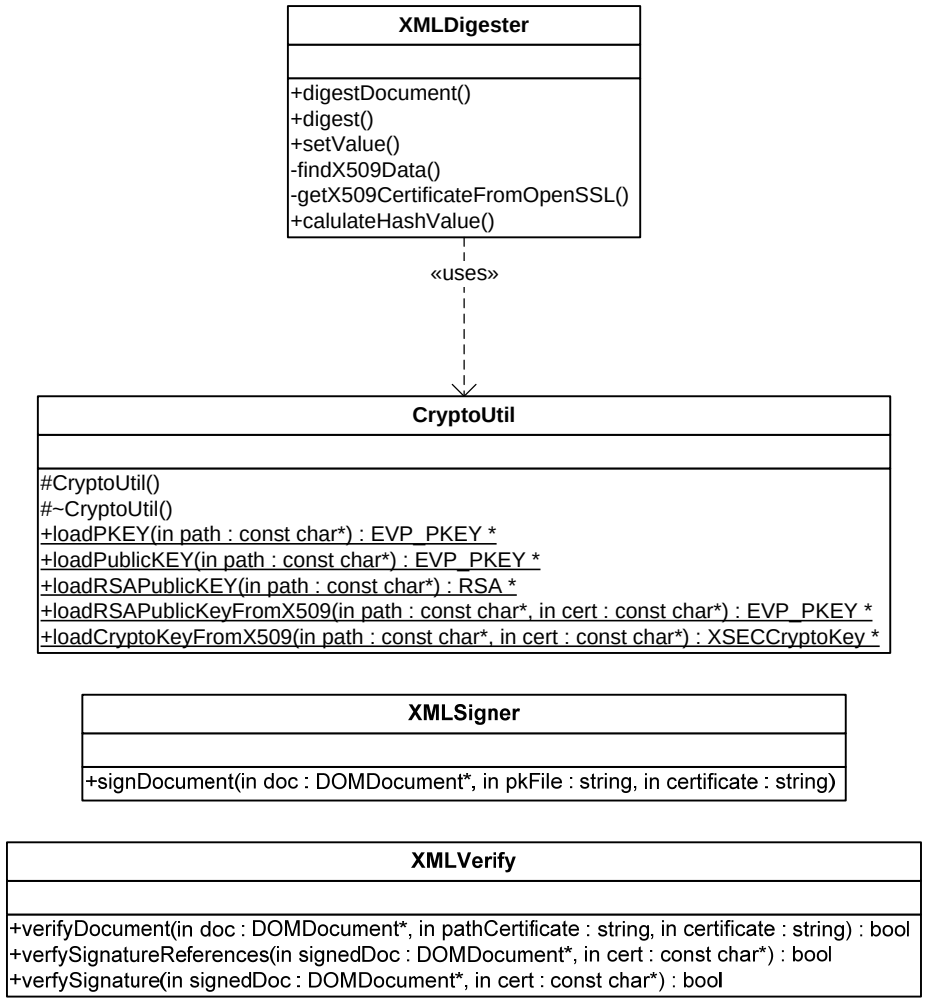
Please note that the temporary writing/re-parsing of the XML document is needed in order to allow xsec to work with familiar structures.

Some classes have been defined in order to simplify the use of xsec library.
These classes are used by the axobjectregistrator that exploits this classes to calculate XML hash for the signature.



The class XMLUtil has been developed to simplify the DOM loading of the temporary XML file.
The XMLDigester uses XMLSignature Apache library in order to calculate the XML document hash value.
The AXCSWSClient is a GSOAP generated Web Service client for the ObjectRegistrator Service of AXCS.

In the following the classes created for evaluating hash and for testing its behaviour are reported.



The encryption will be performed by the AXMEDIS Certifier and Supervisor and the encrypted hash will be returned by Certification Protocol response.
For the Certification Protocol details see AXMEDIS – DE – 3-1-2-3-13