INTERNATIONAL STANDARD

ISO/IEC 15938-12

First edition
2008-12-15

AMENDMENT 1
2011-03-01

Information technology — Multimedia content description interface —

Part 12: **Query format**

AMENDMENT 1: Reference software and flat metadata output

Technologies de l'information — Interface de description du contenu multimédia

Partie 12: Format de requête

AMENDEMENT 1: Logiciel de référence et sortie de métadonnées plates

Reference number
ISO/IEC 15938-12:2008/Amd.1:2011(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to ISO/IEC 15938-12:2008 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — Multimedia content description interface —

Part 12: Query format

AMENDMENT 1: Reference software and flat metadata output

In Clause 9, Output Description, 9.2 Syntax, add the highlighted part:

```
<complexType name="FieldType">
  <simpleContent>
    <extension base="mpqf:xPathType">
      <attribute name="typeName" type="string" use="optional"/>
      <attribute name="fromREF" type="IDREF" use="optional"/>
      <attribute name="fieldREF" type="IDREF" use="optional"/>
      <attribute name="resultMode" use="optional" default="structured">
        <simpleType>
          <restriction base="string">
            <enumeration value="flat"/>
            <enumeration value="structured"/>
          </restriction>
        </simpleType>
      </attribute>
    </extension>
  </simpleContent>
</complexType>
```

In Clause 9, Output Description, 9.3 Semantics, add this row at the end of the table

`resultMode`	If set to `flat`, specifies the desire that the selected metadata fragment appears in a `FragmentResult` element in each result item. If the value of this attribute is not specified, or is set to `structured`, the `Description` element will be used instead (carrying all the metadata fragments selected by all the `ReqField` elements).

In Clause 9, Output Description, 9.3 Semantics, replace this row:

`ReqField`	Describes a data path within the item's metadata, which a requester asks to be returned. Paths are specified by making use of absolute XPath expressions, which refer to the root of the item's metadata, or optionally using relative XPath expressions referred to a given schema's complex type.

with:

`ReqField`	Describes a data path within the item's metadata, which a requester asks to be returned. Paths are specified by making use of relative XPath expressions, which refer to the root of the evaluation item's metadata (the one specified by the `EvaluationPath` element), or optionally using absolute XPath expressions referred to the root of the multimedia content's metadata (to which the evaluation item belongs). Depending on the value of the `resultMode` attribute, the resulting metadata fragments of the different `ReqField` elements will appear in several `ResultField` elements or within a single `Description` element (or two if a `Join` operation is used).

In Clause 9, Output Description, 9.4 Example, add the following example at the end (after the current example):

This second example illustrates the use of `resultMode` attribute set to "`flat`" to obtain a flat metadata output. In this example, a simple free text query is specified which searches for textual descriptions containing "San Jose". In addition, the target domain is limited to images of the JPEG format. The `OutputDescription` element is used to select two fields from the metadata of the resulting digital items (width and height of the image), with the `resultMode` attribute set to "`flat`".

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <OutputDescription maxItemCount="30" maxPageEntries="10" freeTextUse="true"
outputNameSpace="urn:mpeg:mpeg7:schema:2004">
        <ReqField typeName="width"
resultMode="flat">MediaInformation/MediaProfile/MediaFormat/VisualCoding/Frame/@w
idth</ReqField>
      <ReqField typeName="height"
resultMode="flat">MediaInformation/MediaProfile/MediaFormat/VisualCoding/Frame/@h
eight</ReqField>
      </OutputDescription>
      <QueryCondition>
        <TargetMediaType xsi:type="mimeType">image/jpeg</TargetMediaType>
        <Condition xsi:type="QueryByFreeText">
          <FreeText>San Jose</FreeText>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

The following is the example of an expected output by the specified `OutputDescription` above.

```
<MpegQuery>
  <Query>
    <Output currPage="1" totalPages="1" expirationDate="2008-05-30T09:00:00">
      <ResultItem xsi:type="ResultItemType" recordNumber="1">
        <TextResult>Title 01</TextResult>
        <mpqf:FragmentResult name="width">640</mpqf:FragmentResult>
        <mpqf:FragmentResult name="height">480</mpqf:FragmentResult>
      </ResultItem>
      <ResultItem recordNumber="2">
        <TextResult>Title 02</TextResult>
        <mpqf:FragmentResult name="width">320</mpqf:FragmentResult>
```

```
        <mpqf:FragmentResult name="height">200</mpqf:FragmentResult>
      </ResultItem>
      <ResultItem recordNumber="3">
        <TextResult>Title 03</TextResult>
        <mpqf:FragmentResult name="width">800</mpqf:FragmentResult>
        <mpqf:FragmentResult name="height">1000</mpqf:FragmentResult>
      </ResultItem>
    </Output>
  </Query>
</MpegQuery>
```

In 13.2, ResultItem, 13.2.2 Syntax, add the highlighted part:

```
<complexType name="ResultItemBaseType" abstract="true"/>
  <complexType name="ResultItemType">
    <complexContent>
      <extension base="mpqf:ResultItemBaseType">
        <sequence>
          <element name="Comment" minOccurs="0" maxOccurs="2">
            <complexType>
              <simpleContent>
                <extension base="string">
                  <attribute name="fromREF" type="string" use="optional"/>
                </extension>
              </simpleContent>
            </complexType>
          </element>
          <!-- Need for comment for each individual item should be cleared. -->
          <!-- One use case can be for each individual responder to identify the
origin of        the result. -->
          <element name="TextResult" minOccurs="0" maxOccurs="2">
            <complexType>
              <simpleContent>
                <extension base="string">
                  <attribute name="fromREF" type="string" use="optional"/>
                </extension>
              </simpleContent>
            </complexType>
          </element>
          <element name="Thumbnail" minOccurs="0" maxOccurs="2">
            <complexType>
              <simpleContent>
                <extension base="anyURI">
                  <attribute name="fromREF" type="string" use="optional"/>
                </extension>
              </simpleContent>
            </complexType>
          </element>
          <element name="MediaResource" minOccurs="0" maxOccurs="2">
            <complexType>
              <simpleContent>
                <extension base="anyURI">
                  <attribute name="fromREF" type="string" use="optional"/>
                </extension>
              </simpleContent>
            </complexType>
          </element>
```

```
          <!-- The media resource is expected to lead the customer to the
location
         of the actual full size media. -->
          <element name="Description" minOccurs="0" maxOccurs="2">
            <complexType mixed="true">
              <sequence>
                <any          namespace="##any"          processContents="strict"
maxOccurs="unbounded"/>
              </sequence>
              <attribute name="fromREF" type="string" use="optional"/>
            </complexType>
          </element>
          <!-- If you want to return embedded in-line media, you should use the
Description. For example, you should instantiate a mpeg7:MediaLocator with inline
media -->
          <element name="AggregationResult" minOccurs="0" maxOccurs="unbounded">
            <complexType>
              <simpleContent>
                <extension base="string">
                  <attribute name="aggregateID" type="string" use="required"/>
                </extension>
                <!-- This aggregateID is given in the Aggregate element
                  of the Input Query. -->
              </simpleContent>
            </complexType>
          </element>
          <element name="FragmentResult" minOccurs="0" maxOccurs="unbounded">
           <complexType>
            <simpleContent>
             <extension base="string">
               <attribute name="name" type="string" use="required"/>
              <attribute name="fromREF" type="string" use="optional"/>
             </extension>
            </simpleContent>
           </complexType>
          </element>
           <!--  elements with names of each aggregate expression -->
        </sequence>
        <attribute name="recordNumber" type="positiveInteger" use="required"/>
        <attribute name="rank" type="positiveInteger" use="optional"/>
        <attribute name="confidence" type="mpqf:zeroToOneType" use="optional"/>
        <attribute name="originID" type="anyURI" use="optional"/>
        <!-- Can contain the serviceID  or URL of the responder responding to the
Input
Query, when there are multiple services responding to the single request. -->
      </extension>
    </complexContent>
  </complexType>
```

In 13.2, ResultItem, 13.2.3 Semantics, add this row at the end of the table:

`FragmentResult`	Contains a metadata fragment selected by a `ReqField` element in the output description of the input query in a flat string form. It may be just a number like “65”, or XML data which is packed as CDATA. It is an alternative way to the `Description` element to get selected metadata from the result items.

After Clause 14, Query Management Tools, add the following:

15 MPEG Query Format Reference Software

15.1 Introduction

The following Subclauses describe reference software for the normative clauses of this Part of ISO/IEC 15938. The information provided is applicable for determining the reference software modules available for this Part of ISO/IEC 15938, understanding the functionality of the available reference software modules, and utilizing the available reference software modules.

In addition to the reference software, available (integrated) utility software that utilizes the reference software is also described. This utility software can assist in understanding how to utilize the reference software, as well as providing further insight into this Part of ISO/IEC 15938, e.g. informative Clauses.

15.2 MPQF Reference Software specific terms, definitions and conventions

15.2.1 Terms, definitions, symbols and abbreviated terms

15.2.1.1 module

software component implementing **reference software** or **utility software**

15.2.1.2 reference software

one or more **module**s utilizing normative parts of this Part of ISO/IEC 15938

15.2.1.3 utility software

one or more **module**s utilizing informative parts of this Part of ISO/IEC 15938 and/or the usage of **reference software** within real-world applications

15.2.2 Conventions

In the remainder of this Clause, each reference and utility software module is described following the convention as below:

Module name	Name of the ZIP file with the following structure: /<directory>/<module_name>-<implementation>-<version>.zip <directory>: directory name in which the module can be found 15938-12 <module_name>: name of the module, e.g., Parser, Validator, etc. <implementation>: letter A, B, C, etc. for different implementations. <version>: version number, i.e., n_n_n | n_n | n
Description	Describes the functionality the module provides.
INPUT	Describes the input of the module.
OUTPUT	Describes the output of the module.

Programming Language(s)	Lists the programming language(s) in which the module is written.
Platform(s)	Lists the platforms the module has been tested on and is supposed to run on.
Dependencies	Lists the required libraries and code with version information.
Details	Lists any implementation details, such as architecture diagrams and data flows.

15.3 Overview of the architecture of the 15938-12 reference software

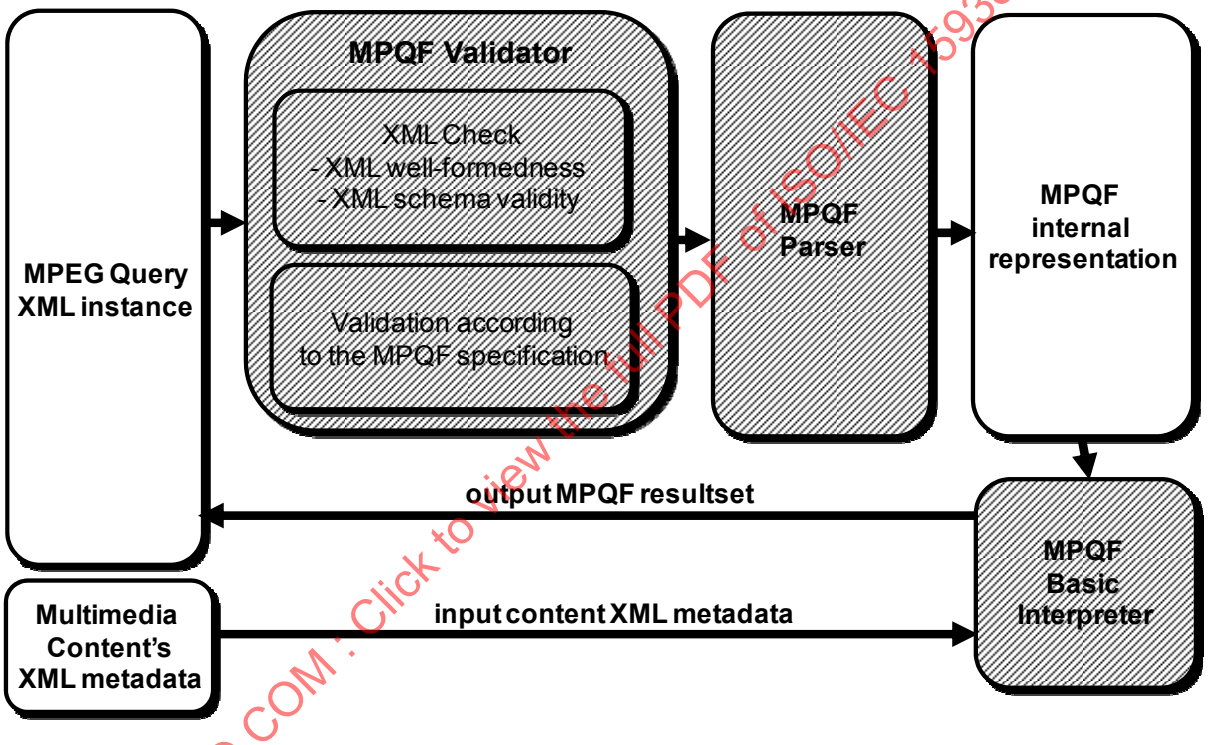

Figure 5 — Reference/utility software architecture

The architecture (see Figure 5) of the Reference Software is divided in three different software modules, the MPQF Validator, the MPQF Parser and the MPQF Basic Interpreter. These software modules are defined in a composite way, the Basic Interpreter makes use of the Parser and the Parser makes use of the Validator.

The MPQF Validator first checks the XML well-formedness and validity of an MPQF input/output query according to the rules of XML 1.1 and the MPQF XML schema. Secondly, the Validator checks if the input or output query is compliant with the rules described in this part of ISO/IEC 15938 which cannot be enforced with the XML schema.

Once the Validator has checked the validity of the MPQF query, the MPQF Parser translates this XML instance into a Java object provided with methods for accessing and modifying the different parts of the query. This Java object is the output of the Validator.

The MPQF Basic Interpreter module receives from the Parser a Java object representing a query and also an input XML file containing MPEG-7 metadata about a collection of images. However, note that MPQF is metadata aqnostic and any other metadata format can be used in combination with the query format. The Basic Interpreter will evaluate the query and will return another Java object representing the response (an output query). This object is then passed to the Parser who will translate it to an XML output MPQF instance.

This Part of ISO/IEC 15938 comprises reference software modules. The following table summarizes the modules:

module name	description
MPQF Validator	-XML well-formedness and schema validity -Validation according to the MPQF specification
MPQF Parser	-Parsing an MPQF instance into its internal data structure -Serializing the internal data structure to a valid MPQF instance
Basic Interpreter	Basic queries without query types

15.4 MPQF Validator

Module name	/15938-12/MPQF_Parser-1_0_0.zip
Description	- XML well-formedness and schema validity - Validation according to the MPQF specification.
INPUT	An MPQF query; URI of the profile used (default = no profile).
OUTPUT	- Well formed, not well formed + reasons why, valid, not valid + reasons why (according to the MPQF XML schema) - Valid, not valid + reasons why (according to the MPQF specification)
Programming Language(s)	Java version 1.5 or higher
Platform(s)	Any platform that supports the programming language
Dependencies	None
Details	-

15.4.1 MPQF Validator Framework

The MPQF validator provides an extensible module based framework which allows an independent development and assembly of verification components. Verification components can be divided into two main groups: syntactic and semantic verification. Syntactic verification deals with the evaluation of XML documents according to the following two characteristics: well-formed and valid. A XML document is well-formed if it obeys the syntax of XML. Furthermore, a XML document is valid if it obeys the syntax of the underlying XML Schema. Related to the MPQF validator, a MPQF query is syntactical correct if it is well-formed and valid according to the MPQF XML Schema.

Semantic verification deals with the evaluation of rules that are not expressed by syntactic means within the XML Schema. For instance, a query may be valid for one multimedia retrieval service (MMRS) but invalid for another one. In series, this can depend on different capabilities the individual MMRS support (e.g., different query types are supported). Another semantic rule emerges in combination with internal references between resources and query types. There are query types which reference to resources at the declaration level in order to increase the reuse of components. However, specific query types are only allowed to point to specific resources. This must be evaluated by the MPQF validator.

In order to support an extensible approach at the best, Figure 6 presents the internal workflow of the system. Whenever an instance of the validator is created a corresponding validation chain is instantiated. A validation chain consists of a set of validation modules which are selected for the individual validation process. An overview of currently available validation modules is presented in 15.4.4.

The validation process evaluates the incoming MPQF query by traversing the validation chain step by step. During this process every validation module verifies the query according to their specific rules (syntactic or semantic). In case of an error, the validation stops and the respective error message is returned.

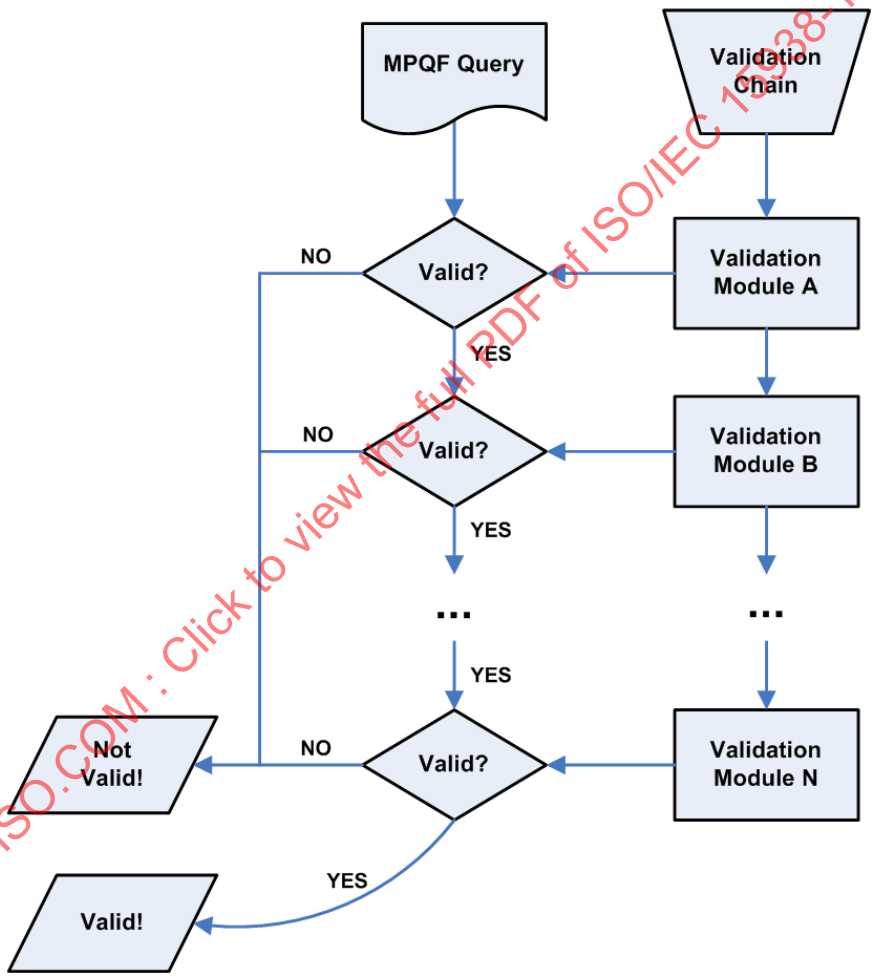

Figure 6 — Workflow of the MPQF validator

15.4.2 Class Hierarchy

Figure 7 demonstrates the class hierarchy of the MPQF validator, where in general three different parts can be distinguished: public classes, validation modules and internal package. In the following, the individual parts are explained in more detail.

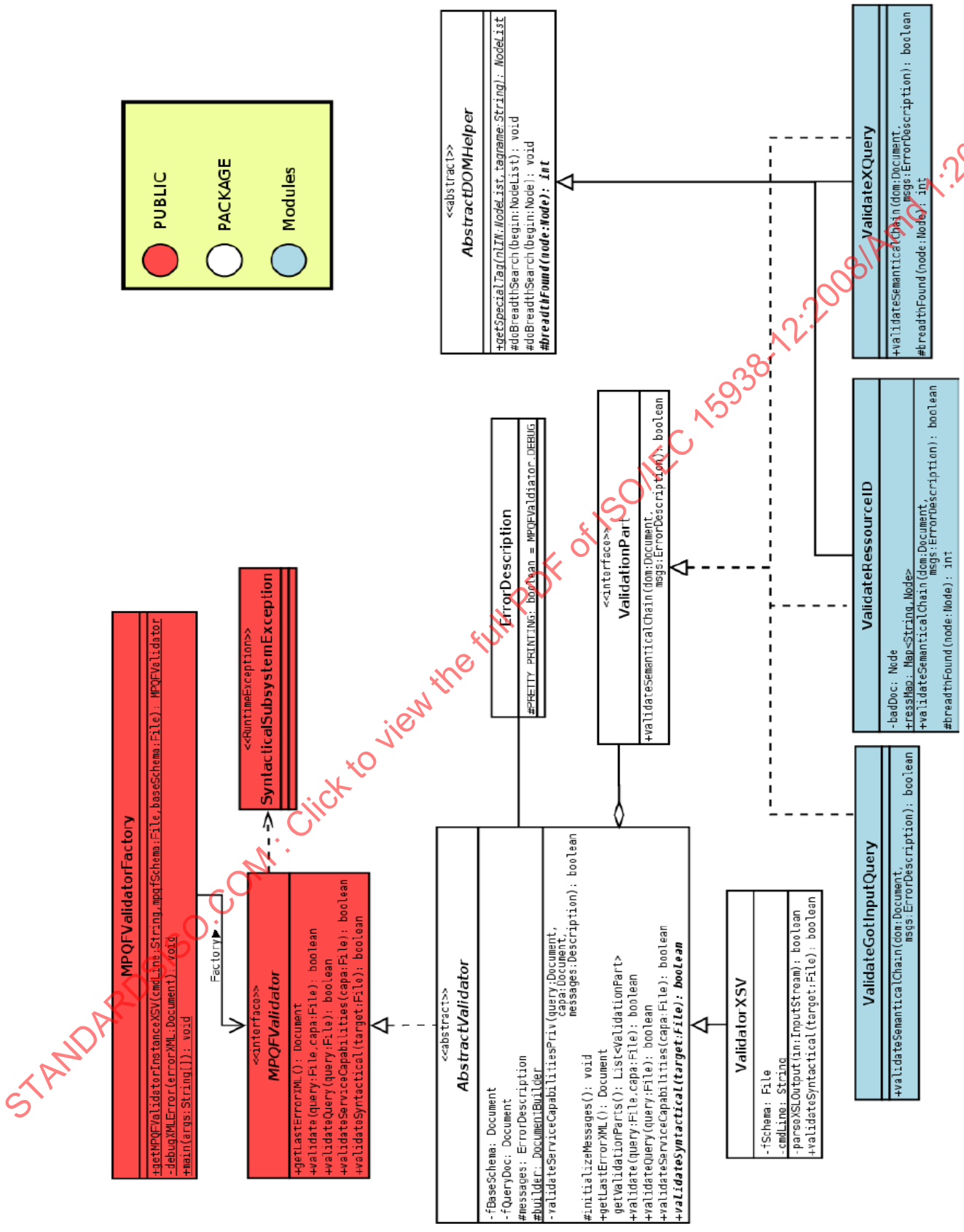

Figure 7 — Class Hierarchy

15.4.3 Public classes

MPQFValidatorFactory

The *MPQFValidatorFactory* realizes the factory pattern software concept which allows the generation of MPQFValidator instances. Furthermore, it provides a main method in order to use the software as a standalone validator client. In this case, the syntactical validation is fixed to the XSV tool. The factory provides a large set of configuration options. For instance, one has the possibility to deliver the MPQF-Schema, the description Schema (schema which describes the information provided in an DescriptionResource field), the query which should be evaluated, the classification scheme files and a service capability description file determining the multimedia repository service which is the target for the execution of the query. Examples for the usage of the factory are provided in 15.4.7.

MPQFValidator

The MPQFValidator is an interface which dictates the public methods of every validator implementation.

SyntacticalSubsystemException

In case this exception is thrown, it symbolizes a configuration error (e.g., wrong amount/type of parameter in the command line) in one of the syntactical validation modules.

15.4.4 Validation Modules

15.4.4.1 Syntactic Validation

ValidateXSV

The ValidateXSV module accomplishes syntactic validation by using the XSV tool (see http://www.ltg.ed.ac.uk/~ht/xsv-status.html).

ValidateAltova

The ValdiateAltova module accomplishes syntactic validation by using the Altova tool (see http://www.altova.com/altovaxml.html).

ValidateXerces

The ValidateXerces module accomplishes syntactic validation by using the Xerces tool (see http://xerces.apache.org/xerces-j/).

15.4.4.2 Semantic Validation

ValidateXQuery

The ValidateXQuery module evaluates an incoming query by verifying that, in case it contains one or more QueryByXQuery elements, the XQuery expressions embedded in them satisfy the following constraints:

- The embedded XQuery expressions are compliant with the XQuery 1.0 specification (according to the Saxon 9.0 implementation).
- It cannot be determined at compile time that the embedded XQuery expressions will return something different from a Boolean value. Otherwise, they won't be valid according to 12.10 of this part of ISO/IEC 15938.

In case the module cannot determine at compile time the return type of an XQuery expression, it will be considered valid and a warning message will be returned suggesting to the user wrapping the expression within the XQuery's fn:Boolean function.

ValidateCapabilities

The ValidateCapabilities module evaluates an incoming query according to the given service capability description of the target multimedia retrieval system (MMRS). During this test, it is verified whether all used query types, algebraic operations, metadata formats, etc. are covered by the respective capability description.

ValidateGroupBy

The ValidateGroupBy module evaluates an incoming query according to GroupByField elements that describes the key for grouping process. It is verified whether all GroupByField elements in GroupBy element are also defined as ReqField elements in OutputDescription element.

ValidateResourceID

The ValidateResourceID module evaluates an incoming query according to the internal linkage of resources. The module guarantees that resources are referenced correctly. Note that, this module should be enhanced for verifying also type safety (e.g., description resource is only referenced by a QueryByDescription query type).

ValidateGotInputQuery

The ValidateInputQuery module evaluates an incoming query by verifying that it contains a *Query* and *Input* tag. This ensures that the XML instance document is a query request.

ValidateRelativeField

The ValidateRelativeField module evaluates an incoming query by verifying that if *typeName* is specified for a *DeclaredFieldType,* only a relative XPath expression is allowed.

15.4.5 Internal Package

AbstractValidator

The AbstractValidator is an abstract class and implements the MPQFValidator interface. It provides some basic functionality for XML parsing and processing. Furthermore, basic functionality for service capability descriptions and classification schemes is given.

ValidationPart

The ValidationPart interface dictates the methods every validation module must implement. In order to keep naming consistency, every validation module which is planned to be used within a validation chain must begin with the name prefix *Validate*.

AbstractDOMHelper

This abstract class provides basic functionality for traversing (breadth search) the query which internally is transferred to a DOM tree. Besides, the extraction of individual nodes within the tree is supported. Another feature is the assistance in creating the result XML file containing the validator evaluation messages which is forwarded to the user.

ErrorDescription

This class holds the final error message and provides means for its manipulation. The structure of the error message is defined by the result messages XML Schema which is described in 15.4.6.

15.4.6 XSD Schema for Result Messages

The following XSD Schema describes the structure of a result message which can be collected after the last validation module is executed. The result message can be retrieved by calling the *getLastErrorXML()* method.

```
<?xml version="1.0" encoding="UTF-8"?> <schema
xmlns:mpqfval="urn:mpeg:mpqfval:schema:2006"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
targetNamespace="urn:mpeg:mpqfval:schema:2006">
    <!-- ################################################################ -->
    <!-- Syntactical-->
    <!-- ################################################################ -->
    <complexType name="MPQFValidatorSyntacticalType">
        <sequence>
            <any namespace="##any"/>
        </sequence>
        <attribute name="valid" type="boolean" use="required"/>
        <attribute name="crash" type="boolean" use="required"/>
        <attribute name="triedLax" type="boolean" use="optional"/>
        <attribute name="validator" use="required">
            <simpleType>
                <restriction base="string">
                    <enumeration value="XSV"/>
                    <enumeration value="Xerces"/>
                </restriction>
            </simpleType>
        </attribute>
    </complexType>
    <!-- ################################################################ -->
    <!-- Semantical-->
    <!-- ################################################################ -->
    <complexType name="MPQFValidatorSemanticalType">
        <simpleContent>
            <extension base="string">
                <attribute name="chainObject" type="string"
                           use="required"/>
                <attribute name="valid" type="boolean" use="required"/>
            </extension>
        </simpleContent>
    </complexType>
    <!-- ################################################################ -->
    <!-- ValidatorException -->
    <!-- ################################################################ -->
    <complexType name="MPQFValidatorExceptionType">
        <simpleContent>
            <extension base="string">
                <attribute name="ExceptionName" type="string"
                           use="required"/>
                <attribute name="RuntimeException" type="string"
                           use="required"/>
                <attribute name="valid" type="boolean" use="required"/>
            </extension>
        </simpleContent>
    </complexType>
    <!-- ################################################################ -->
    <!-- Top Level Element -->
    <!-- ################################################################ -->
    <complexType name="MPQFValidatorType">
        <choice>
            <sequence>
                <element name="Syntactical"
                   type="mpqfval:MPQFValidatorSyntacticalType"/>
                <sequence minOccurs="0" maxOccurs="unbounded">
                    <element name="Semantical"
```

```
                        type="mpqfval:MPQFValidatorSemanticalType"/>
                </sequence>
            </sequence>
            <element name="ValidatorException"
                        type="mpqfval:MPQFValidatorExceptionType"/>
        </choice>
    </complexType>
</schema>
```

15.4.7 Installation / Utilization

The MPQF Validator comes as Java jar file and relies on Java 1.5 installation on the target computer. In addition, in order to enable syntactic validation the respective external tool needs to be installed (e.g., XSV, Xerces, etc.). The validator can be used as standalone application or might be embedded as Java object by using the public factory interfaces.

The standalone version can be used by executing the following command:

java -classpath ./MPQFValidator.jar de.dimis.mpqf.validator.MPQFValidatorFactory <syntactic external tool> <MPQF base schema> <target schema> <query> <service capability description>

The following example uses XSV as external tool for validation. Note that the paths need to be adopted to the target system:

java -classpath ./MPQFValidator.jar de.dimis.mpqf.validator.MPQFValidatorFactory C:\Programme\Tools\XSV\xsv.exe

C:\MPQF\schema\mpqf_final.xsd C:\MPQF\reference_software\validator\etc\schema\M7v2schema.xsd C:\MPQF\reference_software\validator\etc\tests\simple_ok.xml
C:\MPQF\reference_software\validator\etc\tests\capa_empty.xml

The following XML instance document shows a possible result of a query validation. The verified query is syntactically correct (here the output of the XSV tool has been integrated) and the following semantic rules have been successfully applied: ValidateGotInputQuery, ValidateResourceID, ValidateFieldTypes.

```
<?xml version="1.0" encoding="UTF-8"?>
<MPQFValidator
NS1:valid="true" xmlns="urn:mpeg:mpqfval:schema:2008"
xmlns:NS1="urn:mpeg:mpqfval:schema:2008">
   <Syntactical NS1:crash="false" NS1:valid="true" NS1:validator="XSV">
      <xsv docElt="{urn:mpeg:mpqf:schema:2008}MpegQuery"
         instanceAssessed="true" instanceErrors="0"
         rootType="{urn:mpeg:mpqf:schema:2008}:MpegQueryType"
         schemaDocs="file:/C:/MPQF/schema/mpqf_final.xsd"
         schemaErrors="0"
target="file:/C:/MPQF/reference_software/validator/etc/tests/sem_xquery_valid_boo
lean.xml"
      validation="strict" version="XSV 2.10-1 of 2005/04/22 13:10:49"
      xmlns="http://www.w3.org/2000/05/xsv">
         <schemaDocAttempt URI="file:///C:/MPQF/schema/mpqf_final.xsd"
            outcome="success" source="command line"/>
      </xsv>
   </Syntactical>
   <Semantical
      NS1:chainObject="de.dimis.mpqf.validator.ValidateGotInputQuery"
NS1:valid="true">&lt;Input&gt;-tag found in the query.
```

```
   </Semantical>
   <Semantical
      NS1:chainObject="de.dimis.mpqf.validator.ValidateResourceID"
NS1:valid="true">All referenced resource-IDs found in declaration.
   </Semantical>
   <Semantical
      NS1:chainObject="de.dimis.mpqf.validator.ValidateFieldTypes"
NS1:valid="true">Condition fields typechecked (partially yet).
   </Semantical>
</MPQFValidator>
```

15.5 MPQF Parser

Module name	/15938-12/MPQF_Parser-1_0_0.zip
Description	- Parsing an MPQF instance into its internal data structure - The Internal data structure is a Java based one by one representation of the MPQF Schema types providing means in order to access and modify and MPQF instance. - Serializing the internal data structure to a valid MPQF instance
INPUT	- An MPQF query; URI of the profile used (default = no profile) or - An MPQF Java object
OUTPUT	- An MPQF Java object or - An MPQF xml instance document
Programming Language(s)	Java version 1.5 or higher
Platform(s)	Any platform that supports the programming language
Dependencies	MPQF Validator
Details	-

15.5.1 MPQF Parser Framework

The MPQF parser framework provides means for transforming MPQF query instance documents into respective Java objects by a 1 to 1 mapping approach. For this purpose, the XMLBeans (see http://xmlbeans.apache.org) XML data binding technology has been used in order to generate an automatic Java class representation of this part of ISO/IEC 15938.

XMLBeans (see http://xmlbeans.apache.org/) is a XML data binding technology for providing an easier access and process by the use of Java objects. The created Java classes and interfaces support the factory pattern

for instantiating objects that represent the individual XML complex and simple types. The data (attributes and elements) is accessed and modified by getter and setter methods.

The resulting MPQF Java class representation has been integrated into the MPQF parser framework implementation (see Figure 8). In general, a MPQF query can be applied to a heterogeneous set of multimedia repository services (MMRS). Typically, within this set of MMRS a multiplicity of multimedia metadata formats is used. Therefore, the framework can be extended by Java class representations of XML based metadata formats as shown in Figure 8. The current framework contains a Java class representation of the MPEG-7 metadata standard. However, a respective integration into the MPQF parser needs to be accomplished in a future task separately.

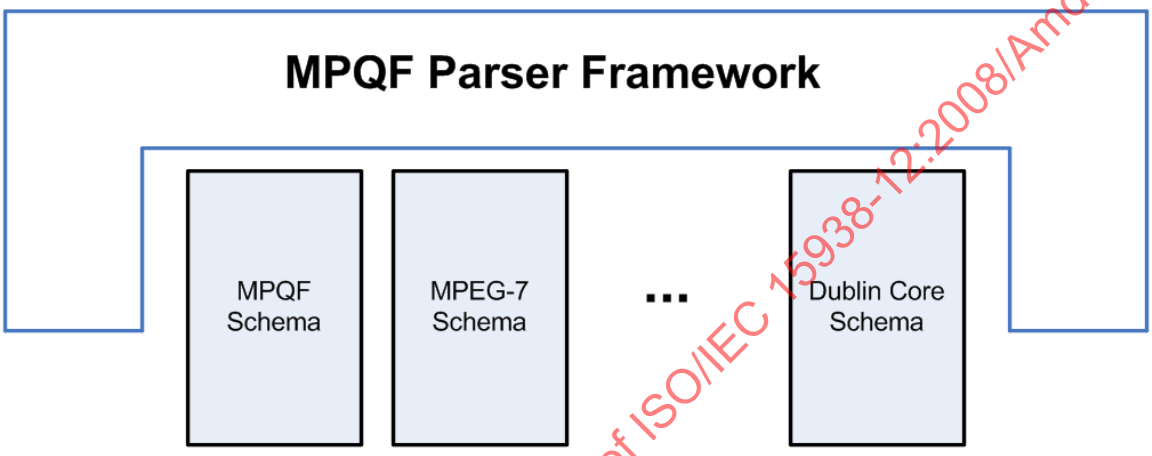

Figure 8 — MPQF Parser Framework

The main idea of integrating Java representations of additional metadata formats relies on the desire to support an easier navigation within information that is provided in the *AnyDescription* element of this part of ISO/IEC 15938. Note, that the current implementation treats this information as string which demands a separate parsing of this part of a query. An example how this can be realized is provided in the *extractMpeg7Input(MpegQueryDocument document)* method. There a parsed MPQF query serves as input and the contained MPEG-7 based *AnyDescription* content is returned.

Integration of new metadata formats

New metadata formats (e.g., Dublin Core) can be integrated by using the XMLBeans *scomp* tool. This tool compiles an XML schema to XMLBeans classes and metadata information. The following example illustrates a possible usage in the context of the MPQF parser framework:

```
scomp -compiler <path to java compiler> -d <target classes folder> -src <target source folder> <XML schemas>

scomp -compiler C:\Programme\Java\jdk1.6.0_07\bin\javac -d C:\XMLBEANS\xmlbeans-2.3.0_binaries\xmlbeans-2.3.0\bin\classes -src C:\XMLBEANS\xmlbeans-2.3.0_binaries\xmlbeans-2.3.0\bin\src mets.xsd dc.xsd
```

For further information, the reader is referred to the following link: http://xmlbeans.apache.org/docs/2.0.0/guide/tools.html#scomp

In a further step, the added XMLBeans classes can be integrated into the MPQF parser interface and implementation.

15.5.2 Class Hierarchy

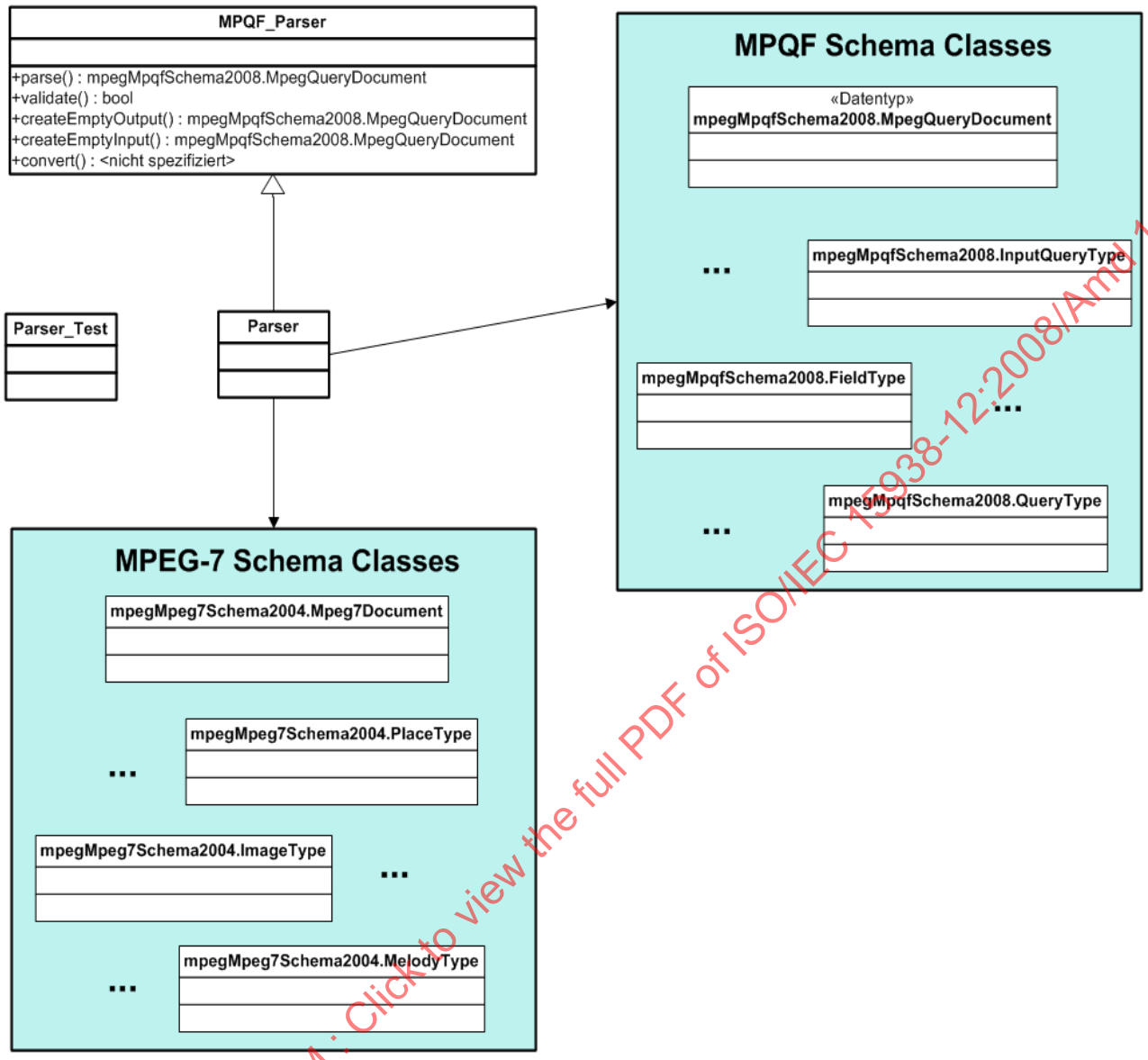

Figure 9 — MPQF Parser Class Hierarchy

Figure 9 presents the overall class hierarchy of the MPQF parser. Note that, not all classes of the respective XML schema packages (MPQF and MPEG-7) are shown. The main entry point of the system is denoted by the *Parser* class which implements the *MPQF_Parser* interface. The parser provides the following main functionality:

15.5.2.1 Parse a Query

Method Information

Parses a MPQF query (stored in a file in the file system) and returns a Java object providing a 1 to 1 mapping.

Method Declaration

```
public mpegMpqfSchema2008.MpegQueryDocument parse(File xmlFile);
```

Parameter Semantic

Name	Type	Semantic
xmlFile	File	Points to a file on the file system which contains a MPQF query (XML instance document) that should be parsed to a Java class. The result is an instantiation of type *MpegQueryDocument* in the package *mpegMpqfSchema2008*.

Method Information

Parses a MPQF query (provided as a String) and returns a Java object providing a 1 to 1 mapping.

Method Declaration

public mpegMpqfSchema2008.MpegQueryDocument parse(String mpqfQuery);

Parameter Semantic

Name	Type	Semantic
mpqfQuery	String	Receives a string which contains a MPQF query (XML instance document) that should be parsed to a Java class. The result is an instantiation of type *MpegQueryDocument* in the package *mpegMpqfSchema2008*.

15.5.2.2 Validate a Query

Method Information

Validates a MPQF query (provided as a Java object) syntactically and returns a Boolean value informing about the validity of the given MPQF query.

Method Declaration

public boolean validate(mpegMpqfSchema2008.MpegQueryDocument doc)

Parameter Semantic

Name	Type	Semantic
doc	MpegQueryDocument	Java object symbolizing the root of a MPQF query.

15.5.2.3 Create Output Query

Method Information

Creates a MPQF Java object that represents an empty output query.

Method Declaration

```
public mpegMpqfSchema2008.MpegQueryDocument createEmptyOutput();
```

15.5.2.4 Create Input Query

Method Information

Creates a MPQF Java object that represents an empty input query.

Method Declaration

```
public mpegMpqfSchema2008.MpegQueryDocument createEmptyInput();
```

15.5.2.5 Convert a Query

Method Information

Converts a MPQF query (given as Java class representation) into a MPQF XML instance document and returns a pointer to a file in the local file system where it has been stored.

Method Declaration

```
public File convert(mpegMpqfSchema2008.MpegQueryDocument doc, String fileName);
```

Parameter Semantic

Name	Type	Semantic
doc	MpegQueryDocument	Java object symbolizing the root of a MPQF query.
fileName	String	File name where the XML instance document should be stored to.

15.5.3 Installation / Utilization

The MPQF parser framework comes as Java jar file and relies on a Java 1.6 installation on the target computer. For the integration of additional XML based multimedia metadata formats, a XMLBeans (http://xmlbeans.apache.org/) installation is needed.

A standalone test version of the parser can be executed by the following command:

java -jar MPQF_Parser-1_0_0.jar TestQuery10.xml

15.6 Basic Interpreter

Module name	/15938-12/MPQF_BasicInterpreter-1_0_0.zip
Description	- Evaluation a simple MPQF request on behalf of an image repository. - The metadata used for the image repository is MPEG-7.
INPUT	- An MPQF input query as an xml file - One MPEG-7 xml instance containing the image repository
OUTPUT	- An MPQF Java object
Programming Language(s)	Java version 1.6 or higher
Platform(s)	Any platform that supports the programming language
Dependencies	NONE

15.6.1 Functionality

The provided MPQF's Basic Interpreter software module serves to help understanding the semantics of certain parts of the language. The table below lists the features which are covered by the provided software.

MPQF feature according to the impl. plan	Description	Covered
Basic conditions	AND, OR, NOT XOR, comparison expressions (only Equal)	YES
Granularity	Different granularities specified with the *EvaluationPath* element below the *QueryCondition* element	YES
Sorting	Any possible usage of the *SortByFieldType* and *SortByAggregateType*	YES
Grouping	Any possible usage of the *GroupBy* element	YES

Sorting	Any possible usage of the *SortByFieldType* and *SortByAggregateType*	YES
Join	JoinType with the evaluation*Path* element	YES

15.6.2 Command line utilization

This module provides a standalone basic interpreter which allows command line testing of MPQF queries over a single MPEG-7 metadata file containing the description of multiple multimedia contents.

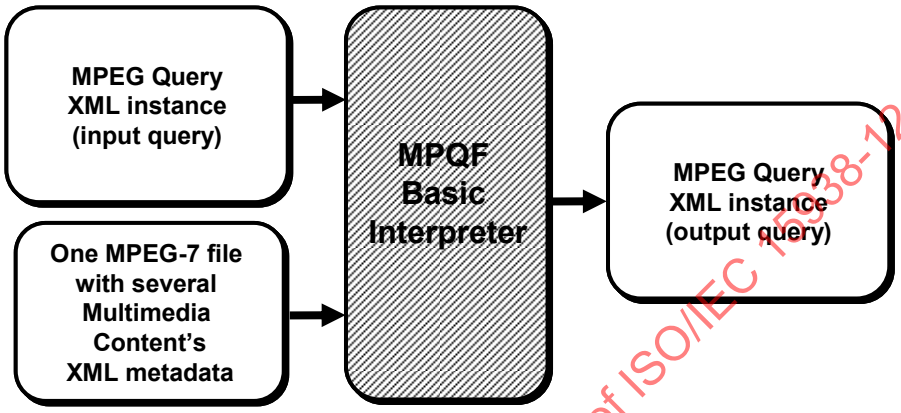

Figure 10 — Overview of the module's functionality

The MPQF Basic Interpreter executable comes as a Java jar file and relies on a Java 1.6 (or higher) installation on the target computer.

The standalone test version of the interpreter can be executed by the following command:

java -Dlog4j.configuration=file:./WEB-INF/classes/log4j.properties -classpath ./WEB-INF/lib/mpqf-1.0.jar;./WEB-INF/lib/xmldb.jar;./WEB-INF/lib/exist.jar;./WEB-INF/lib/log4j-1.2.15.jar;./WEB-INF/lib/xmlrpc-1.2-patched.jar;./WEB-INF/lib/jaxen-1.1.1.jar;./WEB-INF/lib/commons-pool-1.4.jar;./WEB-INF/lib/antlr-2.7.6.jar;./WEB-INF/lib/xercesImpl-2.9.1.jar;./WEB-INF/lib/resolver-1.2.jar;./WEB-INF/lib/quartz-1.6.0.jar;./WEB-INF/lib/commons-logging-1.0.4.jar;./WEB-INF/lib/jta.jar;./WEB-INF/lib/commons-collections-3.1.jar;./WEB-INF/lib/stax-api-1.0.1.jar;./WEB-INF/lib/caliph-emir-cbir.jar;./WEB-INF/lib/lucene-core-2.1.0.jar;./WEB-INF/lib/lire.jar ***org.barcelonatech.kaiko.MPQFTester [testquery.xml] [testMPEG7file.xml] [outputfile.xml]***

A .bat/.sh script which instantiates this commands with two parameters is provided:

mpqf.bat ***[testquery.xml] [testMPEG7file.xml] [outputfile.xml]***

The directory *test* contains several test queries and a test MPEG-7 file with the descriptions of different images. For example:

mpqf.bat WEB-INF/classes/xml/test1_1_emptyquery.xml imageDB/test/xml/mc_metadata1_mpeg7images.xml