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)

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

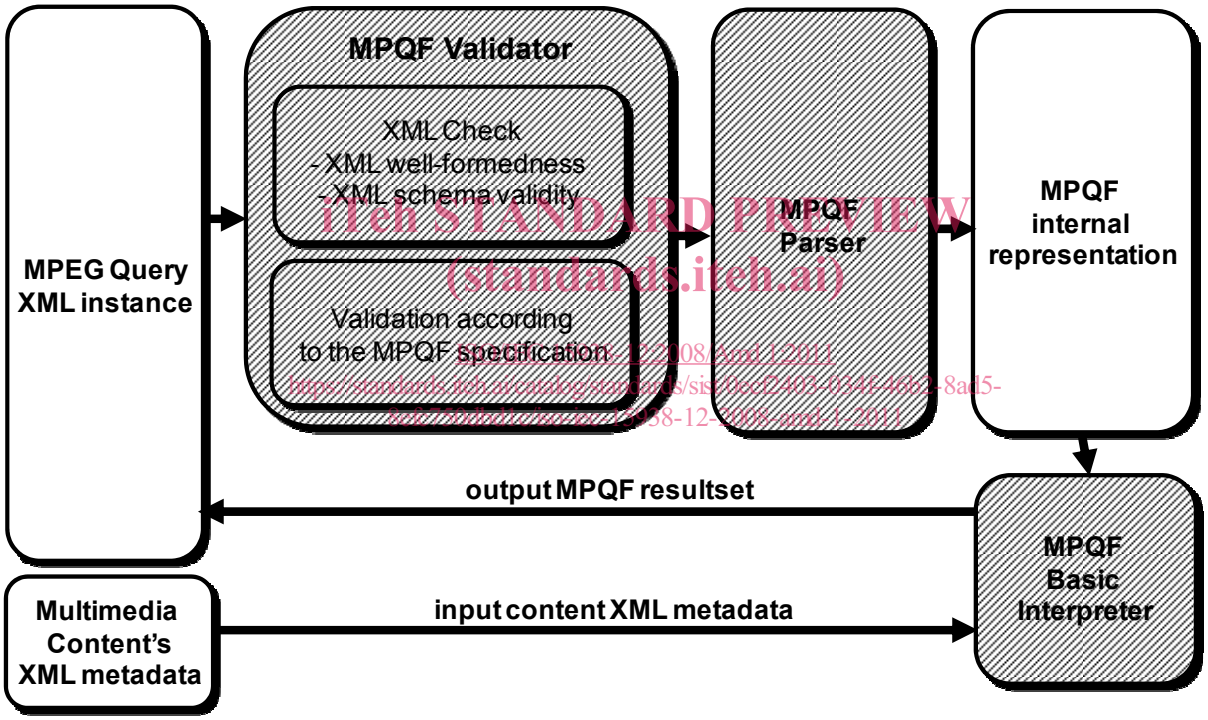

Figure 5 — Reference/utility software architecture

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.

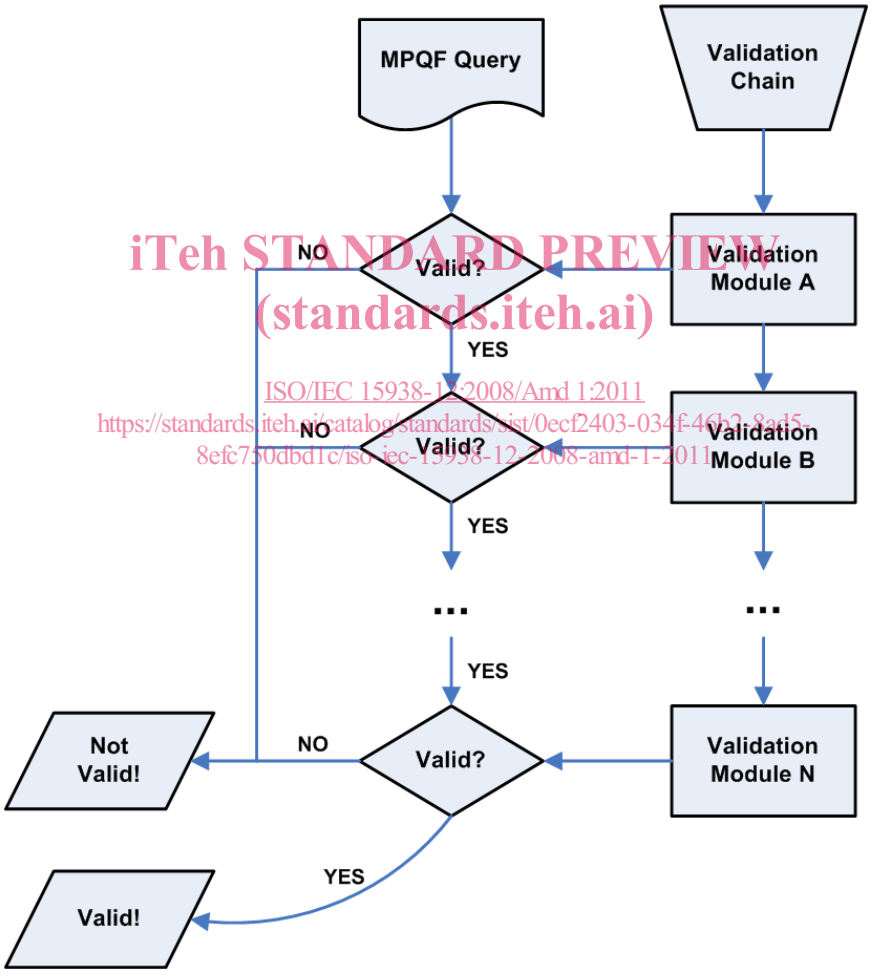

Figure 6 — Workflow of the MPQF validator

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.