INTERNATIONAL STANDARD

ISO/IEC 15938-12

Second edition
2012-11-01

Information technology — Multimedia content description interface —

Part 12:
Query format

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

Partie 12: Format de requête

Reference number
ISO/IEC 15938-12:2012(E)

Contents

Page

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.

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

This second edition cancels and replaces the first edition (ISO/IEC 15938-12:2008), which has been technically revised.
It also incorporates ISO/IEC 15938-12:2008/Cor.1:2009, ISO/IEC 15938-12:2008/Cor.2:2010, ISO/IEC 15938-12:2008:Amd.1:2011 and ISO/IEC 15938-12:2008:Amd.2:2011.

ISO/IEC 15938 consists of the following parts, under the general title *Information technology — Multimedia content description interface*:

— *Part 1: Systems*

— *Part 2: Description definition language*

— *Part 3: Visual*

— *Part 4: Audio*

— *Part 5: Multimedia description schemes*

— *Part 6: Reference software*

— *Part 7:* Conformance testing

— *Part 8: Extraction and use of MPEG-7 descriptions*

— *Part 9: Profiles and levels*

— *Part 10: Schema definition*

— *Part 11: MPEG-7 Profile schemas*

— *Part 12: Query format*

Introduction

The MPEG-7 standard, also known as the "Multimedia Content Description Interface", aims at providing standardized core technologies allowing the description of audiovisual data content in multimedia environments. This is a challenging task given the broad spectrum of requirements and targeted multimedia applications, and the broad number of audiovisual features of importance in such a context. In order to achieve this broad goal, MPEG-7 standardizes:

- Datatypes that are description elements not specific to the audiovisual domain that corresponds to reusable basic types or structures employed by multiple Descriptors and Description Schemes.
- Descriptors (D) to represent Features. Descriptors define the syntax and the semantics of each feature representation. A Feature is a distinctive characteristic of the data, which signifies something to somebody. It is possible to have several descriptors representing a single feature, i.e. to address different relevant requirements. A Descriptor does not participate in many-to-one relationships with other description elements.
- Description Schemes (DS) to specify the structure and semantics of the relationships between their components, which may be both Ds and DSs. A Description Scheme shall have descriptive information and may participate in many-to-one relationships with other description elements.
- A Description Definition Language (DDL) to allow the creation of new DSs and, possibly, Ds and to allow the extension and modification of existing DSs.
- Systems tools to support multiplexing of descriptions or description and data, synchronization issues, transmission mechanisms, file format, etc.

The standard is subdivided into twelve parts:

1. Systems: Architecture of the standard, tools that are needed to prepare MPEG-7 Descriptions for efficient transport and storage, and to allow synchronization between content and descriptions. Also tools related to managing and protecting intellectual property.
2. Description Definition Language: Language for defining new DSs and eventually also new Ds, binary representation of DDL expressions.
3. Visual: Visual description tools (Ds and DSs).
4. Audio: Audio description tools (Ds and DSs).
5. Multimedia Description Schemes: Description tools (Ds and DSs) that are generic, i.e. neither purely visual nor purely audio.
6. Reference Software: Software implementation of relevant parts of the MPEG-7 Standard.
7. Conformance: Guidelines and procedures for testing conformance of MPEG-7 implementations.
8. Extraction and use of MPEG-7 descriptions.
9. Profiles and Levels.
10. Schema Definition.

11. MPEG-7 Profile Schemas.

12. Query Format.

This part of ISO/IEC 15938 contains the tools of the MPEG Query Format (MPQF). It addresses the normative aspects of the MPQF and also illustrates some non-normative examples. The syntax of the Query Format is defined using the guidelines of DDL ISO/IEC 15938-2.

Information technology — Multimedia content description interface —

Part 12: Query format

1 Scope

1.1 Organization of the document

This part of ISO/IEC 15938 describes the query format tools which may be used independently or in combination with other parts of ISO/IEC 15938. Each query format tool is described in two normative sections:

- Syntax: Normative specification of the query and management format.
- Semantic: Normative definition of the semantics of all the components of the corresponding query format specification.

In some instances the query format level tool is also described using either one or two informative sections:

- Examples: Optionally an informative section dealing with examples is appended.
- Definitions: Optionally an informative section dealing with definitions is appended.

1.2 Overview of the Query Format

The query format provides a standardized interface for multimedia content information retrieval systems (e.g. MPEG-7 databases) in three aspects which are input query format, output query format, and query managements. The input query format specifies the interface through which the users can describe their search criteria with a set of precise input parameters in addition to a set of preferred output parameters to depict the return result sets. The output query format specifies the interface format for the result set. The query management provides means for selecting services (e.g. MPEG-7 database) or aggregated services (e.g. service provider that administers a set of different services) based on service properties (e.g. supported query format).

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C Recommendation, 23 January 2007. http://www.w3.org/TR/xpath-datamodel/

XML Path Language (XPath) 2.0. W3C Recommendation, 23 January 2007. http://www.w3.org/TR/xpath20/

3 Terms, definitions, abbreviated terms and conventions

For the purposes of this document, the following terms and definitions, abbreviated terms and conventions apply.

3.1 Terms and definitions

3.1.1
content
data and the associated **metadata**

3.1.2
evaluation item
EI
unit, against which the query condition is tested

NOTE By default, an EI is a **multimedia content** of the multimedia repository, but other types of EI are also possible. An EI can be:

1) A **multimedia content**;

2) A **segment** of a **multimedia resource**;

3) An **XPath-item** related to the **multimedia content's metadata** XML tree.

3.1.3
input query format
interface format going from a requester to one or more responders with two functionalities

— functionality providing a combination of syntax and semantics of the interface, through which the requester assigns search criteria and associated data;

— functionality providing syntax and semantics of the interface, through which the requester specifies the format of the result data

NOTE The second functionality of the input query format provides tools by which the requester can express desired output format that should be conformant to the specification of **output query format**.

3.1.4
metadata
data expressed as a schema valid XML instance to carry additional information describing a **multimedia resource**, where the schema defines the information model of the data

3.1.5
multimedia content
coded representation of the information contained in or related to a **multimedia resource** in a formalized manner suitable for interpretation by human means

3.1.6
multimedia resource
URI identifiable portion of raw data of a video, an image, an audio or text in any format, that is associated with a MIME Content-Type

3.1.7
output query format
interface format going from the responder to the requester as a response to the request specified by the **input query format**

NOTE Output query format defines all possible structures of return from responder to the requester. The structure of an actual return shall be decided by OutputDescription element in **input query format**.

3.1.8
query management tools
tools to support the functionality required to manage the query transaction between the requesters and the responders

NOTE The query management tools do not include tools that are supported by network protocols. The query management tools intend to be network agnostic and media agnostic.

3.1.9
segment
spatial, temporal, or spatio-temporal unit of multimedia, for example, a temporal segment of video, or a spatial segment of an image

3.1.10
XPath-item
either a node from the **multimedia content's metadata** XML tree or an atomic value

NOTE Details about the different types of nodes and atomic values can be found in the W3C Recommendation on XQuery 1.0 and XPath 2.0 Data Model. An XPath-item of a **multimedia content's metadata** may or may not be related to a **multimedia content's segment**. Also, a **multimedia content** or a **multimedia content's segment** may or may not be related to XML **metadata**. Within MPQF queries, XPath can be used to select a sequence of **multimedia content's segments** and/or **metadata** XPath-items. According to the W3C Recommendation on XQuery 1.0 and XPath 2.0 Data Model, a sequence is an ordered collection of zero or more XPath-items.

3.2 Abbreviated terms

MPEG:	Moving Picture Experts Group
MPEG-7:	ISO/IEC 15938
MPQF:	MPEG Query Format, ISO/IEC 15938-12
URI:	Uniform Resource Identifier (IETF Standard is RFC 2396)
URL:	Uniform Resource Locator (IETF Standard is RFC 2396)
XML:	Extensible Markup Language (W3C, http://www.w3.org/XML/)
RDF:	Resource Description Framework (W3C, http://www.w3.org/RDF/)
SPARQL:	SPARQL Query Language for RDF. W3C Recommendation 15 January 2008. http://www.w3.org/TR/rdf-sparql-query/

3.3 Conventions

3.3.1 Query and query management tools

This part of ISO/IEC 15938 specifies the format for query and query management tools using XML-Schema.

- Query – A set of tools supporting the definition of the query request as well as the query response defined for the MPEG query format. The structure defined for the `Query` element provides a container for input query format or output query format.
    - Input query format – `Input` and `FetchResult` elements are defined for the query. The input query format structure provides a container for a query request. Such a request should contain a set of conditions and/or the output description which specifies the structure and content of the output query format and/or a set of declarations.

  - Output query format – The `Output` element is defined for the query. It provides a container for all the responses from a responder to a requester. It may contain not only query results but also any messages such as error and exceptions.

- Management tools – A set of tools for the query management defined for the MPEG query format including service discovery, querying service capability, and service capability description. The structure defined for the management tools provides a container for input management tools or the output management tools.

  - Input management tools – An `Input` element is defined for the management tools intended to be sent from a requester to one or more responders.

  - Output management tools – An `Output` element is defined for the management tools intended to be sent from a responder to one or more requesters.

3.3.2 Naming convention

In order to specify tools for the query format, this part of ISO/IEC 15938 uses constructs provided by the language specified in ISO/IEC 15938-2 [1], such as "element", "attribute", "simpleType" and "complexType". The names associated to these constructs are created on the basis of the following conventions:

- If the name is composed of various words, the first letter of each word is capitalized. The rule for the capitalization of the first word depends on the type of construct and is described below.

- Element naming: the first letter of the first word is capitalized (e.g. TimePoint element of TimeType).

- Attribute naming: the first letter of the first word is not capitalized (e.g. timeUnit attribute of IncrDurationType).

- complexType naming: the first letter of the first word is capitalized, the suffix "Type" is used at the end of the name, except the concrete types inherited from abstract types.

- simpleType naming: the first letter of the first word is not capitalized, the suffix "Type" may be used at the end of the name (e.g. timePointType).

NOTE The full name of the complexType or simpleType is used when referencing a complexType or simpleType in the definition of the query format.

3.3.3 Documentation convention

The syntax of each datatype, descriptor and description scheme is specified using the constructs provided by ISO/IEC 15938-2 [1], and is shown in this part of ISO/IEC 15938 using a specific font and background:

```
<complexType name="ExampleType">
   <sequence>
      <element name="Element1" type="string" minOccurs="1" maxOccurs="1"/>
   </sequence>
   <attribute name="attribute1" type="string" use="default" value="attrvalue1"/>
</complexType>
```

The semantics of each datatype, descriptor and description scheme is specified using a table format, where each row contains the name and a definition of a type, element or attribute:

Name	*Definition*
ExampleType	Specifies an ...
element1	Describes the …
attribute1	Describes the …

Non-normative examples are included in separate sections, and are shown in this part of ISO/IEC 15938 using a separate font and background:

```
<Example attribute1="example attribute value">
   <Element1>example element content</Element1>
</Example>
```

Moreover, the schema defined in this part of ISO/IEC 15938 follows a type-centric approach. As a result, almost no elements (in the XML schema sense) are defined. Most of the description tools are specified only by defining a complexType or a simpleType. In order to create a description, it has to be assumed that an element of a given type (complexType or simpleType) has been declared somewhere in the schema, for example as a member of another complexType or simpleType.

The examples in the informative sections assume that the following declaration has been made:

```
<element name="Example" type="mpqf:MyToolType">
```

Therefore, the example shown above is a valid description.

3.3.4 Wrapper convention of the schema

The Syntax defined in this part of ISO/IEC 15938 assumes the following schema wrapper.

```
<schema xmlns:mpqf="urn:mpeg:mpqf:schema:2008"
xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:mpeg:mpqf:schema:2008" elementFormDefault="qualified"
attributeFormDefault="unqualified">
```

4 Structure and Data Model

4.1 Structure

The Query Format defines the interface between a requester and a responder. The requester may be any requester and the according responder might be the service provider, which can in turn additionally be a requester to a number of other databases. Additionally, one or more service providers can utilize the Query Format acting as responder from the requester side and as requester, when forwarding the messages to a number of databases. Furthermore, the service provider may have the capability to aggregate the results from different databases and to reply a combined result to the requester. Figure 1 depicts a possible setup for the use of the MPEG Query Format.

The part of the query format related to the messages from the requester to the responder, describing their search criteria, is called Input Query Format; the part of the query format related to the messages in return is called Output Query Format. Additionally, the format for the management messages provides means for selecting services (e.g., MPEG-7 database) or aggregated services (e.g., service provider that administers a set of different services) based on service properties (e.g., supported query format, etc.).

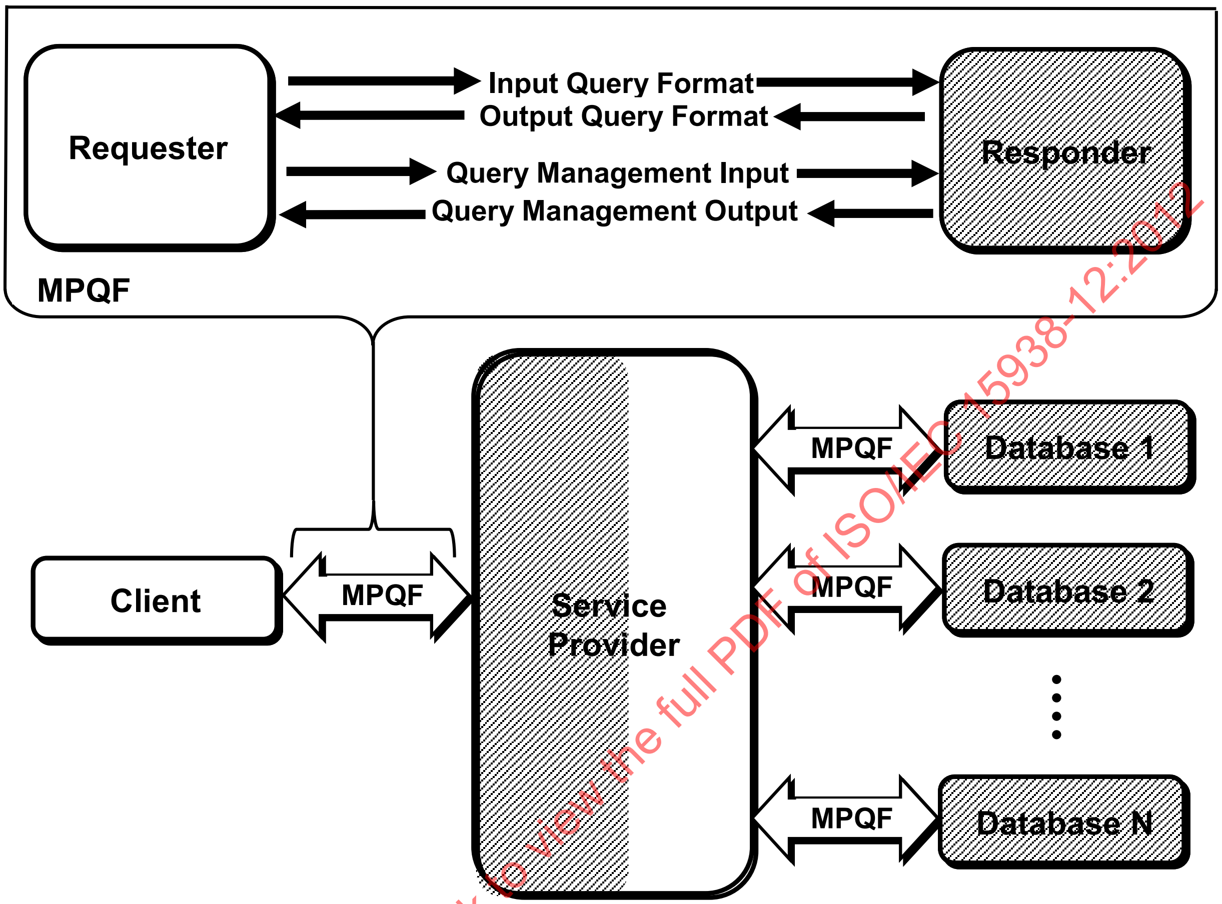


Figure 1 — Possible scenario for the use of the MPEG Query Format

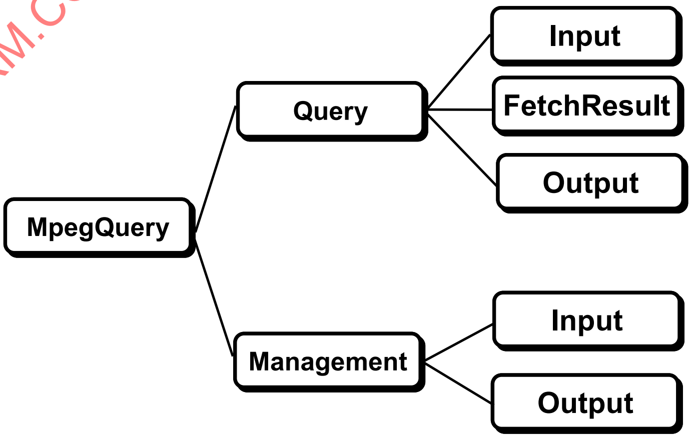


Figure 2 — Schema overview of the uppermost elements of the MPEG Query Format

The root element of the MPEG Query Format is named “MpegQuery” and it contains a sequence of `Query` and `Management` elements. Table 1 explains both elements in more detail:

Table 1 — Query and Management elements

`Query`	A set of tools are defined for the query and response for the MPEG Query Format. The structure defined for the `Query` element provides a container for Input Query Format or Output Query Format.
`Management`	A set of tools for the query management defined for the MPEG Query Format including service discovery, querying service capability, and service capability description. The structure defined for the `Management` element provides a container for Management Input Tools or the Management Output Tools.

The `Query` element of the MPEG Query Format defines a set of tools for the query and its response. Table 2 depicts the containing elements in more detail. Note that the `Input` and the `FetchResult` elements belong to the Input Query Format, which provides a container for describing a query request (e.g., the query condition and/or the output description which specifies the structure and content of the output query format).

Table 2 — Elements in Query Tools

`Input`	The `Input` element is part of the Input Query Format. Its structure provides a container for describing a query request. Such a request can consist of a query condition and/or the output description which specifies the structure and content of the output query format and/or some declaration parts.
`FetchResult`	The `FetchResult` element is also part of the Input Query Format. It allows the user to request the results of a previous query issued using e.g., the asynchronous mode.
`Output`	The `Output` element describes the Output Query Format. It provides a container for all the responses from a responder to a requester. It may contain not only query results but also any messages such as errors, exceptions or comments.

The `Management` element of the MPEG Query Format represents the Query Management Tools of the Query Format. It describes a set of tools for the query management including service discovery, querying service capability, and service capability description. Similar to the query tools (Input Query Format and Output Query Format), the management part distinguishes the tools for the request (Input Management Type) and the response (Output Management Type). Table 3 depicts the containing elements in more detail.

Table 3 — Elements in Query Management Tools

`Input`	The `Input` element, which is defined for the management tools, is intended to be sent from a requester to one or more responders.
`Output`	The `Output` element, which is defined for the management tools, is intended to be sent from a responder to one requester.

4.2 Data Model

Processing and evaluation of an MPQF query is executed against one or more multimedia repositories. Note, all introduced terms used within this subclause are explained in 3.1.

From the point-of-view of the MPQF, a multimedia repository is an unordered set of multimedia contents (MCs). An MPQF query specifies a search condition tree composed by conditions which may be interconnected by Boolean expressions. Conditions within the condition tree operate on one evaluation-item (EI) at a given time, or two EIs if a Join operation is used. By default, an evaluation-item (EI) is a multimedia content in the multimedia repository, but other types of EIs are also possible. An EI can be either a multimedia content (MC), a segment of a multimedia resource, or an XPath-item related to the MC's metadata XML tree. A segment of a multimedia content is a spatial and/or temporal unit of multimedia resource, for example, a temporal segment of a video, or a segment of an image.

According to the XQuery 1.0 and XPath 2.0 Data Model (XDM), an XPath-item is either a node from the MC's metadata XML tree or an atomic value. Details about the different types of nodes and atomic values can be found in the W3C Recommendation of the XQuery 1.0 and XPath 2.0 Data Model (XDM). An XPath-item of an MC's metadata may or may not be related to an MC's segment. A multimedia content or an MC's segment may or may not be related to XML metadata. Within MPQF queries, XPath can be used to select a sequence of MC's segments and/or metadata XPath-items.

The scope of query evaluation and the granularity of the result set shall be determined by the `EvaluationPath` element specified in the `QueryCondition` element. For example, if the `EvaluationPath` element points to a certain level of a video segment, then the result set is a collection of video segments with the requested metadata. If the `EvaluationPath` element points to a video resource, then the result set consists of a collection of video resources with the requested metadata. If the `EvaluationPath` element in the `QueryCondition` element is not specified, the output result is provided as a collection of multimedia contents, as stored in the repository, each satisfying the query condition.

XPath expressions appearing in MPQF can be absolute or relative. When absolute, independently of the kind of evaluation item being addressed, an XPath expression will always refer to the root of the metadata document related to the multimedia content which the evaluation-item belongs to (even if the evaluation-item is related just to a fragment of this document).

When relative, an XPath expression will always refer to the root node of the metadata fragment (which can be the root of the metadata document or cannot) related to the evaluation item being addressed.

5 Root Element

5.1 Introduction

The `MpegQuery` element serves as the root element of the MPEG Query Format. The root element shall be used as the topmost element in all messages transmitted. This applies on the one side to the input query format and the output query format of a query request/response as well as on the other side to the query management messages of the input/output.

5.2 Syntax

```
<element name="MpegQuery" type="mpqf:MpegQueryType"/>

<complexType name="MpegQueryType">
   <choice>
      <element name="Query">
         <complexType>
            <choice>
               <element name="Input" type="mpqf:InputQueryType"/>
               <element name="FetchResult">
                  <complexType>
                     <attribute name="queryID" type="anyURI"/>
                     <attribute name="retrievePageNum" type="positiveInteger"/>
                  </complexType>
               </element>
               <element name="Output" type="mpqf:OutputQueryType"/>
            </choice>
         </complexType>
      </element>
      <element name="Management">
         <complexType>
            <choice>
               <element name="Input" type="mpqf:InputManagementType"/>
               <element name="Output" type="mpqf:OutputManagementType"/>
            </choice>
         </complexType>
      </element>
   </choice>
   <attribute name="mpqfID" type="anyURI" use="required"/>
</complexType>
```

5.3 Semantics

Semantics of the `MpegQueryType` type:

Name	*Definition*
MpegQuery	Serves as the root element of the MPEG Query Format. The MpegQuery element shall be used as the topmost element in any instance of MPEG Query Format.
MpegQueryType	Specifies the syntax of the root element. Within this element, either one of the Input, FetchResult or Output element of the Query or one of the Input or Output element of the Management element shall be instantiated, respectively.
Query	Wraps the user query request (an Input or a FetchResult element) or the responder response to a user query request (the Output element).
Input	Wraps the user query specified using the Input Query Format, which is to be sent from the requester to the responder.
FetchResult	Allows the user to request the result of a previous query. In case of a query request issued using the asynchronous mode (with the immediateResponse attribute of the InputQueryType type set to false), the FetchResult element shall be used to indicate the preparedness of the user to receive results. In case of a query request issued using the synchronous mode, the FetchResult element shall be used to receive individual pages of the result set (if paging is activated).
queryID	The requester is looking for a the result of a certain identifier in asynchronous mode. The attribute queryID specifies this identifier.
retrievePageNum	Specifies the number of page the requester wants the responder to return (optional) The default value is '1.'
Output	Wraps the responder response to a user query as described using the Output Query Format. The output is sent from the responder to the requester.
Management	Describes the Query Management tools which may be exchanged between responders and requesters. Within this element only one of Input or Output element shall be instantiated.
Input	Wraps a user request for service capabilities.
Output	Wraps the response to a user request for service capabilities.
mpqfID	Specifies a unique identifier which is assigned to every message sent between a requester and a responder.

5.4 Example

The following examples show the use of the root element for the MPEG Query Format. The first three examples demonstrate the use of the `Query` element. First, a user query request is simulated by using the `Input` element. Note, that no details have been specified. Second, the result of a previous asynchronous request is going to be fetched by the user. Finally, the result set of a query request is demonstrated by using the `Output` element.

The last example shows the use of a service discovery request by using the `Management` element followed by an `Input` element. In this case the response would contain a list of all available services which are registered.

```
<MpegQuery mpqfID="idForMyRequest1">
  <Query>
    <Input>
      . . .
    </Input>
  </Query>
</MpegQuery>
```

```
<MpegQuery mpqfID="idForMyRequest2">
  <Query>
    <FetchResult queryID="idFromAsyncRequest1"/>
  </Query>
</MpegQuery>
```

```
<MpegQuery mpqfID="idForMyResponse1">
  <Query>
    <Output>
      . . .
    </Output>
  </Query>
</MpegQuery>
```

```
<MpegQuery mpqfID="idForMyRequest3">
  <Management>
    <Input>
      . . .
    </Input>
  </Management>
</MpegQuery>
```

6 Datatypes

6.1 MediaLocatorType

6.1.1 Introduction

The `MediaLocatorType` type serves as a basic datatype for locating a multimedia resource in the `MediaResource` element.

The `MediaLocatorType` type shall be composed of either a `MediaUri` element or `InlineMedia` element for locating a multimedia resource. The `InlineMedia` element enables an alternative method to a `MediaUri` for locating multimedia resource, by embedding the multimedia resource into the description.

6.1.2 Syntax

```
<complexType name="MediaLocatorType">
     <sequence>
         <choice>
             <element name="MediaUri" type="anyURI"/>
             <element name="InlineMedia" type="mpqf:InlineMediaType"/>
         </choice>
     </sequence>
 </complexType>
 <complexType name="InlineMediaType">
     <choice>
         <element name="MediaData16" type="hexBinary"/>
         <element name="MediaData64" type="base64Binary"/>
     </choice>
     <attribute name="type" type="mpqf:mimeType" use="required"/>
 </complexType>
```

6.1.3 Semantics

Semantics of the `MediaLocatorType` type:

Name	*Definition*
`MediaLocatorType`	Specifies the `MediaLocatorType` type. Describes the location of a multimedia resource in `MediaResource` element. The location of a multimedia resource should be specified either by reference to the external multimedia resource using the `MediaUri` element or by embedding the multimedia resource within the description as an `InlineMedia` instance.
`MediaUri`	Points to the actual resource that is used for querying.
`InlineMedia`	Describes the actual resource that is used for querying.

Semantics of the `InlineMediaType` type:

Name	*Definition*
`InlineMediaType`	Specifies the multimedia resource embedded in the resource. The `InlineMediaType` type is a container for the actual binary multimedia resource.
`MediaData16`	Describes the actual resource that is used for querying, encoded as a textual string in base-16 format.
`MediaData64`	Describes the actual resource that is used for querying encoded as a textual string in base-64 format.
`Type`	This attribute specifies the `mimeType` of the multimedia resource.

6.2 MimeType

6.2.1 Introduction

The `mimeType` type is defined to specify the syntax for a `mimeType` element which shall be used to identify the media type of a multimedia resource.

6.2.2 Syntax

```
<simpleType name="mimeType">
  <restriction base="string">
    <whiteSpace value="collapse"/>
    <pattern value='([Xx]\-)?[!#$%&#x27;*+.0-9A-Z\^-~&#x7f;]+/([Xx]\-
                    )?[!#$%&#x27;*+.0-9A-Z\^-~&#x7f;\-]+'/>
  </restriction>
</simpleType>
```

6.2.3 Semantics

Semantics of the `mimeType` type:

Name	*Definition*
`mimeType`	Specifies the syntax of the `mimeType` type, which is used to identify the type of multimedia resource.
`pattern value`	Specifies the allowed pattern for `mimeTypes` and corresponds to RFC 2045 MIME (Part 1).

6.3 XPathType

6.3.1 Introduction

XPath expressions are used to identify elements and attributes in the corresponding responder description (specified by the namespace of the description, e.g., MPEG-7) in Input Query or Output Query. The instance of `xPathType` shall be compliant to the specification XML Path Language (XPath) 2.0, W3C Recommendation 23 January 2007.

6.3.2 Syntax

```
<simpleType name="xPathType">
  <restriction base="token"/>
</simpleType>
```

6.3.3 Semantics

Semantics of the `xPathType` type:

Name	*Definition*
`xPathType`	Describes the valid `XPath` expression.

6.4 ZeroToOneType

6.4.1 Introduction

The `zeroToOne` datatype represents a `float` value in the range of 0.0 to 1.0.

6.4.2 Syntax

```
<simpleType name="zeroToOneType">
  <restriction base="float">
    <minInclusive value="0.0"/>
    <maxInclusive value="1.0"/>
  </restriction>
</simpleType>
```

6.4.3 Semantics

Semantics of the `zeroToOneType` type:

Name	*Definition*
`zeroToOneType`	Describes a floating value in the range of 0.0 to 1.0 inclusive.

6.5 TermType

6.5.1 Introduction

The `TermType` type is to be used in the classification scheme and other tools to control the use of the terms inside the query format. The `Name` element allows the specification of the label of the term. The `Description` element allows the definition and the description in text format. The `href` attribute is required to point to the classification scheme, in which the terms are defined. A hierarchy of nested terms can be expressed by successive applying of the `Term` element.

6.5.2 Syntax

```
<complexType name="TermType">
  <sequence>
    <element name="Name" type="string" minOccurs="0"/>
    <element name="Description" type="string" minOccurs="0"/>
    <element name="Term" type="mpqf:TermType" minOccurs="0"
         maxOccurs="unbounded"/>
  </sequence>
  <attribute name="href" type="mpqf:SimpleTermType" use="required"/>
</complexType>

<simpleType name="SimpleTermType">
  <restriction base="anyURI"/>
</simpleType>
```

6.5.3 Semantics

Semantics of the TermType type:

Name	*Definition*
TermType	Specifies a TermType type which is used for describing capabilities and its references to classification schemes.
Name	Specifies the name of the term.
Description	Specifies a detailed description of the term.
Term	Specifies a new term which supports nested term definitions.
href	Specifies the SimpleTermType type pointing to a specific term of a classification scheme.
SimpleTermType	Specifies the URN of the term.

6.5.4 Example

For the examples, check the classification schemes in Annex B.

6.6 ClassificationScheme

6.6.1 Introduction

The ClassificationScheme element is an element, which contains the classification scheme. It makes use of the TermType type, which allows a nested definition of terms.

6.6.2 Syntax

```
<element name="ClassificationScheme">
  <complexType>
    <sequence>
      <element name="Term" type="mpqf:TermType" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="uri" type="anyURI" use="required"/>
    <attribute name="domain" use="optional">
      <simpleType>
        <list itemType="mpqf:xPathType"/>
      </simpleType>
    </attribute>
  </complexType>
</element>
```

6.6.3 Semantics

Semantics of the `ClassificationScheme` element:

Name	*Definition*
`ClassificationScheme`	This element contains the actual classification scheme
`Term`	Describes the content of the classification scheme (the single terms including a description).
`Uri`	Identifies the classification scheme with a Uniform Resource Identifier (URI). This URI shall uniquely identify a classification scheme. The use of a URN as a classification scheme URI is preferred.
`Domain`	Indicates a set of description elements to which this `ClassificationScheme` applies. The set is specified using an XPath expression. This attribute is a hint on how the `ClassificationScheme` can be used in a description. One or more values may be specified. The XPath in this attribute is defined relative to a *description root*, not the DDL schema. For example: a value of ".//Classification/Genre" for domain indicates that the `ClassificationScheme` can be used for "Genre“ elements inside of all "Classification“ elements in the description.

6.6.4 Example

See Annex B for classification scheme examples.

6.7 RelationType

6.7.1 Introduction

The RelationType type describes a relation among DS instances and controlled terms. The arguments of a relation are categorized into the source (first) and the target (second) arguments. Both the source and target arguments for a relation are ordered. A relation may have a variable number of arguments in each of the source and the target.

6.7.2 Syntax

```
<complexType name="RelationType">
  <sequence/>
  <attribute name="relationType" type="anyURI" use="required"/>
  <attribute name="sourceResource" type="IDREF" use="required"/>
  <attribute name="targetResource" type="IDREF" use="optional"/>
</complexType>
```

6.7.3 Semantics

Semantics of the `RelationType` type:

Name	*Definition*
`RelationType`	Specifies the `RelationType` type of a spatial or temporal relation.
`relationType`	Describes the type of the Relation. Depending on the used QueryType (`SpatialQueryType`, `TemporalQueryType`) the allowed classification scheme types differ. The allowed classification schemes are defined at the corresponding query types
`sourceResource`	Specifies the source element of the specific relation. The source element shall be defined in the declaration part of type `DescriptionResourceType` type and point to a valid description according to the used query type (e.g., for a `SpatialQueryType` type only a MPEG-7 StillImage or comparable (of other metadata standards) description is allowed)
`targetResource`	Specifies the target element of the specific relation. The target element shall be defined in the declaration part of type `DescriptionResourceType` type and point to a valid description according to the used query type (e.g., for a `SpatialQueryType` type only a MPEG-7 StillImage or comparable (of other metadata standards) description is allowed)

6.7.4 Example

See the `SpatialQuery` and `TemporalQuery` type for the examples.

6.8 SemanticFieldType

The SemanticFieldType supports the addressing of individual Subject/Object parts of an RDF triple which are involved in a comparison expression or the sorting operation.

6.8.1 Syntax

```
<complexType name="SemanticFieldType">
  <sequence>
    <element name="Var" type="string"/>
  </sequence>
</complexType>
```

6.8.2 Semantics

Semantics of the `SemanticFieldType`:

Name	*Definition*
`SemanticFieldType`	Specifies the representation of a semantic field within the query format. A semantic field addresses a node (subject/object) according to the RDF data model (triple). The

Name	*Definition*
	syntax is similar to the SPARQL query language and starts with a question mark and a descriptive identifier or a specific URI for the variable.
`Var`	Defines a place holder for the subject of a RDF triple. The place holder must start with a question mark and a descriptive identifier or a specific URI.

6.8.3 Example

The main purpose of the SemanticFieldType (the variable element) is the addressing of RDF nodes to be used in sorting and comparison expressions. For instance, one want to have the result set sorted after a specific node information or want to filter the result set according to specific literal values. An example might be the price of a hotel which is modeled like hotel->hasPrice->value (e.g., Bloom->hasPrice->80 which results to a requesting triple: ?hotel -> hasPrice -> ?price), then one want to filter all hotels whose price is above a certain threshold. In this case, the GreaterThan condition (comparison expression already defined in MPQF) is used to filter the identified literal to a specific bound.

```
<MpegQuery mpqfID="">
   <Query>
      <Input>
         <QueryCondition>
            <Condition xsi:type="AND">
               <Condition xsi:type="SemanticRelation"
                         anchor="true" anchorDistance="0">
                  <Subject>?hotel</Subject>
                  <Property>hasPrice</Property>
                  <Object>?price</Object>
               </Condition>
               <Condition xsi:type="GreaterThan">
                  <SemanticArithmeticField>
                     <Var>?price</Var>
                  </SemanticArithmeticField>
                  <DoubleValue>100.0</DoubleValue>
               </Condition>
            </Condition>
         </QueryCondition>
      </Input>
   </Query>
</MpegQuery>
```

7 Input Query Format

7.1 Introduction

This clause specifies the `Input` element defined for the Input Query Format. The Input Query Format structure is defined by the `InputQueryType` type. It provides a container for describing a query request containing a set of conditions and/or the specification of the structure and content of the output query format and a declaration part. In general, this type provides all necessary constructs in order to express a multimedia query request to a set of databases.

7.2 Syntax

```
<complexType name="InputQueryType">
  <sequence>
    <element name="QFDeclaration" type="mpqf:QFDeclarationType" minOccurs="0"/>
    <element name="OutputDescription" type="mpqf:OutputDescriptionType"
minOccurs="0"/>
    <element name="QueryCondition" type="mpqf:QueryConditionType" minOccurs="0"/>
    <element name="ServiceSelection" type="mpqf:ServiceSelectionType"
          minOccurs="0"/>
  </sequence>
  <attribute name="previousAnswerID" type="anyURI" use="optional"/>
  <attribute name="immediateResponse" type="boolean" use="optional"
default="true"/>
  <attribute name="timeout" type="duration" use="optional"/>
</complexType>
<complexType name="ServiceSelectionType">
  <sequence>
    <element name="ServiceID" type="anyURI" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

7.3 Semantics

Semantics of the `InputQueryType` type:

Name	*Definition*
InputQueryType	Specifies the syntax of the Input element, which describes the input query request to be sent from the requester to the responder.
QFDeclaration	Defines resources that can be referenced for describing query conditions and output data fields in the QueryCondition or OutputDescription element, respectively. A resource is either a multimedia resource or a description resource. A multimedia resource is specified either by the inlined raw data of an image, video, etc. or a pointer (URL) to its location. The description resource contains information of low level (e.g., color) or high level (e.g, objects) features of an example media. The description can be given by any valid XML-based multimedia metadata schema. Furthermore, one has the possibility to declare specific fields (e.g., MPEG-7 Creator element) of any XML-based multimedia metadata schema in order to reuse this information within the OutputDescription element or the QueryCondition element.
OutputDescription	Defines the content as well as the structure of the returned value from the responder. Here, the requester has the possibility to select the desired information which the result set should contain. For instance, in case of searching for songs, one would like to know the song title, location, etc. Depending on the used multimedia metadata standard (of the target service) the XML elements can vary. Furthermore, by using absolute path entries, the requester has the possibility to force the structure of the returned value.
QueryCondition	Defines the search and filter criteria within a single query.
ServiceSelection	Specifies the set of services where the query should be evaluated. This element is supposed to be used in case of communicating with an aggregation service in order to signalize which subset of the available services should evaluate the request. The syntax is specified by the ServiceSelectionType type.

Name	*Definition*
previousAnswerID	Specifies a unique identifier to determine a query response already received (optional). This attribute is intended to be used in operations (e.g., relevance feedback) such as a search within the previous result set.
immediateResponse	Defines the mode of operation. The default value of this optional attribute is “true”. When either the value is “true” or the attribute is not present, the requester expects to receive the response immediately (until timeOut). If the value of this attribute is “false”, the requester is expected to receive only a “mpqfID” as response (asynchronous mode). In the asynchronous mode, the requester will come later (after the timeOut) to harvest the response (sending a message including the FetchResult element). In case of a given value for the expirationDate attribute, the result set is required to be cached. In case of no expirationDate attribute is given, it is an implementation issue.
timeOut	Specifies the temporal duration during which the responder can perform searching the relevant databases. In the immediate response mode of operation, it can be interpreted as a time limit within which an aggregation responder can wait for the individual service provider’s response. In the asynchronous mode of operation, it can be interpreted as a way of specifying a time point, after which the requester intends to issue a FetchResult query.

Semantics of the `ServiceSelectionType` type:

Name	*Definition*
`ServiceSelectionType`	Specifies the syntax of the `ServiceSelection` element, which determines the set of services where the request should be sent to.
`ServiceID`	Defines one service by its ID of type `anyURI`. This information is the same as given by the service’s capability description and denotes the service’s entry point.

7.4 Example

Examples for the `Input` element are given in the following sub-clauses of `QFDeclaration`, `OutputDescription` and `QueryCondition`. An example for a service selection is presented below. The `ServiceSelection` element in the `input` element determines the set of databases that have been chosen for retrieval. The use of the `ServiceSelection` element assumes the existence of an aggregation service which takes care of on the one side distributing the query to the selected target databases and on the other side aggregating the individual result sets in a sophisticated manner. In case of a direct query request to a single database, the service selection element may be omitted.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <ServiceSelection>
        <ServiceID>http://www.test-retrieval.com</ServiceID>
        <ServiceID>http://www.test-retrieval1.com</ServiceID>
      </ServiceSelection>
    </Input>
  </Query>
</MpegQuery>
```

8 QFDeclaration

8.1 Introduction

The optional `QFDeclaration` element allows declaring reusable definitions of data paths and resources that can be referenced multiple times within a query. Its structure is defined by the `QFDeclarationType` type, which consists of a sequence of `DeclaredField` and `Resource` elements and Prefix information declaring used namespaces. A `DeclaredField` element allows declaring a reusable data path within an item's metadata. A `Resource` element allows declaring a reusable resource description using one of the subtypes of the `ResourceType` (the `MediaResourceType` type or the `DescriptionResourceType` type). A `Prefix` element allows declaring a shortcut for a namespace which is used in semantic expressions.

8.2 Syntax

```
<complexType name="QFDeclarationType">
  <sequence>
    <element name="DeclaredField" type="mpqf:DeclaredFieldType" minOccurs="0"
maxOccurs="unbounded"/>
    <element name="Resource" type="mpqf:ResourceType" minOccurs="0"
maxOccurs="unbounded"/>
    <element name="Prefix" minOccurs="0" maxOccurs="unbounded">
      <complexType>
          <attribute name="name" type="string" use="required" />
          <attribute name="uri" type="anyURI" use="required" />
      </complexType>
    </element>
  </sequence>
  <!-- Declaration of entities that can be reused in OutputDescription or
QueryCondition -->
</complexType>

<complexType name="DeclaredFieldType">
  <simpleContent>
    <extension base="mpqf:xPathType">
      <attribute name="id" type="ID" use="required"/>
      <attribute name="typeName" type="string" use="optional"/>
    </extension>
  </simpleContent>
</complexType>

<complexType name="ResourceType" abstract="true">
  <attribute name="resourceID" type="ID" use="required"/>
</complexType>

<complexType name="MediaResourceType">
  <complexContent>
    <extension base="mpqf:ResourceType">
      <sequence>
        <element name="MediaResource" type="mpqf:MediaLocatorType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="DescriptionResourceType">
  <complexContent>
    <extension base="mpqf:ResourceType">
      <sequence>
        <element name="AnyDescription">
          <complexType mixed="true">
            <sequence>
              <any namespace="##any"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

8.3 Semantics

Semantics of the `QFDeclarationType` type:

Name	*Definition*
`QFDeclarationType`	Specifies the syntax of the `QFDeclaration` element, which allows declaring reusable definitions of data paths and resources that can be referenced multiple times within a query.
`DeclaredField`	Allows declaring a reusable data path within an item's metadata. Its structure is defined by the `DeclardFieldType` type.
`Resource`	Allows declaring a reusable resource description using one of the subtypes of the abstract `ResourceType` type (the `MediaResourceType` type or the `DescriptionResourceType` type).

Semantics of the `Prefix` element:

Name	*Definition*
`Prefix`	Defines the name and the unique URI of a namespace, e.g. *http://www.w3.org/1999/02/22-rdf-syntax-ns#* for RDF.
`Name`	Defines the name or shortcut of the namespace. For instance: *xsd*.
`Uri`	Defines the assigned URI for this prefix and shortcut. For instance: *http://www.w3.org/2001/XMLSchema#*

Semantics of the `DeclaredFieldType` type:

Name	*Definition*
`DeclaredFieldType`	Complex type which allows declaring a reusable data path within an item's metadata the same way as the `FieldType` type does, but adding an ID that can be referenced multiple times within a query. As the `FieldType` type, the `DeclaredFieldType` type is derived by extension from the `xPathType` type which is itself derived from the `token` type. It serves to specify a data path within an item's metadata using an XPath expression. XPath expressions appearing in MPQF can be absolute or relative. When absolute, independently of the kind of evaluation item being addressed (see the Terminology section), an XPath expression will always refer to the root

Name	*Definition*
	of the metadata document related to the multimedia content to which the evaluation item belongs (even if the evaluation-item is related just to a fragment of this document). When relative, an XPath expression will refer to the root node of the metadata fragment (which can be the root of the metadata document or cannot) related to the evaluation item being addressed. If a `typeName` is specified, the XPath expression will refer to all the elements belonging to the named type appearing anywhere within the evaluation item being addressed. In this case only relative XPath expressions are allowed.
`Id`	Specifies a unique identifier for the declared field. The attribute `id` can be used for referencing the field multiple times within the query.
`typeName`	Specifies the name of a complex data type defined in MPEG-7 or any other metadata schema, which will serve as the starting point of a relative XPath expression.

Semantics of the `ResourceType` type:

Name	*Definition*
`ResourceType`	Specifies the root type of all resource types. It is abstract.
`resourceID`	Specifies a unique identifier for the declared resource. The identifier can be used for referencing the resource multiple times within the query.

Semantics of the `MediaResourceType` type:

Name	*Definition*
`MediaResourceType`	Subtype of the `ResourceType` type which allows describing one or more resources by containing or pointing to their raw media data.
`MediaResource`	Element of the `MediaLocatorType` type which allows describing a resource by containing or pointing to its raw media data.

Semantics of the `DescriptionResourceType` type:

Name	*Definition*
`DescriptionResourceType`	Specifies the `DescriptionResourceType` type which is an extension of the `ResourceType` type.
`DescriptionResource`	Specifies the container for any description based on a specific schema specified by the namespace declaration within the description.

8.4 Example

In this example, the `QFDeclaration` element allows to describe an image resource for retrieval. The following table shows a `QFDeclaration` example of an image "image.jpg" with the identifier `ImageResourceID`. By referring to `ImageResourceID`, a query condition that searches image files similar to the resource can be described in the `QueryCondition` element.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QFDeclaration>
        <Resource resourceID="ImageResourceID" xsi:type="MediaResourceType">
        <MediaResource>
          <MediaUri>image.jpg</MediaUri>
        </MediaResource>
        </Resource>
      </QFDeclaration>
    </Input>
  </Query>
</MpegQuery>
```

The next example shows that the `QFDeclaration` element allows describing Dublin Core metadata as a resource. Any XML-schema valid metadata can be target with putting its namespace in the `AnyDescription` element.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QFDeclaration>
        <Resource resourceID="DC_title" xsi:type="DescriptionResourceType">
         <AnyDescription xmlns:dc="http://purl.org/dc/elements/1.1/"
                xsi:schemaLocation="http://purl.org/dc/elements/1.1/ dc.xsd">
          <dc:title>World Cup 2006</dc:title>
         </AnyDescription>
        </Resource>
      </QFDeclaration>
    </Input>
  </Query>
</MpegQuery>
```

9 Output Description

9.1 Introduction

The `OutputDescription` element is part of the `Input` element in the `Query` element which describes the structure of the returned data from the responder to the requester. It specifies which data shall be included in the response which is defined by the `Output` element of the `Query` element and how it is structured. Furthermore, it allows limiting the maximum number of items per output page and the overall item count. Besides, it also allows the users to use the aggregation and sorting processes in the `OutputDescription` element.

9.2 Syntax

```
<complexType name="OutputDescriptionType">
  <sequence>
    <element name="ReqField" type="mpqf:FieldType" minOccurs="0"
maxOccurs="unbounded"/>
    <element name="ReqAggregateID" type="IDREF" minOccurs="0"
maxOccurs="unbounded"/>
   <element name="ReqSemanticField" type="string" minOccurs="0"
               maxOccurs="unbounded"/>
    <element name="GroupBy" minOccurs="0">
      <complexType>
        <sequence>
          <element name="GroupByField" type="mpqf:FieldType"
maxOccurs="unbounded"/>
          <element name="Aggregate" type="mpqf:AggregateExpressionType"
minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </complexType>
    </element>
    <element name="SortBy" type="mpqf:AbstractSortByType" minOccurs="0"
maxOccurs="unbounded"/>
  </sequence>
  <attribute name="maxPageEntries" type="positiveInteger" use="optional"/>
  <attribute name="maxItemCount" type="positiveInteger" use="optional"/>
  <attribute name="freeTextUse" type="boolean" use="optional"/>
  <attribute name="thumbnailUse" type="boolean" use="optional"/>
  <attribute name="mediaResourceUse" type="boolean" use="optional"/>
  <attribute name="outputNameSpace" type="anyURI" use="optional"/>
  <attribute name="distinct" type="boolean" use="optional"
default="false"/>
<!-- This indicates that if I want to receive a link to the actual media
resource. -->
</complexType>

<complexType name="FieldType">
  <simpleContent>
    <extension base="mpqf:xPathType">
      <attribute name="typeName" type="string" use="optional"/>
      <attribute name="fragmentResultName" 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>
```

9.3 Semantics

Semantics of the `OutputDescriptionType` type:

Name	*Definition*
`OutputDescriptionType`	Describes the way how the information related to the selected digital items should be returned in the output pages.
`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). If the `GroupBy` element appears in the query, only aggregation results and fields appearing within a `GroupByField` element can be requested.
`ReqAggregateID`	Describes the ID of the aggregate operation the requester asks to be returned. When one or more `ReqAggregateID`s are used, the aggregate ID should be in the `GroupBy` element.
`ReqSemanticField`	Describes the place holder of the desired semantic variable, which is filtered in the Semantic Expression part of the QueryCondition. The place holder follows the SPARQL syntax beginning with a question mark and a descriptive identifier: e.g. ?person
`GroupBy`	Describes the grouping operation the user wants to apply to the query results.
`GroupByField`	Describes the key for the grouping process. In a usual case, leaf nodes are intended to be used as an instance of the `GroupByField`. When a structured field (e.g. DS) is defined as a key, all the mandatory elements and attributes included in the pointed node shall be considered for grouping. For an optional field to be considered in the grouping, it should be specified individually. All used keys in the `GroupByField` element are required to appear as a `ReqField` element in the `OutputDescription` element.
`Aggregate`	Describes the aggregate operation the user wants to apply to the results of the specified grouping operation.
`SortBy`	Describes the sort operation the user wants to apply to the query results.

Name	*Definition*
maxPageEntries	Describes the maximum number of items in one output page. (optional) The result items of a given query can be split among different output pages.
maxItemCount	Describes the maximum number of result items. (optional)
freeTextUse	Specifies the desire that the TextResult field appears in each result item. (optional) If the value of this attribute is false, the TextResult field shall not appear in the result items. If the value of this attribute is not specified, it is assumed that the requester does not care whether the values appear or not.
thumbnailUse	Specifies the desire that the Thumbnail field appears in each result item. (optional) If the value of this attribute is false, the Thumbnail field is not allowed to appear in the result items. If the value of this attribute is not specified, it is assumed that the requester does not care whether the values appear or not.
mediaResourceUse	Specifies the desire that the MediaResource field appears in each result item. (optional) If the value of this attribute is false, the MediaResource field is not allowed to appear in the result items. If the value of this attribute is not specified, it is assumed that the requester does not care whether the values appear or not.
outputNameSpace	Specifies the URI of the namespace associated to the desired metadata format to appear within the Description element of each result item. It may be a namespace related to MPEG-7 or to any other metadata specifications.
Distinct	Specifies the desire that records containing exact the same metadata are required to be merged or not.

Semantics of the FieldType type:

Name	*Definition*
FieldType	Specifies a data path within an item's metadata. This type is derived by extension from xPathType type. Paths are specified by making use of absolute XPath expressions, which refer to the root of the item's metadata, or optionally relative XPath expressions, which refer to a complex type of a schema given by the outputNameSpace attribute.
fragmentResultName	If the *resultMode* attribute is set to "flat", this attribute specifies the value which will appear within the *name* attribute of the corresponding *FragmentResult* element in each result item.
typeName	Specifies the name of a complex data type defined in MPEG-7 or any other metadata schema given by the outputNameSpace attribute, which will serve as the starting point of a relative XPath expression. (optional).

Name	*Definition*
fromREF	Describes a reference to the Join object ID for use in the OutputDescription element for Join operation.
fieldREF	Describes a reference to the ID of a Field of other operations for reuse.
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).

9.4 Example

The following example illustrates the use of the OutputDescription element for describing the desired output of an MPQF query. 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 five fields from the metadata of the resulting digital items. Additionally, the OutputDescription element specifies that three result pages or less will be computed, that each one of these pages shall have at most ten items and that the freeTextUse field is desired.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <OutputDescription maxItemCount="30" maxPageEntries="10" freeTextUse="true"
outputNameSpace="urn:mpeg:mpeg7:schema:2004" >
        <ReqField typeName="MediaLocatorType">/MediaUri</ReqField>
        <ReqField typeName="CreationInformationType">/Creation/Title</ReqField>
        <ReqField typeName="CreationInformationType">/Creation/Creator</ReqField>
        <ReqField typeName="MediaFormatType">/FileFormat</ReqField>
        <ReqField typeName="MediaFormatType">/FileSize</ReqField>
      </OutputDescription>
      <QueryCondition>
        <TargetMediaType xsi:type="mimeType">image/jpeg</TargetMediaType>
        <Condition xsi:type="QueryByFreeText">
          <FreeText>San Jose</FreeText>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

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>
        <FragmentResult name="width">640</FragmentResult>
        <FragmentResult name="height">480</FragmentResult>
      </ResultItem>
      <ResultItem recordNumber="2">
        <TextResult>Title 02</TextResult>
        <FragmentResult name="width">320</FragmentResult>
        <FragmentResult name="height">200</FragmentResult>
      </ResultItem>
      <ResultItem recordNumber="3">
        <TextResult>Title 03</TextResult>
        <FragmentResult name="width">800</FragmentResult>
        <FragmentResult name="height">1000</FragmentResult>
      </ResultItem>
    </Output>
  </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>
        <Description xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004">
          <mpeg7:Mpeg7>
            <mpeg7:DescriptionUnit xsi:type="mpeg7:CreationInformationType">
              <mpeg7:Creation>
                <mpeg7:Title>Title 01</mpeg7:Title>
              </mpeg7:Creation>
            </mpeg7:DescriptionUnit>
          </mpeg7:Mpeg7>
          <mpeg7:Mpeg7>
            <mpeg7:DescriptionUnit xsi:type="mpeg7:CreationType">
```

```
              <mpeg7:Title>Title 01</mpeg7:Title>
                <mpeg7:Creator xsi:type="mpeg7:CreatorType">
                  <mpeg7:Role href="http://www.roles.com/author"/>
                  <mpeg7:Agent xsi:type="mpeg7:PersonType">
                    <mpeg7:Name>
                      <mpeg7:GivenName>John</mpeg7:GivenName>
                      <mpeg7:FamilyName>Smith</mpeg7:FamilyName>
                    </mpeg7:Name>
                  </mpeg7:Agent>
                </mpeg7:Creator>
            </mpeg7:DescriptionUnit>
          </mpeg7:Mpeg7>
        </Description>
        <AggregationResult aggregateID="avgSize">256</AggregationResult>
      </ResultItem>
      <ResultItem recordNumber="2">
        <TextResult>Title 02</TextResult>
        <AggregationResult aggregateID="avgSize">512</AggregationResult>
      </ResultItem>
      <ResultItem recordNumber="3">
        <TextResult>Title 03</TextResult>
        <AggregationResult aggregateID="avgSize">1024</AggregationResult>
      </ResultItem>
    </Output>
  </Query>
</MpegQuery>
```

9.5 SortBy

9.5.1 Introduction

This sub clause specifies all the types of `SortBy` tools which are used in the `OutputDescription` element. The `AbstractSortByType` type is defined as an abstract data type from which the `SortByFieldType` type and `SortByAggregateType` type are derived.

9.5.2 Syntax

```
<complexType name="AbstractSortByType" abstract="true">
   <attribute name="order" use="required">
      <simpleType>
         <restriction base="string">
            <enumeration value="ascending"/>
            <enumeration value="descending"/>
         </restriction>
      </simpleType>
   </attribute>
</complexType>

<complexType name="SortByFieldType">
   <complexContent>
      <extension base="mpqf:AbstractSortByType">
         <choice>
            <element name="Field" type="mpqf:FieldType"/>
             <element name="SemanticField" type="mpqf:SemanticFieldType"/>
         </choice>
      </extension>
   </complexContent>
```

```
</complexType>

<complexType name="SortByAggregateType">
   <complexContent>
      <extension base="mpqf:AbstractSortByType">
         <choice>
            <element name="Aggregate" type="mpqf:AggregateExpressionType"/>
            <element name="AggregateID" type="IDREF"/>
         </choice>
      </extension>
   </complexContent>
</complexType>
```

9.5.3 Semantics

Semantics of the `SortBy` types:

Name	*Definition*
`AbstractSortByType`	Defines the abstract data type for the two `SortBy` sub-types.
`order`	Indicates the order of the sorting. The current version supports either `ascending` or `descending` sort order.
`SortByFieldType`	Describes a sort type in which the `Field` element is used as a sorting key.
`SemanticField`	Describes the place holder of the desired semantic variable which should be used for sorting. The semantic field is filtered in the Semantic Expression part of the QueryCondition or stems from the SPARQL query type. The place holder follows the SPARQL syntax beginning with a question mark and a descriptive identifier: e.g. ?person
`SortByAggregateType`	Describes a sort type in which the result of an aggregate expression is used as a key.
`Aggregate`	Describes an aggregate expression, whose result is used as a key in sorting process.
`AggregateID`	Describes an aggregate expression by id reference, whose result is used as a key in the sorting process.

9.5.4 Example

The following example illustrates the use of the `OutputDescription` element for describing the desired output of an MPQF query in combination with a `GroupBy` and `SortBy` statement. In addition, the use of an aggregate function is demonstrated. Both statements (`ReqAggregateID` and `SortBy`) reference to the introduced `Aggregate` element in the `GroupBy`.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <OutputDescription outputNameSpace="urn:mpeg:mpeg7:2004">
        <ReqField typeName="CreationInformation">/Creation/Title</ReqField>
        <ReqField typeName="Creator">/Character/FamilyName</ReqField>
        <ReqAggregateID>avgSize</ReqAggregateID>
        <GroupBy>
          <GroupByField typeName="Creator">/Character/FamilyName</GroupByField>
          <Aggregate xsi:type="AVG" aggregateID="avgSize">
            <Field typeName="MediaFormat">/FileSize</Field>
          </Aggregate>
        </GroupBy>
        <SortBy xsi:type="SortByAggregateType" order="ascending">
          <AggregateID>avgSize</AggregateID>
        </SortBy>
      </OutputDescription>
    </Input>
  </Query>
</MpegQuery>
```

The following is an example of expected output by the specified `OutputDescription` element above. Note that the actual output depends on the multimedia content stored in the repository.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Output currPage="1" totalPages="1" expirationDate="2008-05-30T09:00:00">
      <ResultItem xsi:type="ResultItemType" recordNumber="1">
        <TextResult>Title 01</TextResult>
        <Description xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004">
          <mpeg7:Mpeg7>
            <mpeg7:DescriptionUnit xsi:type="mpeg7:CreationInformationType">
              <mpeg7:Creation>
                <mpeg7:Title>Title 01</mpeg7:Title>
              </mpeg7:Creation>
            </mpeg7:DescriptionUnit>
          </mpeg7:Mpeg7>
        </Description>
        <AggregationResult aggregateID="avgSize">256</AggregationResult>
      </ResultItem>
      <ResultItem recordNumber="2">
        <TextResult>Title 02</TextResult>
        <AggregationResult aggregateID="avgSize">512</AggregationResult>
      </ResultItem>
      <ResultItem recordNumber="3">
        <TextResult>Title 03</TextResult>
        <AggregationResult aggregateID="avgSize">1024</AggregationResult>
      </ResultItem>
    </Output>
  </Query>
</MpegQuery>
```

10 Query Condition

10.1 Introduction

The QueryCondition element, of the QueryConditionType type, is the part of the Input Query Format where the user specifies the properties of the media or the metadata to be retrieved. The QueryCondition element is defined as a sequence of desired target media types, an EvaluationPath element and a condition tree, which can be a simple (Condition element) or a Join operation (Join element). The sequence of the target media types declares the desired MIME types which the user expects as a result. The EvaluationPath element (optional) declares an XPath expression, which specifies the node of the metadata fragment related to the evaluation item being addressed (//VideoSegment for instance). It also determines the structure of the output; one result item will be returned for each evaluation item if it matches the condition. If the EvaluationPath element is not specified, the output result shall be provided as a collection of multimedia contents, as stored in the repository each of which satisfies the query condition. If the QueryCondition element does not appear within the Input element, or if the QueryCondition element does not conain neither a Condition nor a Join element, it is considered to be an Empty Query asking to retrieve all records in the database.The condition tree can be specified using the Condition element, of the BooleanExpressionType type, which is the parent abstract type of all the types resulting in a value in the range of [0..1]. A more complex condition tree can be specified using the Join element. The Join operation allows the definition of filtering conditions which act over two sets of multimedia objects.

10.2 Syntax

```
<complexType name="QueryConditionType">
  <sequence>
    <element name="EvaluationPath" type="mpqf:xPathType" minOccurs="0"/>
    <element name="TargetMediaType" type="mpqf:mimeType" minOccurs="0"
maxOccurs="unbounded"/>
    <choice minOccurs="0">
      <element name="Join" type="mpqf:JoinType"/>
      <element name="Condition" type="mpqf:BooleanExpressionType"/>
    </choice>
  </sequence>
</complexType>
```

10.3 Semantics

Semantics of the QueryConditionType type:

Name	*Definition*
QueryConditionType	Defines a sequence of desired target media types, an EvaluationPath element, and a condition tree, which can be simple (Condition element) or a Join operation (Join element).
EvaluationPath	Optional XPath expression, which specifies the node of the metadata fragment related to the evaluation item being addressed. It also determines the structure of the output; one result item will be returned for each evaluation item if it matches the condition. If the EvaluationPath element is not specified, the output result shall be provided as a collection of multimedia contents, as stored in the repository each of which satisfies the query condition.

Name	*Definition*
TargetMediaType	The sequence of the target media types declares the desired MIME types which the user expects as a result. This element is required not to be used in case a Join element is stated.
Condition	The condition tree of the query is specified using a BooleanExpressionType type, which is the parent abstract type of all the types resulting in a value in the range of [0..1].
Join	Optionally, a Join element can be used instead of the Condition element. A Join operation allows the definition of filtering conditions which act over two sets of multimedia objects. With the Join operation, it is possible to specify conditions which define two separate filtered sets and combine them (see JoinType type).

10.4 Example

The following example illustrates the usage of the QueryConditionType type. It demonstrates a combined (by AND) retrieval for all items containing the word "Lausanne" somewhere in a textual description and where the annotated Title is equal to the string "Blade Runner". In addition, the target domain is limited to images of the JPEG compression format (see TargetMediaType element). Note, that the target domain also can be restricted to all images by using *image/**.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <TargetMediaType>image/jpeg</TargetMediaType>
        <Condition xsi:type="AND">
          <Condition xsi:type="QueryByFreeText">
            <FreeText>Lausanne</FreeText>
          </Condition>
          <Condition xsi:type="Equal">
            <StringField typeName="CreationType">Title</StringField>
            <StringValue>Blade Runner</StringValue>
          </Condition>
       </Condition>
     </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

The following example illustrates the use of the EvaluationPath element. It shows a query to search for all video segments containing the word "Lausanne" somewhere in a textual description. Note, that for each matching segment, one different result item will be returned.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <EvaluationPath>//VideoSegment</EvaluationPath>
        <Condition xsi:type="QueryByFreeText">
          <FreeText>Lausanne</FreeText>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

11 Expression Types

11.1 Introduction

Expression Types allow to build expression trees. There are three main abstract Expression Types: the `BooleanExpressionType` type, which is the base type of all the expressions resulting in a value in the range of [0..1], the `ArithmeticExpressionType` type which is the base type of all the expressions resulting in a numeric value (Add, Subtract, Multiply, Divide, Modulus, Abs, Ceiling, Floor and Round), and the `StringExpressionType` type, which is the base type of all the expressions resulting in a string value (UpperCase and LowerCase).

11.2 Syntax

```
<complexType name="BooleanExpressionType" abstract="true">
  <attribute name="preferenceValue" type="mpqf:zeroToOneType"
                 use="optional" default="1"/>
    <attribute name="thresholdValue" type="mpqf:zeroToOneType"
    use="optional" />
</complexType>
<complexType name="ArithmeticExpressionType" abstract="true"/>
<complexType name="StringExpressionType" abstract="true"/>
```

11.3 Semantics

Semantics of the abstract `Expression` types:

Name	*Definition*
`BooleanExpressionType`	An abstract type that is the base type for defining the boolean expressions.
`preferenceValue`	Used to indicate the importance, priority or weight assigned to a particular user preference component, relative to other components. The range of the preference values is from 0 to 1. Positive float value indicates the degree of desired preference. A default, positive, value of 1 corresponds to a nominal preference. The sum of all assigned preference values at one level is required to be 1.
`thresholdValue`	Used to indicate the minimum value the score of an evaluation-item based on a specific operation (query type, expression) is required to have. Otherwise the evaluation-item is not considered further during evaluation.
`ArithmeticExpressionType`	An abstract type that is the base type for defining the arithmetic expressions.
`StringExpressionType`	An abstract type that is the base type for defining the string expressions.

11.4 Operands

11.4.1 DateTimeOperands

11.4.1.1 Introduction

The DateTimeOperands group allows specifying one element which has a date or time value as content. Each element can instantiate one of the four types which result in a date or a time value (the date type, the time type, the dateTime type and the FieldType type).

11.4.1.2 Syntax

```
<group name="DateTimeOperands">
  <sequence>
    <choice>
      <element name="DateTimeValue" type="dateTime"/>
      <element name="DateValue" type="date"/>
      <element name="TimeValue" type="time"/>
      <element name="DateTimeField" type="mpqf:FieldType"/>
    </choice>
  </sequence>
</group>
```

11.4.1.3 Semantics

Semantics of the DateTimeOperands group:

Name	*Definition*
DateTimeOperands	Defines the DateTimeOperands group, which allows specifying one element that can instantiate one of the four types which result in a date and/or a time value.
DateTimeValue	Allows specifying a constant date-time value by dateTime type.
DateValue	Allows specifying a constant date value by date type.
TimeValue	Allows specifying a constant time value by time type.
DateTimeField	Allows specifying a path to a date-time value within the metadata of a multimedia item by FieldType type.

11.4.1.4 Example

The following example illustrates how the group DateTimeOperands allows that the input of the Equal operation can be a constant date-time value and a path to a date-time value within the metadata of the evaluation item.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <Condition xsi:type="Equal">
          <DateTimeField
typeName="CreationType">/CreationCoordinates/Date/TimePoint</DateTimeField>
```

```
 <DateTimeValue>2002-05-30T09:00:00</DateTimeValue>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

11.4.2 DurationOperands

11.4.2.1 Introduction

The `DurationOperands` group allows specifying one element which consists of a duration value. Each element can instantiate one of the two types that result in a duration value (the `duration` type and the `FieldType` type).

11.4.2.2 Syntax

```
<group name="DurationOperands">
  <sequence>
    <choice>
      <element name="DurationValue" type="duration"/>
      <element name="DurationField" type="mpqf:FieldType"/>
    </choice>
  </sequence>
</group>
```

11.4.2.3 Semantics

Semantics of the `DurationOperands` group:

Name	*Definition*
`DurationOperands`	Defines the `DurationOperands` group, which allows specifying one element from one of the two types that result in a duration value.
`DurationValue`	Allows specifying a constant duration value of `duration` type.
`DurationField`	Allows specifying a path to a duration value within the metadata of an evaluation item by `FieldType` type.

11.4.2.4 Example

The following example illustrates how the group `DurationOperands` allows that the input of the `Equal` operation can be a constant duration value and a path to a duration value within the metadata of the object for example.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <Condition xsi:type="Equal">
  <DurationField typeName="SomeVideoDescriptionType">Duration</DurationField>
  <DurationValue>P5Y2M10DT15H</DurationValue>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

11.5 Boolean Expression Types

11.5.1 Introduction

The `BooleanExpressionType` type is an abstract type which is the base type of the Boolean operators (`AND`, `OR`, `NOT`, `XOR`), the `ComparisonExpressionType` type and the `QueryType` type. These types allow to build expressions evaluating to a value in the range of [0..1]. The `BooleanExpressionType` type provides one attribute that indicates the degree of preference (relative weight of importance) of the query condition. Preference values (if present) are always taken into account within the same hierarchy level of the XML tree (i.e. between siblings of the same sequence). The `AND` and the `OR` types should have two or more condition elements. The `XOR` type should have exactly two condition elements, and the `NOT` type should have exactly one condition element.

11.5.2 Syntax

```
<complexType name="AND">
  <complexContent>
    <extension base="mpqf:BooleanExpressionType">
      <sequence>
        <element name="Condition" type="mpqf:BooleanExpressionType"
                      minOccurs="2" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="scoringFunction" type="mpqf:SimpleTermType"
use="optional"/>
    </extension>
  </complexContent>
</complexType>
<complexType name="OR">
  <complexContent>
    <extension base="mpqf:BooleanExpressionType">
      <sequence>
        <element name="Condition" type="mpqf:BooleanExpressionType"
                       minOccurs="2" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="scoringFunction" type="mpqf:SimpleTermType"
use="optional"/>
    </extension>
  </complexContent>
</complexType>
<complexType name="NOT">
  <complexContent>
    <extension base="mpqf:BooleanExpressionType">
      <sequence>
        <element name="Condition" type="mpqf:BooleanExpressionType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="XOR">
  <complexContent>
```

```
    <extension base="mpqf:BooleanExpressionType">
      <sequence>
        <element name="Condition" type="mpqf:BooleanExpressionType"
                       minOccurs="2" maxOccurs="2"/>
      </sequence>
      <attribute name="scoringFunction" type="mpqf:SimpleTermType"
use="optional"/>
    </extension>
  </complexContent>
</complexType>
```

11.5.3 Semantics

Semantics of the Boolean Expression Types:

Name	*Definition*
`BooleanExpressionType`	An abstract type that is the base type for defining the boolean expressions.
`scoringFunction`	An attribute referring to a classification scheme, which contains different scoring functions. If combining two different QueryTypes, such as `QueryByMedia` and `QueryByDescription` with an `AND`, a fuzzy logic may be applied. That means that values of type float are aggregated based on the desired scoring function (e.g,, `Min`, `Sum`, `Product`, etc.). The terms of supported scoring functions are required to be refered at a classification scheme. In case a responder may not support this attribute, it should be ignored.
`AND`	A complex type inherited from the `BooleanExpressionType` type, for using the Boolean "AND" operator in a query and it shall result to a value in the range of [0..1]. If the resulting values (scores) are not exactly 0 or 1, the computation of the result for the `AND` operation is up to the implementers, who may apply rules as provided in subclause 11.5.4.
`OR`	A complex type inherited from the `BooleanExpressionType` type, for using the Boolean "OR" operator in query and it shall result to a value in the range of [0..1]. If the resulting values (scores) are not exactly 0 or 1, the computation of the result for the `OR` operation is up to the implementers, who may apply rules as provided in subclause 11.5.4.
`NOT`	A complex type inherited from the `BooleanExpressionType` type, for using the "NOT" operator in a query and it shall result to a value in the range of [0..1]. If the resulting values (scores) are not exactly 0 or 1, the computation of the result for the `NOT` operation is up to the implementers, who may apply rules as provided in subclause 11.5.4.
`XOR`	A complex type inherited from the `BooleanExpressionType` type, for using the "XOR" operator in query and it shall result to a value in the range of [0..1]. If the resulting values (scores) are not exactly 0 or 1, the computation of the result for the `XOR` operation is up to the implementers, who may apply rules as provided in subclause 11.5.4.

11.5.4 Definitions

This subclause is an informative section.

It is recommended that the scoring function of the AND operator may conform to the rules applied in Definition 1 (t-norm) and to the weighted t-norm in case of a given preference value. The final score is calculated by the (weighted) input values of the respective conditions.

It is recommended that the scoring function of the OR operator may conform to the rules applied in Definition 2 (t-conorm) and to the weighted t-conorm in case of a given preference value. The final score is calculated by the (weighted) input values of the respective conditions.

It is recommended that the evaluation of the NOT operator may conform to the rules applied in Definition 3. The final score is calculated by the (weighted) input values of the respective conditions.

It is recommended that the scoring function of the XOR operator may conform to the rules applied in Definition 4 and to the same weighted rules as for t-conorm in case of a given preference value. The final score is calculated by the (weighted) input values of the respective conditions

In order to evaluate individual scores at the AND, OR or XOR element, this part of ISO/IEC 15938 recommends to use the scoring function T: [0,1] X [0,1] -> [0,1] which relates to the t-norms (for AND) and t-conorms (for OR).

Definition 1: Function T(AND): [0,1] x [0,1]-> [0,1] is a t-norm if it satisfies the following criteria:

1. $T(s_1, s_2) = T(s_2, s_1)$ (commutativity)
2. $T(s_1, T(s_2, s_3)) = T(T(s_1, s_2), s_3)$ (associativity)
3. $s_1 \leq s_2 \wedge s_3 \leq s_4 \rightarrow T(s_1, s_2) \leq T(s_3, s_4)$ (monotonicy)
4. $T(s_1, 1) = s_1$ (border condition)

Definition 2: Function T(OR): [0,1] x [0,1]-> [0,1] is a t-conorm if it satisfies the following criteria:

1. criteria 1-3 of the t-norm
2. $T(s_1, 0) = s_1$ (border condition)

The score evaluation for the NOT operation is defined as follows:

Definition 3: Function T(NOT): [0,1] -> [0,1] requires the following conditions:

1. $T(NOT)(T(NOT)(s_1)) = s_1$
2. DeMorgan laws are applied:
   a. $T(NOT)\ (T(AND)(s_1, s_2)) = T(OR)\ (T(NOT)(s_1), T(NOT)(s_2))$
   b. $T(NOT)\ (T(OR)(s_1, s_2)) = T(AND)\ (T(NOT)(s_1), T(NOT)(s_2))$

The score evaluation for the XOR operation is defined as follows:

Definition 4: Function T(XOR): $[0,1]^n$ -> [0,1] is realized by combining the operation AND, OR and NOT as follows:

1. $T(XOR)(s_1, s_2, ..., s_n) = T(OR)\ (\ T(AND_i)\ )$ with i=1..n
2. $T(AND_i) = T(AND)\ (T(NOT)\ s_1,.., T(NOT)\ s_{(i-1)}, s_i, T(NOT)_{(si+1)},..., T(NOT)\ s_n)$

In order to integrate preference values (p ε [0,1]), the weighted t-norm and t-conorm are defined:

$T(AND_p)$: $([0,1] \times [0,1])^n \rightarrow [0,1]$, where the first parameter identifies the preference. In addition, the following rules apply:

1. $T(AND_p)\ (0,s_1, p_2, s_2, \ldots, p_n, s_n) = T(AND_p)\ (p_2,s_2, \ldots, p_n, s_n)$
2. If for all $p_1, \ldots, p_n$ in the filter condition tree (at one level) it is true that: $p_1 = \ldots = p_n$ (all preference values have the same value) then:

   $$T(AND_p)\ (p_1, s_1, p_2, s_2,\ldots, p_n, s_n) = T(AND)(s_1, s_2, \ldots, s_n)$$
3. The weighted scoring function T within the filter condition tree must be continuous concerning the scores (s_x) as well as its weights (p_x).

Note, the same rules apply to $T(OR_p)$. The function $T(NOT_p)$ is reduced to $T(NOT)$. The function $T(XOR_p)$ is substituted by $T(AND_p)$, $T(OR_p)$ and $T(NOT_p)$ as defined in definition 4.

11.5.5 Example

The following example illustrates compounding some Boolean operations. The top level “AND” operator, includes four nested operations (“AND”, “OR”, “XOR”, and “NOT”). The second level “AND” operator, with preference value “0.1”, includes two kinds of query conditions, and the “OR” and the “XOR” operators, with preference value “0.4”, include two query conditions. The “NOT” operator, with preference value “0.1”, includes only one query condition.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <EvaluationPath>//VideoSegment</EvaluationPath>
        <Condition xsi:type="AND">
          <Condition xsi:type="AND" preferenceValue="0.1">
            <Condition xsi:type="QueryByFreeText" preferenceValue="0.7">
              <FreeText>Lausanne</FreeText>
            </Condition>
            <Condition xsi:type="QueryByXQuery" preferenceValue="0.3">
              <XQuery><![CDATA[
          $a := node()//MediaFormat/Content/Name
          $a/text()="Video" ]]></XQuery>
            </Condition>
          </Condition>
          <Condition xsi:type="OR" preferenceValue="0.4">
            <Condition xsi:type="Equal" >
              <DateTimeField
typeName="CreationType">/CreationCoordinates/Date/TimePoint</DateTimeField>
              <DateTimeValue>2002-05-30T09:00:00</DateTimeValue>
            </Condition>
            <Condition xsi:type="QueryByFreeText" >
              <FreeText>San Jose</FreeText>
            </Condition>
          </Condition>
          <Condition xsi:type="XOR" preferenceValue="0.4">
            <Condition xsi:type="Equal">
              <BooleanField
typeName="MediaFormatType">bitrate/@variable</BooleanField>
              <BooleanValue>true</BooleanValue>
            </Condition>
            <Condition xsi:type="QueryByFreeText">
              <FreeText>Marrakech</FreeText>
            </Condition>
          </Condition>
          <Condition xsi:type="NOT" preferenceValue="0.1">
            <Condition xsi:type="Equal">
              <ArithmeticField
typeName="MediaFormatType">FileSize</ArithmeticField>
              <LongValue>1000</LongValue>
            </Condition>
          </Condition>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

11.6 Comparison Expression Types

11.6.1 Introduction

The `ComparisonExpressionType` type is an abstract type which extends the `BooleanExpressionType` type and becomes the base type of all the expression types describing comparison expressions. All the comparison expression types should extend the abstract `ComparisonExpressionType` type. Comparison expressions in this part of ISO/IEC 15938 are required to return a Boolean value (expressed as either 0 or 1) as a result. Currently, this part of ISO/IEC 15938 specifies the following comparison expression types: `GreaterThan`, `GreaterThanEqual`, `LessThan`, `LessThanEqual`, `Equal`, `NotEqual` and `Contains`. These comparison expressions allow specifying a sequence of two elements which represents the operands. The `GreaterThan`, `GreaterThanEqual`, `LessThan`, and `LessThanEqual` expressions are required to have two group member elements of the same kind of `DateTimeOperands` or `DurationOperands or two arithmetic operands`.

An element included in the `Equal` and `NotEqual` expressions can be instantiated by one member of the following operands groups (`DateTimeOperands, DurationOperands) or the respective arithmetic, boolean or string operands`.

The `Contains` comparison expression is required to have only string operands, represented in the `StringOperands` group. By default all comparison expressions that can be instantiated with `StringOperands ("Equal", "NotEqual", "Contains")` should be implemented in a way to process the `StringOperands` case sensitive.

11.6.2 Syntax

```
<complexType name="ComparisonExpressionType" abstract="true">
  <complexContent>
   <extension base="mpqf:BooleanExpressionType"/>
  </complexContent>
</complexType>
<complexType name="GreaterThan">
  <complexContent>
    <extension base="mpqf:ComparisonExpressionType">
      <choice>
        <choice minOccurs="2" maxOccurs="2">
          <element name="DoubleValue" type="double"/>
          <element name="LongValue" type="long"/>
          <element name="ArithmeticField" type="mpqf:FieldType"/>
          <element name="ArithmeticExpression"
                      type="mpqf:ArithmeticExpressionType"/>
           <element name="SemanticArithmeticField"
```

```
                      type="mpqf:SemanticFieldType"/>
        </choice>
       <group ref="mpqf:DateTimeOperands" minOccurs="2" maxOccurs="2"/>
       <group ref="mpqf:DurationOperands" minOccurs="2" maxOccurs="2"/>
   </choice>
    </extension>
  </complexContent>
</complexType>
<complexType name="GreaterThanEqual">
<complexContent>
  <extension base="mpqf:ComparisonExpressionType">
   <choice>
     <choice minOccurs="2" maxOccurs="2">
       <element name="DoubleValue" type="double"/>
       <element name="LongValue" type="long"/>
       <element name="ArithmeticField" type="mpqf:FieldType"/>
       <element name="ArithmeticExpression"
             type="mpqf:ArithmeticExpressionType"/>
       <element name="SemanticArithmeticField" type="mpqf:SemanticFieldType"/>
    </choice>
    <group ref="mpqf:DateTimeOperands" minOccurs="2" maxOccurs="2"/>
    <group ref="mpqf:DurationOperands" minOccurs="2" maxOccurs="2"/>
   </choice>
    </extension>
  </complexContent>
</complexType>
<complexType name="LessThanEqual">
  <complexContent>
   <extension base="mpqf:ComparisonExpressionType">
     <choice>
       <choice minOccurs="2" maxOccurs="2">
         <element name="DoubleValue" type="double"/>
         <element name="LongValue" type="long"/>
         <element name="ArithmeticField" type="mpqf:FieldType"/>
         <element name="ArithmeticExpression"
                type="mpqf:ArithmeticExpressionType"/>
         <element name="SemanticArithmeticField" type="mpqf:SemanticFieldType"/>
       </choice>
       <group ref="mpqf:DateTimeOperands" minOccurs="2" maxOccurs="2"/>
       <group ref="mpqf:DurationOperands" minOccurs="2" maxOccurs="2"/>
     </choice>
    </extension>
  </complexContent>
</complexType>
<complexType name="LessThan">
  <complexContent>
    <extension base="mpqf:ComparisonExpressionType">
     <choice>
       <choice minOccurs="2" maxOccurs="2">
          <element name="DoubleValue" type="double"/>
          <element name="LongValue" type="long"/>
          <element name="ArithmeticField" type="mpqf:FieldType"/>
          <element name="ArithmeticExpression"
                type="mpqf:ArithmeticExpressionType"/>
          <element name="SemanticArithmeticField" type="mpqf:SemanticFieldType"/>
        </choice>
        <group ref="mpqf:DateTimeOperands" minOccurs="2" maxOccurs="2"/>
        <group ref="mpqf:DurationOperands" minOccurs="2" maxOccurs="2"/>
      </choice>
    </extension>
  </complexContent>
</complexType>
<complexType name="Equal">
  <complexContent>
    <extension base="mpqf:ComparisonExpressionType">
       <choice>
         <choice minOccurs="2" maxOccurs="2">
          <element name="DoubleValue" type="double"/>
          <element name="LongValue" type="long"/>
          <element name="ArithmeticField" type="mpqf:FieldType"/>
          <element name="ArithmeticExpression"
                type="mpqf:ArithmeticExpressionType"/>
          <element name="SemanticArithmeticField" type="mpqf:SemanticFieldType"/>
         </choice>
        <choice minOccurs="2" maxOccurs="2">
          <element name="BooleanValue" type="boolean"/>
          <element name="BooleanField" type="mpqf:FieldType"/>
          <element name="BooleanExpression" type="mpqf:BooleanExpressionType"/>
          <element name="SemanticBooleanField" type="mpqf:SemanticFieldType"/>
         </choice>
         <group ref="mpqf:DateTimeOperands" minOccurs="2" maxOccurs="2"/>
          <group ref="mpqf:DurationOperands" minOccurs="2" maxOccurs="2"/>
          <choice minOccurs="2" maxOccurs="2">
           <element name="StringValue" type="string"/>
           <element name="StringField" type="mpqf:FieldType"/>
           <element name="StringExpression" type="mpqf:StringExpressionType"/>
           <element name="SemanticStringField" type="mpqf:SemanticFieldType"/>
          </choice>
        </choice>
     </extension>
  </complexContent>
</complexType>
<complexType name="NotEqual">
  <complexContent>
    <extension base="mpqf:ComparisonExpressionType">
   <choice>
     <choice minOccurs="2" maxOccurs="2">
       <element name="DoubleValue" type="double"/>
       <element name="LongValue" type="long"/>
       <element name="ArithmeticField" type="mpqf:FieldType"/>
       <element name="ArithmeticExpression"
                type="mpqf:ArithmeticExpressionType"/>
        <element name="SemanticArithmeticField" type="mpqf:SemanticFieldType"/>
     </choice>
     <choice minOccurs="2" maxOccurs="2">
       <element name="BooleanValue" type="boolean"/>
       <element name="BooleanField" type="mpqf:FieldType"/>
       <element name="BooleanExpression" type="mpqf:BooleanExpressionType"/>
       <element name="SemanticBooleanField" type="mpqf:SemanticFieldType"/>
    </choice>
    <group ref="mpqf:DateTimeOperands" minOccurs="2" maxOccurs="2"/>
    <group ref="mpqf:DurationOperands" minOccurs="2" maxOccurs="2"/>
    <choice minOccurs="2" maxOccurs="2">
      <element name="StringValue" type="string"/>
      <element name="StringField" type="mpqf:FieldType"/>
      <element name="StringExpression" type="mpqf:StringExpressionType"/>
     <element name="SemanticStringField" type="mpqf:SemanticFieldType"/>
    </choice>
   </choice>
    </extension>
  </complexContent>
</complexType>
<complexType name="Contains">
  <complexContent>
    <extension base="mpqf:ComparisonExpressionType">
   <sequence>
     <choice minOccurs="2" maxOccurs="2">
       <element name="StringValue" type="string"/>
       <element name="StringField" type="mpqf:FieldType"/>
       <element name="StringExpression" type="mpqf:StringExpressionType"/>
       <element name="SemanticStringField" type="mpqf:SemanticFieldType"/>
     </choice>
   </sequence>
    </extension>
  </complexContent>
</complexType>
```

11.6.3 Semantics

Semantics of the Comparison Expression Types:

Name	*Definition*
`ComparisonExpressionType`	An abstract type based on the `BooleanExpressionType` type that is the base type for defining comparison expressions.
`GreaterThan`	A complex type based on the `ComparisonExpressionType` type, for using the comparable “greater than” operation in a query. This operation has two operands that are required to be values of `Arithmetic`, `DateTime`, or `Duration` operand type.
`GreaterThanEqual`	A complex type based on the `ComparisonExpressionType` type, for using the comparable “greater than” or “equal” operation in a query. This operation has two operands that are required to be values of `Arithmetic`, `DateTime`, or `Duration` operand type.
`LessThan`	A complex type based on the `ComparisonExpressionType` type, for using the comparable “less than” operation in a query. This operation has two operands that are required to be values of `Arithmetic`, `DateTime`, or `Duration` operand type.
`LessThanEqual`	A complex type based on the `ComparisonExpressionType` type, for using the comparable “less than” or “equal” operation in a query. This operation has two operands that are required to be values of `Arithmetic`, `DateTime`, or `Duration` operand type.
`Equal`	A complex type based on the `ComparisonExpressionType` type, for using the comparable “equal” operation in a query. This operation has two operands that are required to be values of `Arithmetic`, `Boolean`, `String`, `DateTime`, or `Duration` operand type. String operands are processed case sensitive by default.
`NotEqual`	A complex type based on the `ComparisonExpressionType` type, for using the comparable “not equal” operation in a query. This operation has two operands that are required to be values of `Arithmetic`, `Boolean`, `String`, `DateTime`, or `Duration` operand type. `String` operands are

Name	*Definition*
	processed case sensitive by default.
`Contains`	A complex type based on the `ComparisonExpressionType` type, for using the comparable "contains" expression in a query. The `Contains` expression specifies the query expression for a text query and looks for the following string anywhere in the preceding string. This expression has two operands that are required to be values of `String` type. `String` operands are processed case sensitive by default.
DoubleValue	Allows specifying a constant arithmetic value by using double type.
LongValue	Allows specifying a constant arithmetic value by using long type.
ArithmeticField	Allows specifying a path to a numeric value within the metadata of an evaluation item using FieldType type.
ArithmeticExpression	Allows specifying an expression that is required to be evaluated to an arithmetic value. The abstract type ArithmeticExpressionType is the base type of all operations with a numeric output (Add, Subtract, etc.).
SemanticArithmeticField	Allows specifying the path to a numeric value of a semantic property.
BooleanValue	Allows specifying a constant Boolean value by using boolean type.
BooleanField	Allows specifying a path to a Boolean value within the metadata of an evaluation item, by using FieldType.
BooleanExpression	Allows specifying an expression evaluating to a Boolean value. The abstract type BooleanExpressionType is the base type from which all the operations with a Boolean output (Equal, etc.), the query types and the boolean operators (AND, OR, etc.) are inherited.
SemanticBooleanField	Allows specifying a path to a Boolean value of a semantic property.
StringValue	Allows specifying a constant string value by using string type.
StringField	Allows specifying a path to a string value within the metadata of an evaluation item, by using FieldType type.
StringExpression	Allows specifying an expression evaluating to a string value. The abstract type StringExpressionType is the base type, from which all the operations with a string output are inherited.
SemanticStringField	Allows specifying a path to a string value of a semantic property.

11.6.4 Example

The following example illustrates that comparison expression types are inherited from `ComparisonExpressionType` type.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <Condition xsi:type="GreaterThan">
          <ArithmeticField
             typeName="MediaFormatType">/TargetChannelBitRate</ArithmeticField>
          <LongValue>340000</LongValue>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

11.7 Arithmetic Expression Types

11.7.1 Introduction

The `ArithmeticExpressionType` type is an abstract type, which is the base type of all the arithmetic expressions. All the arithmetic expressions (including the aggregate expressions) are required to extend the `ArithmeticExpressionType` type. Currently, this part of ISO/IEC 15938 specifies the following arithmetic expressions: `Add`, `Subtract`, `Multiply`, `Divide`, `Modulus`, `Abs`, `Ceiling`, `Floor` and `Round`. The `Add`, `Subtract`, `Multiply`, `Divide` and `Modulus` arithmetic expressions should have two operands that reference member elements of the `ArithmeticOperand` group. The `Abs`, `Ceiling`, `Floor` and `Round` arithmetic expressions should have a single operand of type `ArithmeticExpressionType`.

11.7.2 Syntax

```
<complexType name="Add">
 <complexContent>
  <extension base="mpqf:ArithmeticExpressionType">
   <sequence>
    <choice minOccurs="2" maxOccurs="2">
   <element name="DoubleValue" type="double"/>
   <element name="LongValue" type="long"/>
   <element name="ArithmeticField" type="mpqf:FieldType"/>
   <element name="ArithmeticExpression" type="mpqf:ArithmeticExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="Subtract">
 <complexContent>
  <extension base="mpqf:ArithmeticExpressionType">
   <sequence>
    <choice minOccurs="2" maxOccurs="2">
   <element name="DoubleValue" type="double"/>
   <element name="LongValue" type="long"/>
   <element name="ArithmeticField" type="mpqf:FieldType"/>
   <element name="ArithmeticExpression" type="mpqf:ArithmeticExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="Multiply">
 <complexContent>
   <extension base="mpqf:ArithmeticExpressionType">
    <sequence>
    <choice minOccurs="2" maxOccurs="2">
   <element name="DoubleValue" type="double"/>
   <element name="LongValue" type="long"/>
   <element name="ArithmeticField" type="mpqf:FieldType"/>
   <element name="ArithmeticExpression" type="mpqf:ArithmeticExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="Divide">
 <complexContent>
  <extension base="mpqf:ArithmeticExpressionType">
   <sequence>
    <choice minOccurs="2" maxOccurs="2">
    <element name="DoubleValue" type="double"/>
    <element name="LongValue" type="long"/>
    <element name="ArithmeticField" type="mpqf:FieldType"/>
    <element name="ArithmeticExpression" type="mpqf:ArithmeticExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="Modulus">
 <complexContent>
   <extension base="mpqf:ArithmeticExpressionType">
    <sequence>
     <choice minOccurs="2" maxOccurs="2">
    <element name="DoubleValue" type="double"/>
    <element name="LongValue" type="long"/>
    <element name="ArithmeticField" type="mpqf:FieldType"/>
    <element name="ArithmeticExpression" type="mpqf:ArithmeticExpressionType"/>
     </choice>
    </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="Abs">
 <complexContent>
  <extension base="mpqf:ArithmeticExpressionType">
   <sequence>
    <choice>
    <element name="DoubleValue" type="double"/>
    <element name="LongValue" type="long"/>
    <element name="ArithmeticField" type="mpqf:FieldType"/>
    <element name="ArithmeticExpression" type="mpqf:ArithmeticExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="Ceiling">
 <complexContent>
  <extension base="mpqf:ArithmeticExpressionType">
   <sequence>
    <choice>
    <element name="DoubleValue" type="double"/>
    <element name="LongValue" type="long"/>
    <element name="ArithmeticField" type="mpqf:FieldType"/>
    <element name="ArithmeticExpression" type="mpqf:ArithmeticExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

```
<complexType name="Floor">
 <complexContent>
  <extension base="mpqf:ArithmeticExpressionType">
   <sequence>
    <choice>
   <element name="DoubleValue" type="double"/>
   <element name="LongValue" type="long"/>
   <element name="ArithmeticField" type="mpqf:FieldType"/>
   <element name="ArithmeticExpression" type="mpqf:ArithmeticExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="Round">
 <complexContent>
  <extension base="mpqf:ArithmeticExpressionType">
   <sequence>
    <choice>
   <element name="DoubleValue" type="double"/>
   <element name="LongValue" type="long"/>
   <element name="ArithmeticField" type="mpqf:FieldType"/>
   <element name="ArithmeticExpression" type="mpqf:ArithmeticExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

11.7.3 Semantics

Semantics of the `ArithmeticExpressionType` type:

Name	*Definition*
`Add`	A complex type based on `ArithmeticExpressionType` type, for using the “Add” arithmetic operation.
`Subtract`	A complex type based on `ArithmeticExpressionType` type, for using the “Subtract” arithmetic operation.
`Multiply`	A complex type based on `ArithmeticExpressionType` type for using the “Multiply” arithmetic operation.
`Divide`	A complex type based on `ArithmeticExpressionType` type, for using the “Divide” arithmetic operation. The assumption here is that the result type of the arithmetic expression will depend on the type of the operands, following the conventions of the ANSI C implicit casting mechanism.
`Modulus`	A complex type based on `ArithmeticExpressionType` type, for using the “Modulus” arithmetic operation that returns the remainder of the integer division of the two operands.
`Abs`	A complex type based on `ArithmeticExpressionType` type, for using the “Abs” arithmetic operation that returns the absolute value of the operand.

Name	*Definition*
`Ceiling`	A complex type based on `ArithmeticExpressionType` type, for using the “Ceiling” arithmetic operation that returns the ceiling of the operand.
`Floor`	A complex type based on `ArithmeticExpressionType` type, for using the “Floor” arithmetic operation that returns the floor of the operand.
`Round`	A complex type based on `ArithmeticExpressionType` type, for using the “Round” arithmetic operation that returns the rounded value of the operand.
DoubleValue	Allows specifying a constant arithmetic value by using double type.
LongValue	Allows specifying a constant arithmetic value by using long type.
ArithmeticField	Allows specifying a path to a numeric value within the metadata of an evaluation item using FieldType type.
ArithmeticExpression	Allows specifying an expression that is required to be evaluated to an arithmetic value. The abstract type ArithmeticExpressionType is the base type of all operations with a numeric output (Add, Subtract, etc.).

11.7.4 Example

The following example illustrates the use of the arithmetic expressions in the context of comparison operators. In particular, the absolute value of a field, calculated using the “Abs” expression, is compared with a constant value.

```
<MpegQuery mpqfID="someID">
 <Query>
   <Input>
     <QueryCondition>
       <Condition xsi:type="GreaterThan">
        <ArithmeticExpression xsi:type="Abs">
         <ArithmeticField>Mpeg7/Description/AudioVisual/Semantic/SemanticBaseType/
           AttributeValuePair/IntegerValue</ArithmeticField>
         </ArithmeticExpression>
         <LongValue>40</LongValue>
       </Condition>
     </QueryCondition>
   </Input>
 </Query>
</MpegQuery>
```

11.8 Aggregate Expression Types

11.8.1 Introduction

The abstract `AggregateExpressionType` type is the base type of all the aggregate expressions. All the aggregate expressions shall be inherited from the abstract `AggregateExpressionType` type. Currently, this part of ISO/IEC 15938 specifies the following aggregate expressions: `AVG`, `StdDev`, `Variance`, `SUM`, `Count`, `MAX` and `MIN`. All aggregate expressions are only allowed to be used in the OutputDescription element and shall not be used in the `QueryCondition` element.

11.8.2 Syntax

```
<complexType name="AggregateExpressionType" abstract="true">
   <attribute name="aggregateID" type="ID" use="optional"/>
</complexType>
<complexType name="AVG">
  <complexContent>
    <extension base="mpqf:AggregateExpressionType">
      <sequence>
        <element name="Field" type="mpqf:FieldType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="StdDev">
  <complexContent>
    <extension base="mpqf:AggregateExpressionType">
      <sequence>
        <element name="Field" type="mpqf:FieldType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="Variance">
  <complexContent>
    <extension base="mpqf:AggregateExpressionType">
      <sequence>
        <element name="Field" type="mpqf:FieldType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="SUM">
  <complexContent>
    <extension base="mpqf:AggregateExpressionType">
      <sequence>
        <element name="Field" type="mpqf:FieldType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="Count">
  <complexContent>
    <extension base="mpqf:AggregateExpressionType">
      <sequence>
        <element name="Field" type="mpqf:FieldType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="MAX">
  <complexContent>
    <extension base="mpqf:AggregateExpressionType">
      <sequence>
        <element name="Field" type="mpqf:FieldType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="MIN">
```

```
  <complexContent>
    <extension base="mpqf:AggregateExpressionType">
      <sequence>
        <element name="Field" type="mpqf:FieldType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

11.8.3 Semantics

Semantics of the `AggregateExpressionType` type:

Name	*Definition*
`AggregateExpressionType`	An abstract type that is the base type for defining aggregate expressions. Types that inherit from this type are not allowed to occur in the `QueryCondition` element.
`aggregateID`	An attribute of ID type used to give an unique identifier to an aggregate.
`AVG`	A complex type based on `AggregateExpressionType` type, for calculating the average value of a certain numeric field.
`StdDev`	A complex type based on `AggregateExpressionType` type, for calculating the standard deviation value of a certain numeric field.
`Variance`	A complex type based on `AggregateExpressionType` type, for calculating the variance value of a certain numeric field.
`SUM`	A complex type based on `AggregateExpressionType` type, for calculating the sum value of a certain numeric field.
`Count`	A complex type based on `AggregateExpressionType` type, for calculating the number of items within a group. If a metadata field is specified, only items in which that field exists and does not contain an empty string will be counted.
`MAX`	A complex type based on `AggregateExpressionType` type, for calculating the maximum value of a certain numeric field.
`MIN`	A complex type based on `AggregateExpressionType` type, for calculating the minimum value of a certain numeric field.

11.8.4 Example

Significant examples are provided at the `GroupBy` element.

11.9 String Expression Types

11.9.1 Introduction

The `StringExpressionType` type is an abstract type, which is the base type of all the string expressions. All the string expressions shall extend the abstract type `StringExpressionType`. This part of ISO/IEC 15938 specifies the following string expressions: `LowerCase`, `UpperCase`. Their purpose is to facilitate case insensitive processing of `StringOperands`.

11.9.2 Syntax

```
<complexType name="UpperCase">
 <complexContent>
  <extension base="mpqf:StringExpressionType">
   <sequence>
    <choice>
   <element name="StringValue" type="string"/>
   <element name="StringField" type="mpqf:FieldType"/>
   <element name="StringExpression" type="mpqf:StringExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="LowerCase">
 <complexContent>
  <extension base="mpqf:StringExpressionType">
   <sequence>
    <choice>
   <element name="StringValue" type="string"/>
   <element name="StringField" type="mpqf:FieldType"/>
   <element name="StringExpression" type="mpqf:StringExpressionType"/>
    </choice>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

11.9.3 Semantics

Semantics of the `StringExpressionType` type:

Name	*Definition*
`UpperCase`	A complex type based on `StringExpressionType` type, for transforming a string to upper case.
`LowerCase`	A complex type based on `StringExpressionType` type, for transforming a string to lower case.
StringValue	Allows specifying a constant string value by using string type.
StringField	Allows specifying a path to a string value within the metadata of an evaluation item, by using FieldType type.
StringExpression	Allows specifying an expression evaluating to a string value. The abstract type StringExpressionType is the base type, from which all the operations with a string output are inherited.

11.9.4 Example

The following example illustrates the use of the expressions in the context of comparison operators. In particular, the upper case value of a constant is compared to a field.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <Condition xsi:type="Equal">
          <StringField typeName="CreationType">Title</StringField>
          <StringExpression xsi:type="LowerCase">
            <StringValue>Blade Runner</StringValue>
          </StringExpression>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

11.10 Semantic Expression Types

11.10.1 Introduction

11.10.2 Syntax

```
<complexType name="SemanticExpressionType" abstract="true">
  <complexContent>
   <extension base="mpqf:BooleanExpressionType">
     <attribute name="anchorDistance"
                   type="nonNegativeInteger" use="required" />
     <attribute name="anchor" type="boolean" use="optional" />
   </extension>
  </complexContent>
</complexType>

<complexType name="SubClassOf">
  <complexContent>
   <extension base="mpqf:SemanticExpressionType">
      <attribute name="var" type="string"/>
      <attribute name="class" type="string"/>
   </extension>
  </complexContent>
</complexType>
<complexType name="TypeOf">
  <complexContent>
   <extension base="mpqf:SemanticExpressionType">
      <attribute name="var" type="string"/>
      <attribute name="class" type="string"/>
    </extension>
  </complexContent>
</complexType>
<complexType name="EquivalentClass">
  <complexContent>
   <extension base="mpqf:SemanticExpressionType">
      <attribute name="var" type="string"/>
```

```
        <attribute name="class" type="string"/>
    </extension>
  </complexContent>
</complexType>
<complexType name="ComplementOf">
  <complexContent>
    <extension base="mpqf:SemanticExpressionType">
        <attribute name="var" type="string"/>
        <attribute name="class" type="string"/>
    </extension>
  </complexContent>
</complexType>
<complexType name="IntersectionOf">
  <complexContent>
    <extension base="mpqf:SemanticExpressionType">
        <attribute name="var" type="string"/>
        <attribute name="class" type="string"/>
    </extension>
  </complexContent>
</complexType>
<complexType name="UnionOf">
  <complexContent>
    <extension base="mpqf:SemanticExpressionType">
        <attribute name="var" type="string"/>
        <attribute name="class" type="string"/>
    </extension>
  </complexContent>
</complexType>
<complexType name="InverseOf">
  <complexContent>
    <extension base="mpqf:SemanticExpressionType">
        <attribute name="var" type="string"/>
        <attribute name="class" type="string"/>
    </extension>
  </complexContent>
</complexType>
<complexType name="DisjointWith">
  <complexContent>
    <extension base="mpqf:SemanticExpressionType">
        <attribute name="var" type="string"/>
        <attribute name="class" type="string"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="SemanticRelation">
  <complexContent>
    <extension base="mpqf:SemanticExpressionType">
      <sequence>
        <element name="Subject" type="string"/>
        <element name="Property" type="string"/>
        <element name="Object" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

11.10.3 Semantics

Semantics of the `SemanticExpressionType`:

Name	*Definition*
`SemanticExpressionType`	An abstract type based on the BooleanExpressionType type that is the base type for defining semantic expressions.
`anchorDistance`	Defines the number of hops the identified triple is reachable through the RDF graph from the given starting RDF triple. The starting RDF triple is indicated by enabling the anchor flag. An anchorDistance of 0 indicates that the addressed node can be within any distance to the anchor node.
`Anchor`	Defines a marker for a RDF triple within a query graph which serves as source for other triples. The main idea is to simplify the generation of semantic queries and to allow optimizers to improve resulting SPARQL queries.

Semantics of the `class specific` types:

Name	*Definition*
`SubClassOf`	A complex type based on the SemanticExpressionType type for defining a sub class relationship. This operation has two attributes that are required to represent the following pattern: var SubClassOf class. The var and class parameter can be URIs or place holder based on the SPARQL notation (starting with a question mark and a descriptive identifier).
`TypeOf`	A complex type based on the SemanticExpressionType type for defining a type of relationship. The attributes are defined as for the SubClassOf operation.
`EquivalentClass`	A complex type based on the SemanticExpressionType type for defining an equivalent class relationship. The attributes are defined as for the SubClassOf operation.
`ComplementOf`	A complex type based on the SemanticExpressionType type for defining a complement of relationship. The attributes are defined as for the SubClassOf operation.
`IntersectionOf`	A complex type based on the SemanticExpressionType type for defining an intersection of relationship. The attributes are defined as for the SubClassOf operation.
`UnionOf`	A complex type based on the SemanticExpressionType type for defining a union of relationship. The attributes are defined as for the SubClassOf operation.
`InverseOf`	A complex type based on the SemanticExpressionType type for defining an inverse of relationship. The attributes are defined as for the SubClassOf operation.

Name	*Definition*
`DisjointWith`	A complex type based on the SemanticExpressionType type for defining a disjoint with relationship. The attributes are defined as for the SubClassOf operation.

Semantics of the `SemanticRelationType` type:

Name	*Definition*
`SemanticRelation`	Specifies the representation of a semantic relation within the query format. A semantic relation describes a triple in the RDF data model. The syntax is similar to the SPARQL query language and starts with a question mark and a descriptive identifier or a specific URI for the subject and object as well as a respective value representing the property.
`Subject`	Defines a place holder for the subject of a RDF triple. The place holder must start with a question mark and a descriptive identifier or a specific URI.
`Property`	Defines the predicate (RDF notation) or property (OWL notation) of a RDF triple. The name space can be predefined in the QFDeclaration section.
`Object`	Defines a place holder for the object of a RDF triple. The place holder must start with a question mark and a descriptive identifier or a specific URI.

11.10.4 Usage guidance

11.10.4.1 General

This subclause is an informative section.

11.10.4.2 Linking structural (XML) and semantic data (RDF/XML)

Multimedia annotation comes in a large diversity. On the one side XML based metadata have been introduced (MPEG-7, etc.) which have strengthens in expressing structural information of the described multimedia content. For instance, low level features (like color descriptors) or temporal (spatial) behavior (video segments, image regions) are described by XML based annotations. On the other side, there are RDF based descriptions which support the annotation of high-level and semantic related annotations. Both concepts have advantages in their specific domain and drawbacks if they are used in the opposite domain (e.g., RDF for structural design and XML for semantic design).

To address both models in one search request, a linking between the two worlds need to be provided. The following paragraph describes such a possible linking for supporting MPQF requests that target on a global data set.

In order to accomplish a linkage between the extracted semantic concepts and its structured counterparts the following definition is introduced:

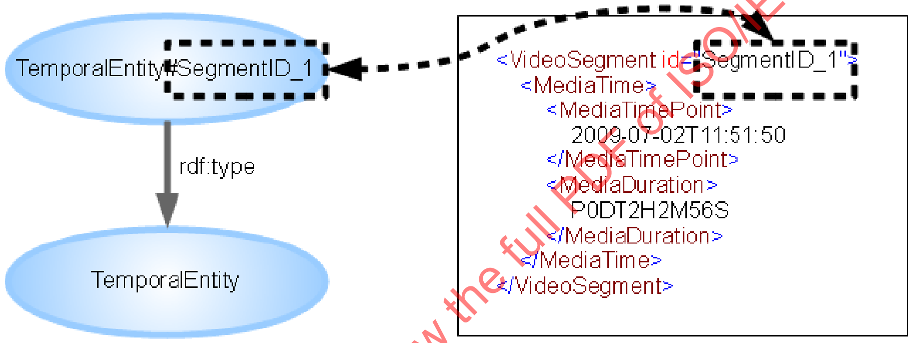
11.10.4.3 Definition

Let A = CONCEPT#value the segment id value of a corresponding semantic concept expressing e.g. a temporal event description and B = //ELEMENT [@ID = value] the segment ID extracted via the given XPath expression of the structured description (e.g., the VideoSegment of an MPQF-7 description). Then, the linkage within the hybrid data model can be derived by the following Equi-Join:

$$E \otimes_{A=B} VS := \{e \in E \wedge vs \in VS \wedge e_A = vs_B\}$$

where E and V S are the semantic and structured annotations, respectively.

The definition foresees a unique key system that is accomplished by instantiating semantic and structured information (see Figure 3). Here, at the semantic side a unique URI containing a fragment identifier [4] can be attached to every instantiation of a concept (e.g., TemporalEntity). For instance at a TemporalEntity instance the following URI may be used: http://domain.tld/TemporalEntity#Segment_ID_1. Then, at the respective structured counterpart, an ID attribute is attached containing the same identifier value. In our example the MPEG-7 VideoSegment element would be instantiated with the following attribute id=SegmentID_1. By evaluating the given definition with value = SegmentID_1, the linkage of the corresponding information can be solved.



```
<VideoSegment id="SegmentID_1">
  <MediaTime>
    <MediaTimePoint>
      2009-07-02T11:51:50
    </MediaTimePoint>
    <MediaDuration>
      P0DT2H2M56S
    </MediaDuration>
  </MediaTime>
</VideoSegment>
```

Figure 3 — Linking Semantic and XML Data

11.10.4.4 EvaluationPath

The EvaluationPath allows focusing the scope of the query evaluation to a specific subset. For instance by using the MPEG-7 standard, one is able to concentrate on either Videos or VideoSegments during the evaluation. The scope is represented by an XPath pointing to the desired element e.g., //Video or //VideoSegment.

The use of the EvaluationPath for semantic retrieval is implicitly defined by the linkage to XML based metadata. Although it is possible to use XPath for RDF/XML data, it is not recommended as different mappings would result in different XPath expressions. By this, as defined, the focus should set by targeting on the structured data which is indented to provide a linking to the semantic descriptions. However, note, the exact linking between structured and semantic data is an implementation issue.

Furthermore, there are additional restrictions of the EvaluationPath already existing in the current version of the MPEG Query Format. First, the EvaluationPath is only as useful as an available XML Schema (the metadata description of the multimedia data) provides a deep hierarchy and deep descriptive level. In case there is only a flat description or no XML Schema available, the EvaluationPath is hardly beneficial. In addition, there also exist query types (e.g., QueryByMedia) where an EvaluationPath has no further meaning. A query using a single QueryByMedia condition targeting to an example image might not use the EvaluationPath at all.

11.10.4.5 Evaluation of semantic conditions

The amendment for semantic enhancement to MPQF comes with two different ways of expressing semantic conditions. First of all, there is a QueryBySPARQL query condition which supports expressing SPARQL expressions. Those expressions are denoted to be executed by SPARQL engines, where the content of the QueryBySPARQL is restricted to an ASK SPARQL query. The evaluation processes (similar to the QueryByXQuery condition) the occurrence (matching) of the modeled subgraph in the RDF repository and results in values on a true/false basis.

Second, semantic filtering can be expressed by a set of semantic relations, expressions and comparison expressions enhanced by semantic means. This approach follows the same concept as the introduction of the Comparison expressions. Those expressions have been introduced in case that no full XQuery engine is available. In the same manner, the semantic relations can be used for the semantic case. In sum the semantic relations need to describe a subgraph for filtering against the semantic data set.

11.10.5 Example

The following example illustrates the use of semantic expressions on an imaginary semantic annotation in a surveillance system scenario. The example searches for persons and its location and time information (see OutputDescription) that matches the drawn RDF subgraph (see condition part). The example demonstrates the use of the integrated anchor and anchorDistance attributes. Here the first SemanticRelation-Condition defines the anchor meaning that all other specified conditions build the search graph pattern related to their given anchorDistance. The RDF subgraph assumes as data model the annotation of a closed (all persons are known and can be tracked by specific sensors) video surveillance scenario containing locations (region of interests (ROI) or rooms) which are monitored by cameras. At runtime of the system, automatic events are triggered when persons are entering rooms or specific areas (ROI, e.g. a coffee area). In this context, the RDF subgraph and therefore the query searches for all persons that have been monitored entering a specific ROI which is controlled by a camera located in *Room240*. Besides the use of the anchor and anchorDistance attributes supports the ease of query creation by specifying one SemanticRelation as root (anchor is set to true) and all other relations are located within the distance specified by the anchorDistance flag. On the one side a user does not need to specify all relations and on the other side a possible query optimizer is able to produce an optimized subgraph. However, it has to be noted that query optimization is an implementation issue and that the semantic of the query has to be evaluated.

```
<MpegQuery mpqfID="">
  <Query>
    <Input>
      <QFDeclaration>
        <Prefix name="xsd"
                    uri="http://www.w3.org/2001/XMLSchema#"/>
        <Prefix name="rdf"
                    uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
        <Prefix name="ontology"
                    uri="http://www.example.de/ontology#"/>
        <Prefix name="config"
                    uri="http://www.example.de/configuration#"/>
      </QFDeclaration>
      <OutputDescription>
        <ReqSemanticField>?person</ReqSemanticField>
        <ReqSemanticField>?location</ReqSemanticField>
        <ReqSemanticField>?temp</ReqSemanticField>
      </OutputDescription>
      <QueryCondition>
        <Condition xsi:type="AND">
          <Condition xsi:type="SemanticRelation" anchor="true"
anchorDistance="0">
            <Subject>?person</Subject>
            <Property>ontology:hasName</Property>
            <Object>?name</Object>
          </Condition>
          <Condition xsi:type="SemanticRelation" anchorDistance="3">
            <Subject>?cam</Subject>
            <Property>ontology:isLocated</Property>
            <Object>?location</Object>
          </Condition>
          <Condition xsi:type="SemanticRelation" anchorDistance="3">
            <Subject>?location</Subject>
            <Property>ontology:hasName</Property>
            <Object>"Room240"^^xsd:string</Object>
          </Condition>
        </Condition>
      </QueryCondition>
    </Input>
  </Query> </MpegQuery>
```

12 Query Types

12.1 Introduction

The `QueryType` type is an abstract type which extends the `BooleanExpressionType` type and becomes the parent class of all the query types including the `QueryByMedia`, `QueryByDescription`, `QueryByFeatureRange`, `SpatialQuery`, `TemporalQuery`, `QueryByXQuery`, `QueryByFreeText`, and `QueryByRelevanceFeedback` types. All the subtypes inherited from the `QueryType` type shall return a value in the range of [0..1], denoting whether an evaluation item satisfies the condition specified by the operation (e.g. `QueryByMedia`) or not.

12.2 Syntax

```
<complexType name="QueryType" abstract="true">
  <complexContent>
    <extension base="mpqf:BooleanExpressionType"/>
  </complexContent>
</complexType>
```

12.3 Semantics

Semantics of the `QueryType` type:

Name	*Definition*
`QueryType`	Specifies an abstract type that extends the `BooleanExpressionType` type to be the base type of all the query types such as `QueryByMedia` or `QueryByFreeText`.

12.4 QueryByMedia

12.4.1 Introduction

The `QueryByMedia` type extends the abstract `QueryType` type. The `QueryByMedia` type enables a requester to realize a search based on a given example media resource. It provides also an attribute to indicate the search criteria regarding similar-match or exact-match.

12.4.2 Syntax

```
<complexType name="QueryByMedia">
  <complexContent>
    <extension base="mpqf:QueryType">
      <choice>
        <element name="MediaResource" type="mpqf:MediaResourceType"/>
        <element name="MediaResourceREF" type="IDREF"/>
      </choice>
      <attribute name="matchType" use="optional" default="similar">
        <simpleType>
          <restriction base="string">
            <enumeration value="similar"/>
            <enumeration value="exact"/>
          </restriction>
        </simpleType>
      </attribute>
<attribute name=" TargetMediaPath" use="optional"
       type="mpqf:xPathType"/>
    </extension>
  </complexContent>
</complexType>
```

12.4.3 Semantics

Semantics of the `QueryByMedia` type:

Name	*Definition*
`QueryByMedia`	Specifies the tool for a search by example, which allows the requester to perform a similarity or exact match search based on a given multimedia content. This type extends the `QueryType` type and allows embedding either a `MediaResource` or a reference to a resource defined in the `QFDeclaration` element.
`MediaResource`	Specifies the actual resource that is used for querying. Is of `MediaLocatorType` type.
`MediaResourceRef`	Specifies a reference to the resource defined in the `QFDeclaration` element that is used for querying. Is of `IDREF` type. The referenced resource shall be of type MediaResourceType.
`matchType`	Specifies a search type of match (either a similar or an exact match).
`TargetMediaPath`	Allows specifying a relative XPath expression informing about which field of the metadata fragment related to the evaluation item being addressed contains the URI of the media against which the condition should be evaluated.

12.4.4 Example

The following example illustrates the use of the `QueryByMedia` type. The query requests video contents which are similar to the given image.

```
<MpegQuery mpqfID="someID">
 <Query>
```

```
    <Input>
     <QueryCondition>
      <TargetMediaType>video/mpeg</TargetMediaType>
       <Condition xsi:type="QueryByMedia" matchType="similar">
        <MediaResource resourceID="Image001">
          <MediaResource>
           <MediaUri>http://db.mpqf.mpeg/testdata001.jpg</MediaUri>
          </MediaResource>
        </MediaResource>
       </Condition>
      </QueryCondition>
     </Input>
    </Query>
  </MpegQuery>
```

12.5 QueryByDescription

12.5.1 Introduction

The `QueryByDescription` type extends the abstract `QueryType` type. The `QueryByDescription` type enables a requester to realize a search based on a given example description. Any description, i.e. metadata can be embedded if the metadata is based on XML-schema and the description is conformant to the schema. It provides also an attribute to indicate the search criteria regarding similar-match or exact-match.

12.5.2 Syntax

```
<complexType name="QueryByDescription">
  <complexContent>
    <extension base="mpqf:QueryType">
      <choice>
        <element name="DescriptionResource" type="mpqf:DescriptionResourceType"/>
        <element name="DescriptionResourceREF" type="IDREF"/>
      </choice>
      <attribute name="matchType" use="optional" default="similar">
        <simpleType>
          <restriction base="string">
            <enumeration value="similar"/>
            <enumeration value="exact"/>
          </restriction>
        </simpleType>
      </attribute>
<attribute name=" TargetMediaPath" use="optional"
       type="mpqf:xPathType"/>
    </extension>
  </complexContent>
</complexType>
```

12.5.3 Semantics

Semantics of the `QueryByDescription` type:

Name	*Definition*
`QueryByDescription`	Specifies the tool for a search by example, which allows the requester to perform a similarity or exact match search based on a given description.

Name	*Definition*
	This type extends the QueryType type and allows embedding either a DescriptionResource or a reference to a resource defined in the QFDeclaration.
DescriptionResource	Specifies the actual resource that is taken for querying. The resource shall be the description of a low or high level feature according to the used metadata schema.
DescriptionResourceRef	Specifies a reference to the resource defined in the QFDeclaration that is taken for querying. The referenced resource shall be of type DescriptionResourceType.
matchType	Specifies a search type of match (either a similar or an exact match).
TargetMediaPath	Allows specifying a relative XPath expression informing about which field of the metadata fragment related to the evaluation item being addressed contains the URI of the media against which the condition should be evaluated.

12.5.4 Example

The following example illustrates the use of the QueryByDescription type. The query requests images whose MPEG-7 metadata has the CreationInformation/Creation/Title element with the value “Miracle Query Format” exactly.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <TargetMediaType>image/jpeg</TargetMediaType>
        <Condition xsi:type="QueryByDescription" matchType="exact">
          <DescriptionResource resourceID="desc001">
            <AnyDescription xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
               xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
              <mpeg7:Mpeg7>
                <mpeg7:DescriptionUnit xsi:type="mpeg7:CreationInformationType">
                  <mpeg7:Creation>
                    <mpeg7:Title>Miracle Query Format</mpeg7:Title>
                  </mpeg7:Creation>
                </mpeg7:DescriptionUnit>
              </mpeg7:Mpeg7>
            </AnyDescription>
          </DescriptionResource>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

12.6 QueryByFreeText

12.6.1 Introduction

The QueryByFreeText type enables the user to perform a free text search, i.e. a search based on the use of free-text. The QueryByFreeText type extends the QueryType type and contains either a mandatory

FreeText element containing text description or a mandatory RegExp element containing a regular expression as a condition. It has also an optional choice of fields, which allow the user to state (in the SearchField elements) if the search should be performed in specific elements only or if specific elements should be ignored during the search (stated in the IgnoreField elements). If no SearchField or IgnoreField elements are specified, the search will take the whole description into account.

12.6.2 Syntax

```
<complexType name="QueryByFreeText">
  <complexContent>
    <extension base="mpqf:QueryType">
      <sequence>
         <choice>
          <element name="FreeText" type="string"/>
          <element name="RegExp" type="string"/>
         </choice>
         <choice minOccurs="0">
          <element name="SearchField"
                          type="mpqf:FieldType" maxOccurs="unbounded"/>
          <element name="IgnoreField"
                          type="mpqf:FieldType" maxOccurs="unbounded"/>
        </choice>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

12.6.3 Semantics

Semantics of the QueryByFreeTextType type:

Name	*Definition*
QueryByFreeText	Describes a search of multimedia content based on the given text. This type extends the QueryType type and contains an element of string to describe the search condition.
FreeText	Specifies the actual free text term describing the multimedia content of desire.
RegExp	Specifies a regular expression describing the multimedia content of desire.
SearchField	Specifies the element(s) on which the search shall be performed.
IgnoreField	Specifies the element(s) that shall be ignored during the search.

12.6.4 Examples

The following example illustrates the use of the QueryByFreeText type, without any SearchField and IgnoreField elements.

```
<MpegQuery mpqfID="someID">
  <Query>
```

```
    <Input>
      <QueryCondition>
        <Condition xsi:type="QueryByFreeText">
          <FreeText>Testquery</FreeText>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

The example below modifies the previous one so as to search within the `TextAnnotation` elements of the descriptions only.

```
<MpegQuery mpqfID="someID">
 <Query>
  <Input>
    <QueryCondition>
      <Condition xsi:type="QueryByFreeText">
        <FreeText>
          Testquery
        </FreeText>
        <SearchField>Mpeg7/Description/AudioVisual/TextAnnotation</SearchField>
      </Condition>
    </QueryCondition>
  </Input>
 </Query>
</MpegQuery>
```

The following example modifies the first one so as to ignore the `TextAnnotationValue` elements of the descriptions.

```
<MpegQuery mpqfID="someID">
 <Query>
  <Input>
    <QueryCondition>
      <Condition xsi:type="QueryByFreeText">
        <FreeText>
          Testquery
        </FreeText>
        <IgnoreField>
          Mpeg7/Description/AudioVisual/Semantic/SemanticBaseType/
AttributeValuePair/TextAnnotationValue
        </IgnoreField>
      </Condition>
    </QueryCondition>
  </Input>
 </Query>
</MpegQuery>
```

12.7 QueryByFeatureRange

12.7.1 Introduction

The `QueryByFeatureRange` type extends the abstract `QueryType` type. It enables a requester to realize two different range searches. The first option enables a range search based on given descriptions denoting the start and the end range of the retrieval area. The second option allows a range search based on a given description (e.g. an audio low-level feature as AudioSpectrumFlatness) and a distance. The distance element denotes the distance in feature space from the center of mass, whereas the center is described by the `Center` element. The example description shows the usage of this descriptor in more detail. In the first example, the first option is described by the `AnyDescription` element within the `QFDeclaration` element and referenced within the `QueryByFeatureRange` type by the `RangeStart` element and `RangeEnd` element, respectively. The second option is described in the second example by an `AnyDescription` element within the `QFDeclaration` element and referenced within the `QueryByFeatureRange` by `Center and a distance.`

12.7.2 Syntax

```
<complexType name="QueryByFeatureRange">
  <complexContent>
    <extension base="mpqf:QueryType">
      <sequence>
        <choice>
          <element name="Range">
            <complexType>
              <attribute name="RangeStart" type="IDREF" use="required"/>
              <attribute name="RangeEnd" type="IDREF" use="required"/>
            </complexType>
          </element>
          <element name="Distance">
            <complexType>
              <attribute name="Center" type="IDREF" use="required"/>
              <attribute name="Radius" type="mpqf:zeroToOneType" use="required"/>
              <attribute    name="DistanceFunction"    type="mpqf:SimpleTermType"
use="optional"/>
            </complexType>
          </element>
        </choice>
      </sequence>
<attribute name=" TargetMediaPath" use="optional"
       type="mpqf:xPathType"/>
    </extension>
  </complexContent>
</complexType>
```

12.7.3 Semantics

Semantics of the `QueryByFeatureRange` type:

Name	***Definition***
`QueryByFeatureRange`	Specifies the `QueryByFeatureRange` type which is an extension of the `QueryType` type. This type allows the definition of range queries by defining the start and end range of the desired search region.
`Range`	Describes the first option for the range search. Here the query indicates a start and an end point of the range and the search engine shall return items between these points.
`RangeStart`	Specifies the start of a range and is required to reference to a feature within the `QFDeclaration` element. Both, `rangeStart` and `rangeEnd` has to point to the same type of description.

Name	***Definition***
RangeEnd	Specifies the end of a range and is required to reference to a feature within the QFDeclaration element. Both, rangeStart and rangeEnd have to point to the same type of description.
Distance	Describes the second mode of the QueryByFeatureRange type. It is specified by a center and a distance, whereas all items from the center within a certain distance around shall be returned. Since the feature space is dependend on the distance function, the DistanceFunction attribute refers to a classification scheme term specifying the distance function.
Center	Specifies the center of the range search in the feature space and is required to reference to a feature within the QFDeclaration element.
Radius	Specifies the distance from the center to the furthermost distant point in the feature space which shall be considered for the result. This element contains a floating value, which is between zero and one.
DistanceFunction	The distance from the center to the furthermost distant point in the feature space is dependent on a distance function (e.g. Eucledian distance). In order to signal the required distance function to the server, the TermType type is used, which refers to a classification scheme. If the distance function is not further specified, always the Eucledian distance is adopted.
TargetMediaPath	Allows specifying a relative XPath expression informing about which field of the metadata fragment related to the evaluation item being addressed contains the URI of the media against which the condition should be evaluated.

12.7.4 Example

The following example illustrates the use of the QueryByFeatureRange type in the first mode. To indicate the range, two pieces of DominantColor instance are declared within QFDeclaration element. The query requests any contents whose DominantColor feature is in the specified range.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QFDeclaration>
        <Resource xsi:type="DescriptionResourceType" resourceID="startID">
          <AnyDescription xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
            <mpeg7:Mpeg7>
              <mpeg7:DescriptionUnit xsi:type="mpeg7:DominantColorType">
                <mpeg7:ColorSpace type="RGB"/>
                <mpeg7:SpatialCoherency>28</mpeg7:SpatialCoherency>
                <mpeg7:Value>
                  <mpeg7:Percentage>12</mpeg7:Percentage>
                  <mpeg7:Index>1 1 1</mpeg7:Index>
                  <mpeg7:ColorVariance>1 1 1</mpeg7:ColorVariance>
                </mpeg7:Value>
              </mpeg7:DescriptionUnit>
            </mpeg7:Mpeg7>
          </AnyDescription>
        </Resource>
        <Resource xsi:type="DescriptionResourceType" resourceID="endID">
          <AnyDescription xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
            <mpeg7:Mpeg7>
              <mpeg7:DescriptionUnit xsi:type="mpeg7:DominantColorType">
                <mpeg7:ColorSpace type="RGB"/>
                <mpeg7:SpatialCoherency>28</mpeg7:SpatialCoherency>
                <mpeg7:Value>
                  <mpeg7:Percentage>30</mpeg7:Percentage>
                  <mpeg7:Index>5 5 6</mpeg7:Index>
                  <mpeg7:ColorVariance>1 1 1</mpeg7:ColorVariance>
                </mpeg7:Value>
              </mpeg7:DescriptionUnit>
            </mpeg7:Mpeg7>
          </AnyDescription>
        </Resource>
      </QFDeclaration>
      <QueryCondition>
        <Condition xsi:type="QueryByFeatureRange">
          <Range RangeEnd="endID" RangeStart="startID"/>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

The following example illustrates the use of the `QueryByFeatureRange` type in the second mode by specifying a center and a distance. To indicate this range, a piece of a `DominantColor` instance is declared within the `QFDeclaration` element and a distance is given. The query requests any contents whose `DominantColor` feature is in a distance of 0.5.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QFDeclaration>
        <Resource xsi:type="DescriptionResourceType" resourceID="centerID">
          <AnyDescription xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
            <mpeg7:Mpeg7>
              <mpeg7:DescriptionUnit xsi:type="mpeg7:DominantColorType">
                <mpeg7:ColorSpace type="RGB"/>
                <mpeg7:SpatialCoherency>28</mpeg7:SpatialCoherency>
                <mpeg7:Value>
                  <mpeg7:Percentage>12</mpeg7:Percentage>
                  <mpeg7:Index>1 1 1</mpeg7:Index>
                  <mpeg7:ColorVariance>1 1 1</mpeg7:ColorVariance>
                </mpeg7:Value>
              </mpeg7:DescriptionUnit>
            </mpeg7:Mpeg7>
          </AnyDescription>
        </Resource>
      </QFDeclaration>
      <QueryCondition>
        <Condition xsi:type="QueryByFeatureRange">
          <Distance Radius="0.5" Center="centerID"/>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

12.8 SpatialQuery

12.8.1 Introduction

This query type allows requests in the spatial domain where one or two regions (e.g., MPEG-7 StillRegion, etc.) are involved. Relationships among those regions and possible matching regions are expressed by the RelationType type. Note, that the query type allows two different RelationType types. One is the SpatialRelation CS and the other is the BaseRelation CS (see Annex B). The RelationType type provides three attributes. The relationType attribute is mandatory and denotes the relation. The sourceResource attribute is also mandatory and indicates a specific region. Furthermore, another optional attribute targetResource is provided to indicate another specific region. Combining these attributes allows a requester to generate various queries involving spatial relation of regions.

12.8.2 Syntax

```
<complexType name="SpatialQuery">
  <complexContent>
    <extension base="mpqf:QueryType">
      <sequence>
        <element name="SpatialRelation"
               type="mpqf:RelationType"/>
      </sequence>
<attribute name=" TargetMediaPath" use="optional"
       type="mpqf:xPathType"/>
    </extension>
  </complexContent>
</complexType>
```

12.8.3 Semantics

Semantics of the `SpatialQuery` type:

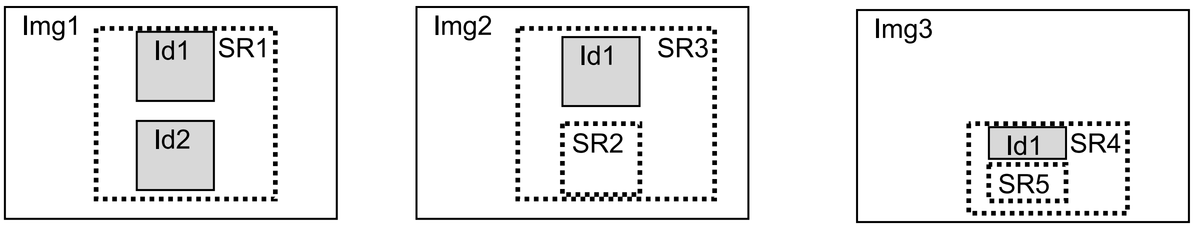
Name	*Definition*
`SpatialQuery`	Specifies the `SpatialQuery` type which is an extension of the `QueryType` type. It allows the retrieval of spatial elements within media objects by defining `SpatialRelation` element. The spatial element to be retrieved is decided by `EvaluationPath` element in `QueryCondition` element.
`SpatialRelation`	Specifies the spatial relation based on the `RelationType` type. It provides means for describing a spatial relation among segment decompositions. Allowed relations are defined in SpatialRelation CS and BaseRelation CS.
`TargetMediaPath`	Allows specifying a relative XPath expression informing about which field of the metadata fragment related to the evaluation item being addressed contains the URI of the media against which the condition should be evaluated.

12.8.4 Example

The following illustrates some scenarios using the ‘north’-related relation. Three relations for ‘north’ are defined in SpatialRelation CS, i.e., ‘north,’ ‘northOf’ and ‘northIn.’ The ‘north’ is used only when both `sourceResource` and `targetResource` attributes are defined, and then the region that includes two defined sub-regions satisfying the specified relationship is returned. The return region varies based on the `EvaluationPath` element. In the case, the `EvaluationPath` element is //Image, all images in the database that satisfy the condition are returned. If the `EvaluationPath` element is //StillRegion sub-regions of images which satisfy the condition are returned.

On the other hand, the relations ‘northOf’ and ‘northIn’ are used only when the `targetResource` attribute is omitted. In the case of a ‘northOf’-relation and an `EvaluationPath` element that indicates an image (i.e., //Image,) the image containing the `sourceResource` attribute is north of something (a sub-region) is returned. If the `EvaluationPath` element indicates a sub-region (i.e., //StillRegion,) the sub-region that is south of `sourceResource` attribute is returned.

In the case of a ‘northIn’-relation, the region containing the `sourceResource` attribute in the north of it is returned. In the example, if the `EvaluationPath` element is //Image, Img1 and Img2 are returned. Although Img3 contains Id1, its location is not in the north of Img3, so Img3 is not returned. In the case that the `EvaluationPath` element is //StillRegion, SR1, SR3 and SR4 are returned.



Expression \ EvaluationPath=	//Image	//StrillRegion
<SpatialRelation sourceResouce="id1" targetResource="id2" relationType="north">	Img1	SR1
<SpatialRelation sourceResource="id1" relationType="north-Of">	Img1,2	SR2,5
<SpatialRelation sourceResource="id1" relationType="north-In">	Img1,2	SR1,3,4

Note that the size of the retrieved sub region (SR1, 2, 3, 4) is left to implementor.

The following example illustrates the use of the `SpatialQueryType` type, in which `sourceResource` and `relationType` attributes are specified.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QFDeclaration>
        <Resource resourceID="stillImage1" xsi:type="DescriptionResourceType">
          <AnyDescription xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
            <mpeg7:Mpeg7>
              <mpeg7:DescriptionUnit xsi:type="mpeg7:StillRegionType">
                <mpeg7:VisualDescriptor xsi:type="mpeg7:DominantColorType">
                  <mpeg7:ColorSpace type="RGB"/>
                  <mpeg7:SpatialCoherency>30</mpeg7:SpatialCoherency>
                  <mpeg7:Value>
                    <mpeg7:Percentage>12</mpeg7:Percentage>
                    <mpeg7:Index>1 1 1</mpeg7:Index>
                    <mpeg7:ColorVariance>1 0 0</mpeg7:ColorVariance>
                  </mpeg7:Value>
                </mpeg7:VisualDescriptor>
                <mpeg7:VisualDescriptor xsi:type="mpeg7:HomogeneousTextureType">
                  <mpeg7:Average>1</mpeg7:Average>
                  <mpeg7:StandardDeviation>1</mpeg7:StandardDeviation>
                  <mpeg7:Energy>1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
                             18 19 20 21 22 23 24 25 26 27 28 29
30</mpeg7:Energy>
                </mpeg7:VisualDescriptor>
              </mpeg7:DescriptionUnit>
            </mpeg7:Mpeg7>
          </AnyDescription>
        </Resource>
      </QFDeclaration>
      <QueryCondition>
        <EvaluationPath>//Image</EvaluationPath>
        <Condition xsi:type="SpatialQuery">
          <SpatialRelation sourceResource="stillImage1"
relationType="urn:mpeg:mpqf:cs:SpatialRelationCS:2008:northwest"/>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

12.9 TemporalQuery

12.9.1 Introduction

This query type allows requests in the temporal domain where one or two segments (e.g., `VideoSegment`, `AudioVisualSegment`, etc.) are involved. Relationships among those segments and possible matching segments are expressed by the `RelationType` type. Note, that the query type only allows terms of the TemporalRelation CS (see Annex B). The `RelationType` type provides three attributes. The `relationType` attribute is mandatory and denotes the relation. The `sourceResource` attribute is also mandatory and indicates a specific segment. Furthermore, another optional attribute `targetResource` is provided to indicate another specific segment. Combining these attributes allows a requester to generate various queries involving temporal relation of segments.

12.9.2 Syntax

```
<complexType name="TemporalQuery">
  <complexContent>
    <extension base="mpqf:QueryType">
      <sequence>
        <element name="TemporalRelation" type="mpqf:RelationType"/>
      </sequence>
<attribute name=" TargetMediaPath" use="optional"
        type="mpqf:xPathType"/>
    </extension>
  </complexContent>
</complexType>
```

12.9.3 Semantics

Semantics of the `TemporalQuery` type:

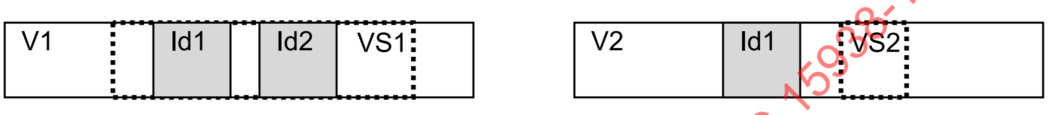
Name	***Definition***
`TemporalQuery`	Specifies the `TemporalQuery` type which is an extension of the `QueryType` type. It allows the retrieval of temporal elements within media objects by defining `TemporalRelation` element. The temporal element to be retrieved is decided by `EvaluationPath` element in `QueryCondition`.
`TemporalRelation`	Specifies the temporal relation based on the `RelationType` type. It provides means for describing a temporal relation among segment decompositions. Allowed classification scheme is TemporalRelation CS.
`TargetMediaPath`	Allows specifying a relative XPath expression informing about which field of the metadata fragment related to the evaluation item being addressed contains the URI of the media against which the condition should be evaluated.

12.9.4 Example

The following illustrates the two possible combinations of attributes in the `TemporalRelation` element and its expected results. There are two videos named ‘v1’ and ‘v2’ and two pre-declared segments named ‘id1’ and 'id2.' The table below shows what would be retrieved with the combination of attributes and EvaluationPaths.

The following illustrates some scenarios using the 'precedes'-related relation. Two relations for 'precedes' are defined in TemporalRelationCS, i.e., 'precedes' and 'precedesOf'. The 'precedes' is used only when both sourceResource and targetResource attributes are defined, and then the region that includes two defined sub-regions satisfying the specified relationship is returned. The return region varies based on the value of the EvaluationPath element. In the example, in the case the EvaluationPath element is //Video, all videos in the database that satisfy the condition are returned. If the EvaluationPath element is //VideoSegment, video segments which satisfy the condition are returned.

On the other hand, the relations 'precedesOf' is used only when the targetResource attribute is omitted. If the EvaluationPath element indicates a video (i.e., //Video,) the video containing the sourceResource attribute preceding something (a video segment) is returned. If the EvaluationPath element indicates a sub-segment (i.e., //VideoSegment,) the sub-segment that follows the sourceResource attribute is returned.



EvaluationPath= / Expression	//Video	//VideoSegment
<TemporalRelation sourceResouce="id1" targetResource="id2" relationType="precede">	V1	VS1
<TemporalRelation sourceResource="id1" relationType="precedeOf">	V1,2	VS2

Note that the period of the retrieved segments (VS1, VS2) is left to implementor.

The following example illustrates the use of the TemoralQuery type, in which all three attributes are specified.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QFDeclaration>
        <Resource resourceID="silenceID" xsi:type="DescriptionResourceType">
          <AnyDescription xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
            <mpeg7:Mpeg7>
              <mpeg7:DescriptionUnit xsi:type="mpeg7:AudioSegmentType">
                <mpeg7:AudioDescriptor xsi:type="mpeg7:SilenceType"/>
              </mpeg7:DescriptionUnit>
            </mpeg7:Mpeg7>
          </AnyDescription>
        </Resource>
        <Resource resourceID="melodyID" xsi:type="DescriptionResourceType">
          <AnyDescription xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004"
xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd">
            <mpeg7:Mpeg7>
              <mpeg7:DescriptionUnit xsi:type="mpeg7:AudioSegmentType">
                <mpeg7:AudioDescriptionScheme xsi:type="mpeg7:MelodyType">
                  <mpeg7:Meter>
                    <mpeg7:Numerator>3</mpeg7:Numerator>
                    <mpeg7:Denominator>4</mpeg7:Denominator>
                  </mpeg7:Meter>
                  <mpeg7:MelodyContour>
                    <mpeg7:Contour>-1</mpeg7:Contour>
                    <mpeg7:Beat>80</mpeg7:Beat>
                  </mpeg7:MelodyContour>
                </mpeg7:AudioDescriptionScheme>
              </mpeg7:DescriptionUnit>
            </mpeg7:Mpeg7>
          </AnyDescription>
        </Resource>
      </QFDeclaration>
      <QueryCondition>
        <Condition xsi:type="TemporalQuery">
          <TemporalRelation sourceResource="silenceID" targetResource="melodyID"
relationType="urn:mpeg:mpqf:cs:TemporalRelationCS:2008:follows"/>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

12.10 Query By XQuery

12.10.1 Introduction

The QueryByXQuery type extends the abstract QueryType type and denotes a query operation for the use of XQuery expressions. The type is structured as a sequence of strings representing XQuery expressions. The XQuery expression is restricted to the use of constructs that produce a Boolean true or false decision on a single evaluation item at the target database. So, the scope of the XQuery expression is the root element of one evaluation item's metadata XML tree. Furthermore, within the XQuery expression included in the QueryByXQuery type, no output description is allowed.

12.10.2 Syntax

```
<complexType name="QueryByXQuery">
  <complexContent>
    <extension base="mpqf:QueryType">
      <sequence>
        <element name="XQuery" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

12.10.3 Semantics

Semantics of the QueryByXQuery type:

Name	*Definition*
QueryByXQuery	Extends the abstract QueryType type and denotes a query operation for the use of XQuery expressions.
XQuery	Specifies the XQuery expression that is used to filter information. Only XQuery expressions that operate on a single evaluation item and do respond either true or false based on the provided condition are allowed. No output description is allowed in the XQuery expression. The return of a Boolean value may be guaranteed by the use of the fn:boolean function provided by XQuery.

12.10.4 Examples

The following example illustrates the use of the `QueryByXQuery` type. In this example, the used XQuery expression filters all evaluation-item, i.e. in this case it is an XPath-item related to the MC's metadata XML tree containing the term *Video* in the `Name` element of the `MediaFormat` element. The expressions can become any complexity as long as they result in a Boolean decision on a true/false basis.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <Condition xsi:type="QueryByXQuery">
          <XQuery>
                  <![CDATA[
                      let $a := node()//MediaFormat/Content/Name
                      return
                      $a/text()="Video"]]>
          </XQuery>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

12.11 Query By Relevance Feedback

12.11.1 Introduction

The `QueryByRelevanceFeedback` type extends the `QueryType` type and describes a query operation that takes the result of the previous retrieval into consideration. The query operation allows the requester to identify good and/or bad examples within a previous result set like e.g., "Give me more examples like this one". Bad examples can be described by using the Boolean `NOT` operator accordingly.

The type is structured as a sequence of `ResultItem` elements of type `positiveInteger`. The appearance of the element is bound from zero to unbounded. If no `ResultItem` element is provided, the whole previous result set is dealt as input for a relevance feedback query. The required attribute `answerID` is of type `anyURI` and refers to a previous result set (indicated through its query ID) received from the responder.

12.11.2 Syntax

```
<complexType name="QueryByRelevanceFeedback">
  <complexContent>
    <extension base="mpqf:QueryType">
      <sequence>
        <element    name="ResultItem"    type="positiveInteger"    minOccurs="0"
maxOccurs="unbounded"/>
      </sequence>
      <attribute name="answerID" type="anyURI" use="required"/>
    </extension>
  </complexContent>
</complexType>
```

12.11.3 Semantics

Semantics of the `QueryByRelevanceFeedback` type:

Name	*Definition*
`QueryByRelevanceFeedback`	Extends the `QueryType` type and describes a query operation that takes the result of the previous retrieval into consideration. The query operation allows the requester to identify good examples within a previous result set like e.g., “Give me more examples like this one”.
`ResultItem`	Specifies the good examples from the `ResultItem` that refers to a record which is identified by its `recordNumber` (see `ResultItemType`) of a previous result. This record serves as input for the new query.
`answerID`	Specifies the identifier of the result set where the relevance feedback operation shall be performed. The previous result set delivered to the requester is identified by the `mpqfID` of the `MpegQueryType` type.

12.11.4 Example

The following example illustrates the use of the `QueryByRelevanceFeedback` operation for describing a relevance feedback request based on a previous result set. In this example, the requester has chosen the original result set items 4, 8 and 10 of the previously received query result as input for a new query. By using this query, a requester indicates to the retrieval system the wish to retrieve more results as the provided ones. In order to identify bad examples, one needs to use the Boolean `NOT` operator in combination with the `QueryByRelevanceFeedback` operation.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input>
      <QueryCondition>
        <Condition xsi:type="QueryByRelevanceFeedback"
answerID="IDofPreviousQuery">
          <ResultItem>4</ResultItem>
          <ResultItem>8</ResultItem>
          <ResultItem>10</ResultItem>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

12.12 Query By ROI

12.12.1 Introduction

The `QueryByROI` type extends the `QueryByMedia` type and describes a query operation that takes a media resource as input and allows the specification of a region of interest. During the evaluation of this query type the region of interest is required to be considered for search (exact or similar). Depending on the used domain (spatial, temporal), the query type provides two different elements, namely `TemporalRegionOfInterest` and `SpatialRegionOfInterest`. It is required that for the spatial domain (images) the `SpatialRegionOfInterest` element is to be used. Furthermore, it is required that for the temporal domain (audio, video) the `TemporalRegionOfInterest` element is to be used.

For the `SpatialRegionOfInterest` element, a region is defined by the `IntegerMatrixType` type which allows the specification of a list of positive integer values describing individual points. The amount of necessary integer values per point is defined by the `dim` (dimension) attribute of the `IntegerMatrixType` type. If the `dim` attribute is set to two then two successive integer values specify one point in 2 D space. If the dim attribute is set to three then three successive integer values specify one point in 3 D space. The individual points define the region where for instance for 2 D, three points identify a triangle, four points a rectangular, and so on. The order of the individual points is demonstrated by the examples below (see Figure 4) and goes contraclockwise.

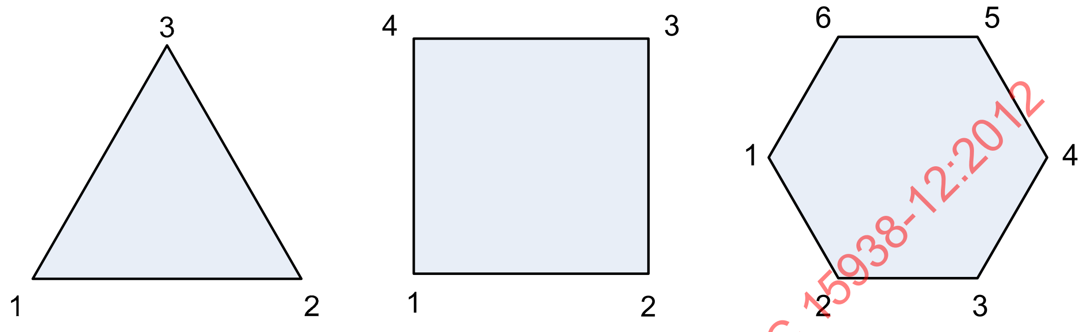


Figure 4 — Example point ordering for 2D objects

For 3D objects the individual 2D shapes need to be specified. The order is demonstrated by the example below (see Figure 5) and goes again contraclockwise. The letters mark the individual planes which need to be specified as 2D objects like above.

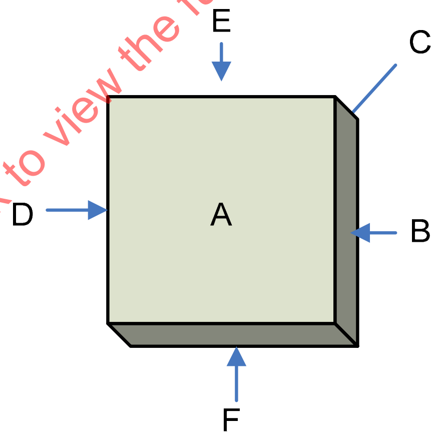


Figure 5 — Example plane ordering for 3D objects

For the `TemporalRegionOfInterest` element, a region is defined by the `TemporalRegionType` type. This type provides two elements, namely `StartTime` and `Duration`. The `StartTime` element specifies the beginning point in time of the temporal region. The value of the `Duration` element added to the value of the `StartTime` element specifies the end point in time of the temporal region.

12.12.2 Syntax

```
<complexType name="QueryByROI">
  <complexContent>
    <extension base="mpqf:QueryByMedia">
      <sequence>
        <choice>
          <element name="TemporalRegionOfInterest" type="mpqf:TemporalRegionType"
minOccurs="1" maxOccurs="unbounded"/>
          <element  name="SpatialRegionOfInterest"  type="mpqf:IntegerMatrixType"
minOccurs="1" maxOccurs="unbounded"/>
          </choice>
          </sequence>
      </extension>
  </complexContent>
</complexType>

<complexType name="IntegerMatrixType">
  <simpleContent>
    <extension base="mpqf:listOfPositiveInteger">
       <attribute name="dim" type="positiveInteger" use="required"/>
    </extension>
  </simpleContent>
</complexType>

<simpleType name="listOfPositiveInteger">
  <list itemType="nonNegativeInteger"/>
</simpleType>

<complexType name="TemporalRegionType">
  <sequence>
    <element    name="StartTime"    type="mpqf:startTimePointType"    minOccurs="1"
maxOccurs="1"/>
    <element     name="Duration"     type="mpqf:mediaDurationType"     minOccurs="1"
maxOccurs="1"/>
  </sequence>
</complexType>

<simpleType name="startTimePointType">
  <restriction base="string">
    <pattern                                    value="(\-?\d+(\-\d{2}(\-
\d{2})?)?)?(T\d{2}(:\d{2}(:\d{2}(:\d+)?)?)?)?(F\d+)?"/>
  </restriction>
</simpleType>

<simpleType name="mediaDurationType">
  <restriction base="string">
    <pattern value="\-?P(\d+D)?(T(\d+H)?(\d+M)?(\d+S)?(\d+N)?)?(\d+F)?"/>
  </restriction>
</simpleType>
```

12.12.3 Semantics

Semantics of the `QueryByROI` type:

Name	*Definition*
`QueryByROI`	The `QueryByROI` type extends the `QueryByMedia` type and describes a query operation that takes a media resource as input and allows the specification of a region of interest. During the evaluation of this query type the region of interest is required to be considered for search (exact or similar).
`TemporalRegionOfInterest`	Specifies a temporal region within a media resource where it is required that the media resource belongs to the temporal domain (audio, video). The temporal region is described by the `TemporalRegionType` type.
`SpatialRegionOfInterest`	Specifies a spatial region within a media resource where it is required that the media resource belongs to the spatial domain (image). The spatial region is described by the `IntegerMatrixType` type.
`IntegerMatrixType`	Specifies a spatial region. A spatial region is defined by the `IntegerMatrixType` type which allows the specification of a list of positive integer values describing individual points. The amount of necessary integer values per point is defined by the `dim` attribute of `IntegerMatrixType` type. The individual points define the region where two points mean a rectangular, three points a triangle and so on.
`Dim`	Attribute that defines the dimension of the individual points (e.g., 2 means that 2 integer values are required to define one point).
`listOfPositiveInteger`	Simple type that specifies a list of positive integer values which describes individual points. The amount of necessary integer values per point is defined by the `dim` attribute of the `IntegerMatrixType` type.
`TemporalRegionType`	Specifies a temporal region within an audio-visual resource.
`StartTime`	Specifies the beginning point in time of the temporal region. The beginning point in time is described in relation to the beginning of the given media resource whose time point is required to be T00:00:00.
`Duration`	Specifies the duration of the temporal region. The end point in time of the temporal region may be calculated by adding the value of the `Duration` element to the value of the `StartTime` element.
startTimePointType	Specifies the pattern for describing the beginning point in time of the temporal region.
mediaDurationType	Specifies the pattern for describing the duration of the temporal region.

12.12.4 Example

The following example illustrates the use of the `QueryByROI` operation. The query request declares two media resources, one image (`resourceID` is image1) and one video (`resourceID` is video1). The query condition presents both scenarios for defining regions based on the given media resource. The first `QueryByROI` condition operates on the spatial domain and points to the image media resource. The region of interest specifies a rectangle based on the four points (20, 20), (50, 20), (50, 50) and (20, 50) which has to be considered for search. The second `QueryByROI` condition operates on the temporal domain and points to the video media resource. The region of interest specifies a beginning point in time of T00:00:00:0F30000 and a duration of PT4M.

```
 <MpegQuery mpqfID="exampleROI">
  <Query>
    <Input>
      <QFDeclaration>
        <Resource xsi:type="MediaResourceType" resourceID="image1">
          <MediaResource>
            <MediaUri>http://testimage</MediaUri>
          </MediaResource>
        </Resource>
        <Resource xsi:type="MediaResourceType" resourceID="video1">
          <MediaResource>
            <MediaUri>http://testvideo</MediaUri>
          </MediaResource>
        </Resource>
      </QFDeclaration>
      <QueryCondition>
        <Condition xsi:type="AND">
          <Condition xsi:type="QueryByROI">
            <MediaResourceREF>image1</MediaResourceREF>
            <SpatialRegionOfInterest dim="2" >20 20 50 20 50 50 20
50</SpatialRegionOfInterest>
          </Condition>
          <Condition xsi:type="QueryByROI">
            <MediaResourceREF>video1</MediaResourceREF>
            <TemporalRegionOfInterest>
              <StartTime>T00:00:00:0F30000</StartTime>
              <Duration>PT4M</Duration>
            </TemporalRegionOfInterest>
          </Condition>
        </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

12.13 Join Query

12.13.1 Introduction

The Join operation allows the definition of filtering conditions which act over multiple sets of multimedia objects. In the Join operation, it is possible to specify conditions which define separate filtered sets and combine them.

The `JoinType` type defines separate filtering flows by using separate `Condition` elements along with an identifier. It defines a combined `JoinCondition` which will act over the pairs of objects resulting from the cross product of the different filtering sets.

12.13.2 Syntax

```
<complexType name="JoinType">
  <sequence>
    <element name="From" maxOccurs="2">
      <complexType>
        <sequence>
          <element name="EvaluationPath" minOccurs="0" type="mpqf:xPathType"/>
          <element name="TargetMediaType" type="mpqf:mimeType" minOccurs="0"
maxOccurs="unbounded"/>
          <element name="Condition" type="mpqf:BooleanExpressionType"/>
        </sequence>
        <attribute name="id" type="ID" use="required"/>
      </complexType>
    </element>
    <element name="JoinCondition" type="mpqf:BooleanExpressionType"/>
  </sequence>
</complexType>
```

12.13.3 Semantics

Semantics of the `JoinType` type:

Name	*Definition*
`JoinType`	Describes Join operation. The number of set to which the Join operation applies is fixed to two at a time.
`From`	Describes an operand of Join operation, i.e. a set of multimedia objects.
`EvaluationPath`	Specifies an element or an attribute in XML documents using an XPath expression. This indicates the granularity of multimedia objects.
`TargetMediaType`	The sequence of the target media types declares the desired MIME types which the user expects as a result.
`Condition`	Specifies the filtering condition that makes a set of multimedia objects.
`Id`	Indicates the identifier of the `From` element in order to be referred by other elements.
`JoinCondition`	Specifies the Join condition of the two sets of multimedia objects defined by the `From` elements.

12.13.4 Examples

The following example shows a query asking for pairs of audio and video objects which share the same creation title. It specifies that the media URI of each pair of selected objects is be returned.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Input previousAnswerID="http://www.altova.com">
      <OutputDescription outputNameSpace="urn:mpeg:mpeg7:schema:2004">
        <ReqField fromREF="id1">/MediaUri</ReqField>
        <ReqField fromREF="id2">/MediaUri</ReqField>
      </OutputDescription>
      <QueryCondition>
        <Join>
          <From id="id1">
            <EvaluationPath>/Mpeg7</EvaluationPath>
            <Condition xsi:type="QueryByXQuery">
              <XQuery><![CDATA[
                       let $a := node()//MediaFormat/Content/Name
                       return
                       $a/text()="Video"]]></XQuery>
            </Condition>
          </From>
          <From id="id2">
            <EvaluationPath>/Mpeg7</EvaluationPath>
            <Condition xsi:type="QueryByXQuery">
              <XQuery><![CDATA[
                           let $a := node()//MediaFormat/Content/Name
                           return
                           $a/text()="Audio"]]></XQuery>
            </Condition>
          </From>
          <JoinCondition xsi:type="QueryByXQuery">
            <XQuery><![CDATA[
                               node(id1)//Creation/Title =
                               node(id2)//Creation/Title ]]></XQuery>
          </JoinCondition>
        </Join>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

12.14 QueryBySPARQL

12.14.1 Introduction

The QueryBySPARQL type extends the abstract QueryType type and denotes a query operation for the use of SPARQL expressions. The type is structured as a sequence of strings representing SPARQL expressions but is restricted to the use of constructs that produce a Boolean true or false decision on a single evaluation item at the target database. Furthermore, within the SPARQL expression included in the QueryBySPARQL type, no output description is allowed.

12.14.2 Syntax

```
<complexType name="QueryBySPARQL">
  <complexContent>
   <extension base="mpqf:QueryType">
     <sequence>
       <element name="SPARQL" type="string"/>
     </sequence>
   </extension>
  </complexContent>
</complexType>
```

12.14.3 Semantics

Semantics of the `QueryBySPARQL` query type:

Name	*Definition*
`QueryBySPARQL`	Extends the abstract QueryType type and denotes a query operation for the use of SPARQL expressions.
`SPARQL`	Specifies the SPARQL expression that is used to filter information. Only SPARQL expression that operate on a single triple and do respond either true or false based on the provided condition are allowed. No output description is allowed in the SPARQL expression. The return of a Boolean value may be guaranteed by the use of the ASK function provided by SPARQL.

12.14.4 Example

The following example illustrates the use of the SPARQL query type on an imaginary semantic annotation in a surveillance system scenario. The example searches for persons and its location and time information (see OutputDescription) that matches the drawn RDF subgraph (see ASK part within the query part). The RDF subgraph assumes as data model the annotation of a closed (all persons are known and can be tracked by specific sensors) video surveillance scenario containing locations (region of interests (ROI) or rooms) which are monitored by cameras. At runtime of the system, automatic events are triggered when persons are entering rooms or specific areas (ROI, e.g. a coffee area). In this context, the RDF subgraph and therefore the query searches for all persons that have been monitored entering a specific ROI which is controlled by a camera located in *Room240.*

```
<MpegQuery mpqfID="">
  <Query>
    <Input>
      <OutputDescription>
        <ReqSemanticField>?person</ReqSemanticField>
        <ReqSemanticField>?location</ReqSemanticField>
        <ReqSemanticField>?temp</ReqSemanticField>
      </OutputDescription>
      <QueryCondition>
        <Condition xsi:type="QueryBySPARQL">
          <SPARQL>
            <![CDATA[
        PREFIX xsd:      <http://www.w3.org/2001/XMLSchema#>
        PREFIX rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        PREFIX ontology: <http://www.example.de/ontology#>
        PREFIX config:   <http://www.example.de/configuration#>
        ASK
        { ?person   ontology:hasName      ?name ;
                    ontology:hasPosition  ?node .
          ?node     ontology:hasPosition_Location  ?roi ;
                    ontology:hasPosition_Duration  ?temp .
          ?roi      ontology:hasCamera    ?cam .
          ?cam      ontology:isLocated    ?location .
          ?location ontology:hasName     "Room240"^^xsd:string .
        } ]]>
          </SPARQL>
        </Condition>
      </QueryCondition>
    </Input>
  </Query></MpegQuery>
```

13 Output Query Format

13.1 Output

13.1.1 Introduction

The Output Query Format is defined by the `OutputQueryType` type. It provides a container for all the results from a responder to a requester. It may contain not only query results but also any messages such as error and exception.

13.1.2 Syntax

```
<complexType name="OutputQueryType">
   <sequence>
      <element name="GlobalComment" type="string" minOccurs="0"/>
      <element    name="ResultItem"    type="mpqf:ResultItemType"    minOccurs="0"
maxOccurs="unbounded"/>
      <element name="SystemMessage" type="mpqf:SystemMessageType" minOccurs="0"/>
   </sequence>
   <attribute name="currPage" type="positiveInteger" default="1"/>
   <attribute name="totalPages" type="positiveInteger" default="1"/>
   <attribute name="expirationDate" type="dateTime" use="optional"/>
  </complexType>
```

13.1.3 Semantics

Semantics of the `OutputQueryType` type:

Name	*Definition*
`OutputQueryType`	Describes all the responses from a responder in respect to an input query request. It may contain results of queries and/or message from the responder. It may also include the page information and expiration date of the results.

Name	*Definition*
`GlobalComment`	Describes information about the query results (optional). It may also be a message from the responder to the requester.
`ResultItem`	Describes a single result returned from a responder (optional).
`SystemMessage`	Describes the message related to the responder system (optional).
`currPage`	Indicates the current page of the results. If no value is specified, the `currPage` attribute is assumed to be 1.
`totalPages`	Indicates the total page of the results. If no value is specified, the `totalPages` attribute is assumed to be 1.
`expirationDate`	Indicates the time point when the responder expires the results (optional).

13.1.4 Example

The following example illustrates the use of the `OutputQueryType` type. It conveys both results and a system message related to the responder at the same time.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Output>
      <GlobalComment>This is the message from the responder</GlobalComment>
      <ResultItem recordNumber="001" rank="1" confidence="1.0">
        <MediaResource>http://www.mpeg7qf/db/video/19701221.mpg</MediaResource>
      </ResultItem>
      <ResultItem recordNumber="002" rank="2" confidence="0.99">
        <MediaResource>http://www.mpeg7qf/db/video/19690117.mpg</MediaResource>
      </ResultItem>
      <ResultItem recordNumber="003" rank="2" confidence="0.98">
        <MediaResource>http://www.mpeg7qf/db/video/19980212.mpg</MediaResource>
      </ResultItem>
      <ResultItem recordNumber="004" rank="3" confidence="0.87">
        <MediaResource>http://www.mpeg7qf/db/video/19990414.mpg</MediaResource>
      </ResultItem>
      <ResultItem recordNumber="005" rank="3" confidence="0.85">
        <MediaResource>http://www.mpeg7qf/db/video/20071119.mpg</MediaResource>
      </ResultItem>
      <SystemMessage>
        <Status>
          <Code>001</Code>
          <Description>Query was successful</Description>
        </Status>
      </SystemMessage>
    </Output>
  </Query>
</MpegQuery>
```

13.2 ResultItem

13.2.1 Introduction

The `ResultItem` element describes a single result item within the output query format and is specified by the `ResultItemType` type. The `ResultItem` element is described by a media resource represented by `anyURI` type, MPEG-7 or other metadata descriptions, or free text. The `ResultItem` element also provides `identifier`, `rank` and `confidence` attributes for each result.

13.2.2 Syntax

```
<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>
```

13.2.3 Semantics

Semantics of the `ResultItemType` type:

Name	*Definition*
`ResultItemBaseType`	Specifies a base class for the `ResultItemType` type.
`ResultItemType`	Specifies the `ResultItem` element. Describes a single result item of the query.
`Comment`	Describes a message related to a result item from responders to a requester.

Name	*Definition*
`fromREF`	Indicates a condition from which the result comes. This intends to be used in the case of Join operation.
`TextResult`	Describes a result in string form (optional). It may be just a number like “65”, or XML data which is packed as CDATA.
`Thumbnail`	Describes a link to a thumbnail of the result item (optional).
`MediaResource`	Describes a link to the result media resource by anyURI type (optional).
`Description`	Describes a result in XML with specifying its namespace (optional). The XML data may be MPEG-7 description.
`AggregationResult`	Describes the result of an aggregation expression.
`aggregateID`	Specifies a unique id of the aggregation. This `aggregateID` is given in the `Aggregate` element of the Input Query.
`recordNumber`	Indicates the number assigned to the result item. The number may also be used to point to the result when relevance feedback retrieval is activated.
`Rank`	Indicates the rank of the item (optional). More than one item may have the same rank.
`Confidence`	Indicates the confidence of the item in the correctness of the result (optional). The value zero represents no confidence and the value 1.0 perfect confidence. All other values between zero and one represent intermediate levels of confidence.
`originID`	Specifies a point from where the result item comes. It may be the URL indicating the service ID of the responder replying to the query request, when there are multiple services responding to the single request.
`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.

13.2.4 Examples

The following example illustrates the use of the `ResultItemType` type. The output conveys four `ResultItem` elements which includes all three kinds of expression, such as by URL, free text, XML data. The examples of XML data are MPEG-7 complete descriptions and description units. Note, that this example does not simulate a real world scenario and should only highlight all different possibilities. In general, every `ResultItem` element should contain the same structure and same amount of elements.

```
<MpegQuery mpqfID="someID">
  <Query>
   <Output>
      <GlobalComment>This is the message from the responder</GlobalComment>
      <ResultItem recordNumber="001" rank="1" confidence="1.0">
         <MediaResource>http://www.mpeg7qf/db/video/19701221.mpg</MediaResource>
      </ResultItem>
      <ResultItem recordNumber="002">
         <TextResult>WorldCup 2002</TextResult>
      </ResultItem>
      <ResultItem recordNumber="003" confidence="0.98">
```

```
          <Description xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004">
             <mpeg7:Mpeg7>
                <mpeg7:Description xsi:type="mpeg7:ContentEntityType">
                   <mpeg7:MultimediaContent xsi:type="mpeg7:ImageType">
                      <mpeg7:Image/>
                   </mpeg7:MultimediaContent>
                </mpeg7:Description>
             </mpeg7:Mpeg7>
          </Description>
       </ResultItem>
       <ResultItem recordNumber="004" confidence="0.78">
          <Description xmlns:mpeg7="urn:mpeg:mpeg7:schema:2004">
             <mpeg7:Mpeg7>
                <mpeg7:DescriptionUnit xsi:type="mpeg7:PersonType">
                   <mpeg7:Name>
                      <mpeg7:GivenName>Mari</mpeg7:GivenName>
                   </mpeg7:Name>
                </mpeg7:DescriptionUnit>
             </mpeg7:Mpeg7>
          </Description>
       </ResultItem>
    </Output>
  </Query>
</MpegQuery>
```

13.3 SystemMessageType

13.3.1 Introduction

The `SystemMessageType` type describes a message related to the responder system. A system message may contain the following information: `status`, `warning`, `exception`.

13.3.2 Syntax

```
<complexType name="SystemMessageType">
  <choice>
    <element name="Status" type="mpqf:InformationType" maxOccurs="unbounded"/>
    <element name="Warning" type="mpqf:InformationType" maxOccurs="unbounded"/>
    <element name="Exception" type="mpqf:InformationType" maxOccurs="unbounded"/>
  </choice>
</complexType>

<complexType name="InformationType">
  <sequence>
    <element name="Code" type="positiveInteger"/>
    <element name="Description" type="string"/>
  </sequence>
</complexType>
```

13.3.3 Semantics

Semantics of the `SystemMessageType` type:

Name	*Definition*
`SystemMessageType`	Specifies the `SystemMessageType` type. Describes a set of messages related to the responder.

Name	*Definition*
`Status`	Describes the status of the responder.
`Warning`	Describes the warning from the responder.
`Exception`	Describes the exception the responder encountered during the process.

Semantics of the `InformationType`:

Name	*Definition*
`Code`	Describes the number assigned for the information.
`Description`	Describes the details of the information.

The code number and its description are defined in Annex A.

13.3.4 Examples

The following example illustrates the use of the `SystemMessageType` type. It tells the requester that the responder is busy.

```
<MpegQuery mpqfID="someID">
  <Query>
    <Output>
      <SystemMessage>
        <Warning>
          <Code>101</Code>
          <Description>Server resource busy </Description>
        </Warning>
      </SystemMessage>
    </Output>
  </Query>
</MpegQuery>
```

14 Query Management Tools

14.1 Introduction

This subclause defines the types used within query management message instances. The message exchange is based on a request-response style of communication between peers and supports the processes of service discovery and service selection based on service capability descriptions and service identification.

14.2 InputManagementType

14.2.1 Introduction

The `InputManagementType` type is used for service discovery. A service denotes a single multimedia repository or an aggregated service providing access to a set of multimedia repositories. The capability of every service is described by its `CapabilityType` type. Each service capability description determines the supported query format, the supported metadata, supported media formats (in examples and resultset), query types (e.g., `QueryByMedia`), supported expressions (e.g. `AND`) and usage conditions (e.g., payment required, etc.).

14.2.2 Syntax

```
<complexType name="InputManagementType">
    <sequence>
      <element name="DesiredCapability" type="mpqf:CapabilityType"
minOccurs="0"/>
      <element name="ServiceID" type="anyURI" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
</complexType>

<complexType name="CapabilityType">
  <sequence>
    <element name="SupportedQFProfile" type="mpqf:CapabilityTermType"
                 minOccurs="0"/>
    <element name="SupportedMetadata" type="anyURI" minOccurs="0"
                 maxOccurs="unbounded"/>
    <element name="SupportedExampleMediaTypes" minOccurs="0">
      <simpleType>
        <list itemType="mpqf:mimeType"/>
      </simpleType>
    </element>
    <element name="SupportedResultMediaTypes" minOccurs="0">
      <simpleType>
        <list itemType="mpqf:mimeType"/>
      </simpleType>
    </element>
    <element name="SupportedQueryTypes" type="mpqf:CapabilityTermType"
                 minOccurs="0" maxOccurs="unbounded"/>
    <element name="SupportedExpressions" type="mpqf:CapabilityTermType"
                 minOccurs="0" maxOccurs="unbounded"/>
    <element name="UsageConditions" minOccurs="0" maxOccurs="unbounded">
      <complexType>
        <complexContent>
          <extension base="mpqf:TermType">
            <attribute name="usageID" type="ID" use="optional"/>
          </extension>
        </complexContent>
      </complexType>
    </element>
  </sequence>
</complexType>

<complexType name="CapabilityTermType">
  <complexContent>
    <extension base="mpqf:TermType">
      <attribute name="usageRefList" type="IDREFS" use="optional"/>
    </extension>
  </complexContent>
</complexType>
```

14.2.3 Semantics

Semantics of the `InputManagementType` type:

Name	*Definition*
`InputManagementType`	Defines the syntax of the `InputManagementType` type, which allows message instances be used for service discovery, service selection and service capability description.
`DesiredCapability`	The content of this element is an instance of a service capability description specifying the users desired service capabilities using the `CapabilityType` type.
`ServiceID`	Indicates the `identifier` (by an URI) of a service. This can either be a URN or a URL.

Service discovery is supported by message instances of type `InputManagementType` that a requesting peer sends to a service, where the following additional semantic rules apply to all instances:

- When this element is empty, a requester is requesting all available service capabilities.
  - As result a requester receives all available services with their capability as return with `ServiceID`.

- When there is only the `DesiredCapability` element presented, one is requesting the `serviceID`'s who support this capability.
  - As result a requester receives a description of the available capability in return with the `ServiceID` of those services who support the desired capability.

- When there are only `ServiceID` elements in the `Input` element, a requester is requesting the capability of the specific services.
  - As result a requester receives the available capability as return with `serviceID`.

- When both `DesiredCapability` element and the `ServiceID` element are present, a requester is asking whether the specified service supports the desired capability.
  - As result the requester receives available capability in return with the `ServiceID` element of those services that support the desired capability among the given `serviceID`'s specified in the input.

Semantics of the `CapabilityType` type:

Name	*Definition*
`CapabilityType`	Defines the syntax of the `CapabilityType` type, which allows the description of service capabilities
`SupportedQFProfile`	Describes the supported query format profile of the specific service by a URN. A `QFProfile` is a defined set of supported metadata, query types and expressions that a service is capable to process. The profile of this part of ISO/IEC 15938 is defined in a classification scheme (see Annex B) and can be identified by its URN.

Name	*Definition*
`SupportedMetadata`	Describes the metadata that can be processed by a certain service using a list of URIs. The set of metadata is identified either by their namespace URN or by the URL of it's definition (DTD/Schema location).
`SupportedExampleMediaTypes`	Describes the supported MIME media types of media resources a certain service can process as input.
`SupportedResultMediaTypes`	Describes the supported MIME media types of media resources a certain service can deliver in the output (result set).
`SupportedQueryTypes`	Describes the supported query types of a certain service. Query types are listed in a classification scheme (see Annex B) and can be identified by their URN.
`SupportedExpressions`	Describes the supported expressions of a certain service. Expressions are listed in a classification scheme (see Annex B) and can be identified by their URN.
`UsageConditions`	Describes the usage conditions of a certain service. Usage conditions are listed in a classification scheme (see Annex B) and can be identified by their URN. Several different usage conditions can be supported by a service. They can be one by one related to query types, expressions or a QueryFormat Profile being referenced respectively.

Semantics of the `CapabilityTermType` type:

Name	*Definition*
`CapabilityTermType`	Defines the syntax of the `CapabilityTermType` type, which represents a specific term denoting a certain e.g., query type. The respective terms shall be defined in the corresponding classification scheme.
`usageRefList`	Describes the reference list which points to a set of specified usage conditions where the capability belongs to. They are identifiably listed in the usageConditions element. For instance, if a `QueryByMedia` query requires authorization and payment, then both respective IDs need to be enlisted in this attribute at the respective supported query type (`QueryByMediaType`) capability description.

14.2.4 Example

In the following, examples for all four possible service discovery scenarios, provided by the `InputManagement` element, are presented. Examples of possible responses, namely the list of matching service capability descriptions, are given in subclause 14.3.4.

Scenario 1: Give me whatever you have!

In this scenario, a requester is interested in a list of available services the aggregation service is able to access. In this case an empty `Input` element needs to be sent by the requester to the responder.

```
<MpegQuery mpqfID="someID">
  <Management>
    <Input/>
  </Management>
</MpegQuery>
```

Scenario 2: Give me everything which matches at least my desired capabilities.

In this scenario, the requester needs to provide a minimum service capability description which expresses its desired retrieval functionality. For instance, the example below requests for all services that support the MPQF version 2008 and the MPEG-7 standard for describing the multimedia data. In addition, the services shall support Boolean expressions (100.3.1) and the two query types, `QueryByMedia` (100.3.6.1) and `QueryByXQuery` (100.3.6.4), respectively. Furthermore, the query types shall satisfy the corresponding usage conditions. In case of the `QueryByMedia` type, Authentication (200.1) and Payment (200.3) should be required. The `QueryByXQuery` type should be of free use (200.4).

```
<MpegQuery mpqfID="someID">
  <Management>
    <Input>
      <DesiredCapability>
        <SupportedQFProfile href="urn:mpeg:mpqf:2008:CS:Full"/>
        <SupportedMetadata>urn:mpeg:mpeg7:2004</SupportedMetadata>
        <SupportedExampleMediaTypes>audio/mp3
                                    video/mpg
        </SupportedExampleMediaTypes>
        <SupportedResultMediaTypes>audio/aac
video/mpg</SupportedResultMediaTypes>
        <SupportedQueryTypes href="urn:mpeg:mpqf:2008:CS:full:100.3.6.1"
                               usageRefList="id1 id3"/>
        <SupportedQueryTypes href="urn:mpeg:mpqf:2008:CS:full:100.3.6.4"
                               usageRefList="id2"/>
        <SupportedExpressions href="urn:mpeg:mpqf:2008:CS:full:100.3.1"/>
        <UsageConditions href="urn::mpeg:mpqf:2008:CS:full:200.1" usageID="id1"/>
        <UsageConditions href="urn:mpeg:mpqf:2008:CS:full:200.4" usageID="id2"/>
        <UsageConditions href="urn:mpeg:mpqf:2008:CS:full:200.3" usageID="id3"/>
      </DesiredCapability>
    </Input>
  </Management>
</MpegQuery>
```

Scenario 3: Give me the service description of the following services.

In this scenario, the requester already knows the location of some services but has no idea about their capabilities. In this case, one has to send an input management request, containing all known `ServiceID` elements, to the aggregation service. The aggregation service then returns their capabilities.

```
<MpegQuery mpqfID="someID">
  <Management>
    <Input>
      <ServiceID>http://exampleservice-1.com</ServiceID>
      <ServiceID>http://exampleservice-2.com</ServiceID>
    </Input>
  </Management>
</MpegQuery>
```

Scenario 4: Give me all services out of the given list of services that matches the desired capabilities.

In this scenario, the requester already knows the location of some services but wants to receive a filtered list according to the desired capabilities. In order to avoid requesting all capabilities of all services and performing the filtering at the requesters side (maybe due limited resource capabilities), the aggregation service takes care of this tasks.

```
<MpegQuery mpqfID="someID">
  <Management>
    <Input>
      <DesiredCapability>
        <SupportedQFProfile href="urn:mpeg:mpqf:2008:CS:Full"/>
        <SupportedMetadata>urn:mpeg:mpeg7:2004</SupportedMetadata>
        <SupportedExampleMediaTypes>audio/mp3
video/mpg</SupportedExampleMediaTypes>
        <SupportedResultMediaTypes>audio/aac
video/mpg</SupportedResultMediaTypes>
        <SupportedQueryTypes href="urn:mpeg:mpqf:2008:CS:full:100.3.6.1"
                             usageRefList="id1 id3"/>
        <SupportedQueryTypes href="urn:mpeg:mpqf:2008:CS:full:100.3.6.4"
                             usageRefList="id2"/>
        <SupportedExpressions href="urn:mpeg:mpqf:2008:CS:full:100.3.1"/>
        <UsageConditions                  href="urn::mpeg:mpqf:2008:CS:full:200.1"
usageID="id1"/>
        <UsageConditions href="urn:mpeg:mpqf:2008:CS:full:200.4" usageID="id2"/>
        <UsageConditions href="urn:mpeg:mpqf:2008:CS:full:200.3" usageID="id3"/>
      </DesiredCapability>
      <ServiceID>http://exampleservice-1.com</ServiceID>
      <ServiceID>http://exampleservice-2.com</ServiceID>
      <ServiceID>http://exampleservice-3.com</ServiceID>
      <ServiceID>http://exampleservice-4.com</ServiceID>
      <ServiceID>http://exampleservice-5.com</ServiceID>
      <ServiceID>http://exampleservice-6.com</ServiceID>
    </Input>
  </Management>
</MpegQuery>
```

14.3 OutputManagementType

14.3.1 Introduction

The `OutputManagementType` type provides means for replying to service discovery requests initiated by the requester. A service or aggregated service provider returns either a list of available service capability descriptions or a system message in case of an error. If no service is available or matches the given capabilities, then an empty `Output` element should be returned.

14.3.2 Syntax

```
<complexType name="OutputManagementType">
  <sequence>
    <choice>
      <element name="AvailableCapability" type="mpqf:AvailableCapabilityType"
               minOccurs="0" maxOccurs="unbounded"/>
      <element name="SystemMessage" type="mpqf:SystemMessageType"
               minOccurs="0"/>
    </choice>
  </sequence>
```

```
</complexType>

<complexType name="AvailableCapabilityType">
 <complexContent>
   <extension base="mpqf:CapabilityType" >
     <attribute name="serviceID" type="anyURI" use="required"/>
   </extension>
 </complexContent>
</complexType>
```

14.3.3 Semantics

Semantics of the `OutputManagementType` type:

Name	*Definition*
`OutputManagementType`	Specifies the syntax of the `OutputManagementType` type, which describes the result of a service discovery request.
`AvailableCapability`	Describes the capability of one available service (of `AvailableCapabilityType` type) fitting the request. The occurrence of an `AvailableCapability` element can be multiple in case of a response from an aggregation service.
`SystemMessage`	Describes a system message in case of an error. They shall be of `SystemMessageType` type.

Semantics of the `AvailableCapabilityType` type:

Name	*Definition*
`serviceID`	Indicates the `identifier` (by an URI) of a service. This can either be a URN or a URL.

14.3.4 Example

In the following examples, the `OutputManagement` datatype is used to describe some service capability descriptions which may result by service discovery requests. The first example shows a possible response for one of the given service discovery requests of the previous subclause. The response contains the capability description of two services matching the request (scenario 1 to 4).

```
<MpegQuery mpqfID="someID">
  <Management>
    <Output>
      <AvailableCapability serviceID="http://exampleservice-1.com">
          <SupportedQFProfile href="urn:mpeg:mpqf:2008:CS:Full"/>
          <SupportedMetadata>urn:mpeg:mpeg7:2004</SupportedMetadata>
          <SupportedExampleMediaTypes>audio/mp3 video/mpg
          </SupportedExampleMediaTypes>
          <SupportedResultMediaTypes>audio/aac
video/mpg</SupportedResultMediaTypes>
          <SupportedQueryTypes href="urn:mpeg:mpqf:2008:CS:full:100.3.6.1"
```

```
                                usageRefList="ex1id1 ex1id3"/>
          <SupportedQueryTypes href="urn:mpeg:mpqf:2008:CS:full:100.3.6.4"
                                  usageRefList="ex1id2"/>
          <SupportedExpressions href="urn:mpeg:mpqf:2008:CS:full:100.3.1"/>
          <UsageConditions href="urn::mpeg:mpqf:2008:CS:full:200.1"
                            usageID="ex1id1"/>
          <UsageConditions href="urn:mpeg:mpqf:2008:CS:full:200.4"
                            usageID="ex1id2"/>
          <UsageConditions href="urn:mpeg:mpqf:2008:CS:full:200.3"
                            usageID="ex1id3"/>
      </AvailableCapability>
      <AvailableCapability serviceID="http://exampleservice-2.com">
          <SupportedQFProfile href="urn:mpeg:mpqf:2008:CS:full"/>
          <SupportedMetadata>urn:mpeg:mpeg7:2004</SupportedMetadata>
          <SupportedExampleMediaTypes>video/mpg</SupportedExampleMediaTypes>
          <SupportedResultMediaTypes>image/*
video/mpg</SupportedResultMediaTypes>
          <SupportedQueryTypes href="urn:mpeg:mpqf:2008:CS:full:100.3.6.1"
                                usageRefList="ex2id1 ex2id3"/>
          <SupportedExpressions href="urn:mpeg:mpqf:2008:CS:full:100.3.1"/>
          <UsageConditions href="urn::mpeg:mpqf:2008:CS:full:200.1"
                            usageID="ex2id1"/>
          <UsageConditions href="urn:mpeg:mpqf:2008:CS:full:200.3"
                            usageID="ex2id3"/>
      </AvailableCapability>
    </Output>
  </Management>
</MpegQuery>
```

The following example demonstrates the result where no services are available or none of the existing services (registered at the aggregation service) matches the requester's requirements. In this case, an empty `Output` element is returned.

```
<MpegQuery mpqfID="someID">
  <Management>
    <Output>
    </Output>
  </Management>
</MpegQuery>
```

The last example presents a possible result in case of an error which occurred during the processing of the request. Depending on the type of error, the messages can vary. Please see Annex A for a list of defined error codes.

```
<MpegQuery mpqfID="someID">
  <Management>
    <Output>
      <SystemMessage>
        <Exception>
          <Code>202</Code>
          <Description>Error while processing the request</Description>
        </Exception>
      </SystemMessage>
    </Output>
  </Management>
</MpegQuery>
```

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

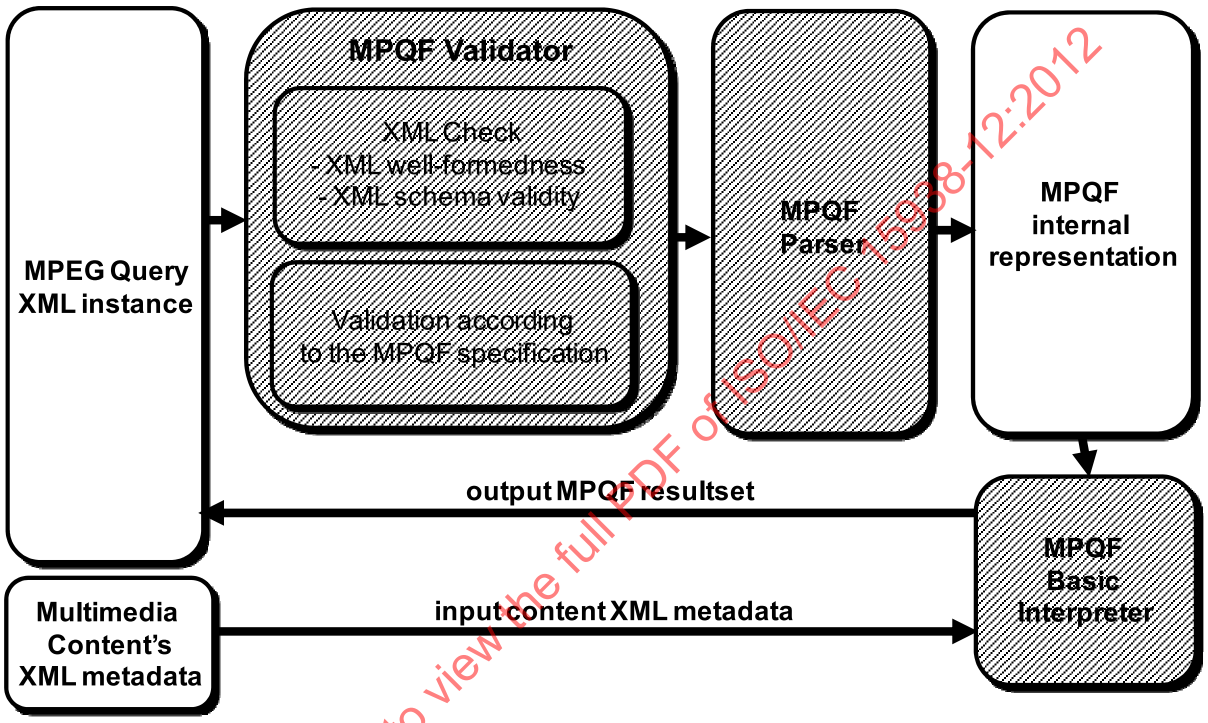


Figure 6 — Reference/utility software architecture

The architecture (see Figure 6) 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 7 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.

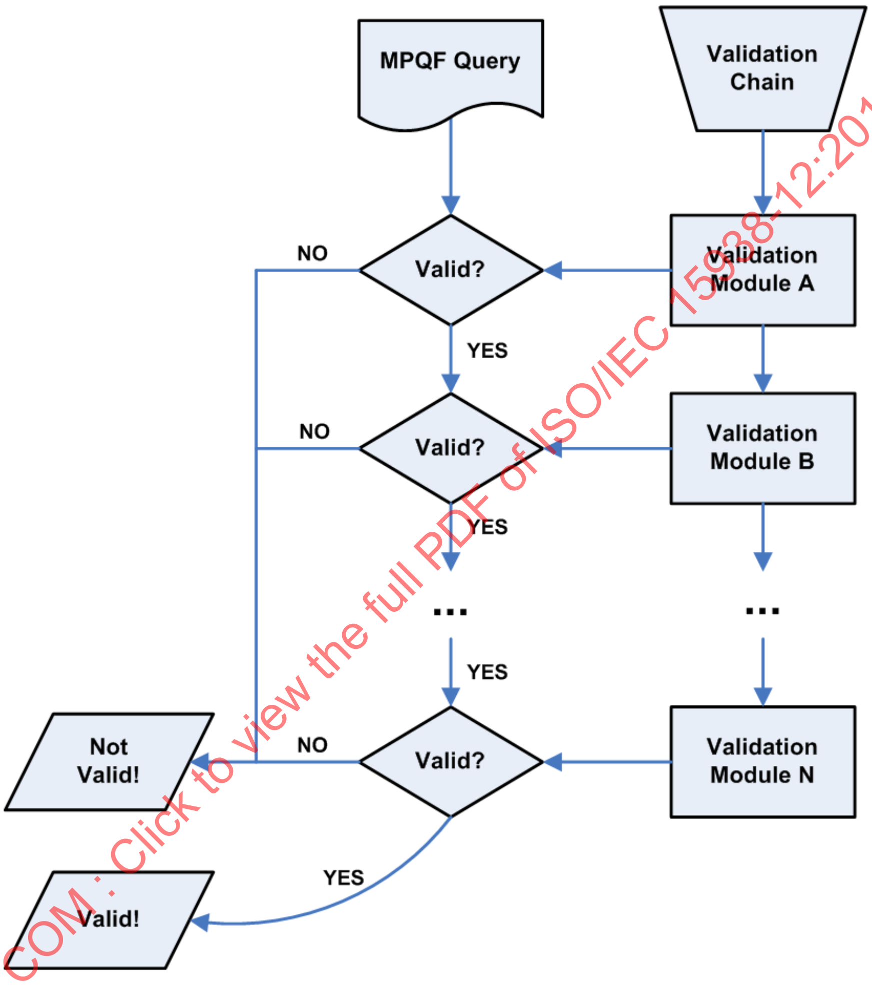
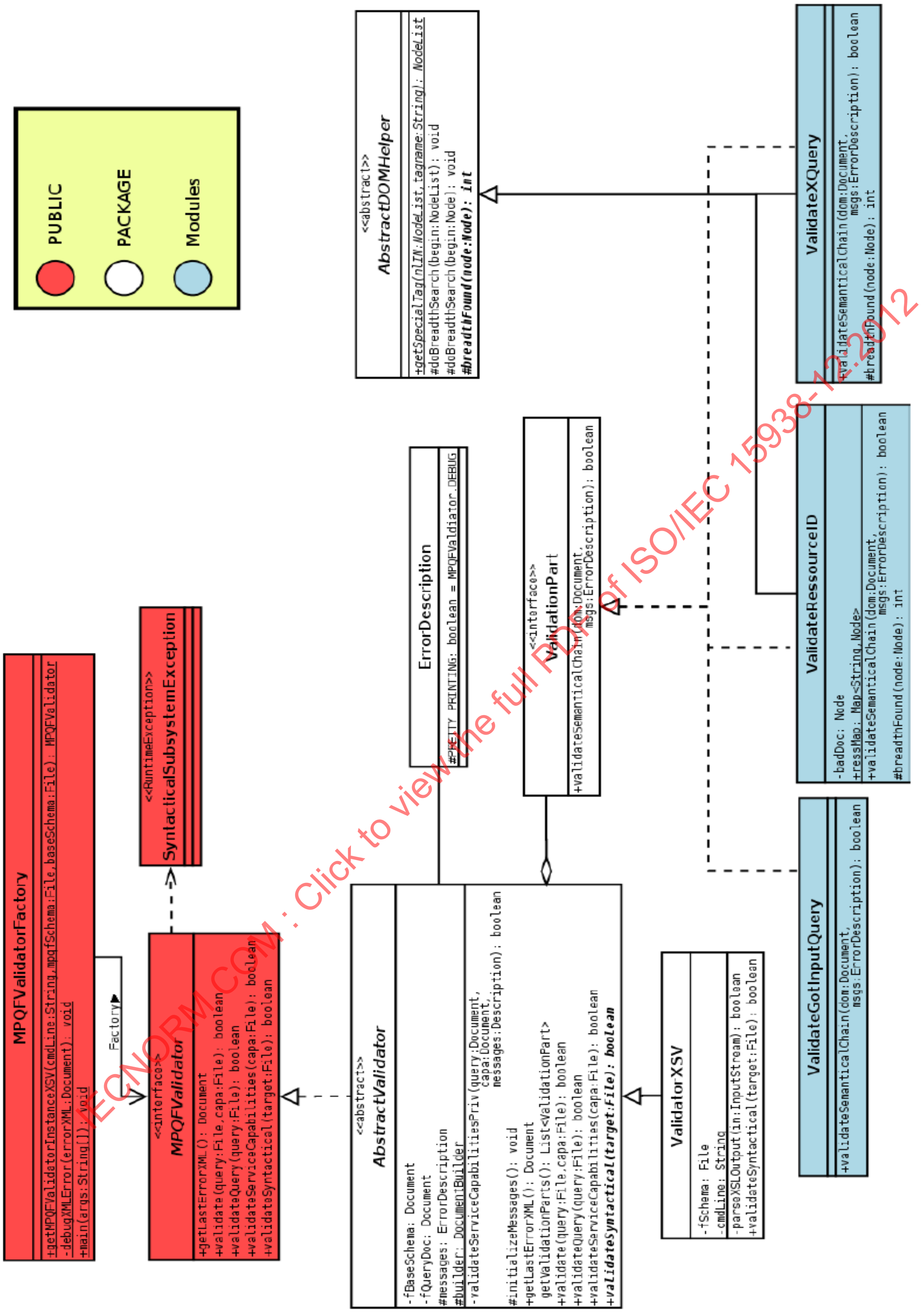


Figure 7 — Workflow of the MPQF validator

15.4.2 Class Hierarchy

Figure 8 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.



PUBLIC
PACKAGE
Modules
<<abstract>>
AbstractDOMHelper
+getSpecialTag(nlIN:NodeList,tagname:String): NodeList
#doBreadthSearch(begin:NodeList): void
#doBreadthSearch(begin:Node): void
#breadthFound(node:Node): int
ErrorDescription
#PRETTY_PRINTING: boolean = MPQFValdiator.DEBUG
<<interface>>
ValidationPart
+validateSemanticalChain(dom:Document, msgs:ErrorDescription): boolean
ValidateXQuery
+validateSemanticalChain(dom:Document, msgs:ErrorDescription): boolean
#breadthFound(node:Node): int
ValidateRessourceID
-badDoc: Node
+ressMap: Map<String,Node>
+validateSemanticalChain(dom:Document, msgs:ErrorDescription): boolean
#breadthFound(node:Node): int
ValidateGotInputQuery
+validateSemanticalChain(dom:Document, msgs:ErrorDescription): boolean
MPQFValidatorFactory
+getMPQFValidatorInstanceXSV(cmdLine:String,mpqfSchema:File,baseSchema:File): MPQFValidator
-debugXMLError(errorXML:Document): void
+main(args:String[]): void
Factory
<<RuntimeException>>
SyntacticalSubsystemException
<<interface>>
MPQFValidator
+getLastErrorXML(): Document
+validate(query:File,capa:File): boolean
+validateQuery(query:File): boolean
+validateServiceCapabilities(capa:File): boolean
+validateSyntactical(target:File): boolean
<<abstract>>
AbstractValidator
-fBaseSchema: Document
-fQueryDoc: Document
#messages: ErrorDescription
#builder: DocumentBuilder
-validateServiceCapabilitiesPriv(query:Document, capa:Document, messages:Description): boolean
#initializeMessages(): void
+getLastErrorXML(): Document
getValidationParts(): List<ValidationPart>
+validate(query:File,capa:File): boolean
+validateQuery(query:File): boolean
+validateServiceCapabilities(capa:File): boolean
+validateSyntactical(target:File): boolean
ValidatorXSV
-fSchema: File
-cmdLine: String
-parseXSLOutput(in:InputStream): boolean
+validateSyntactical(target:File): boolean

Figure 8 — 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 9). 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 9. 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.

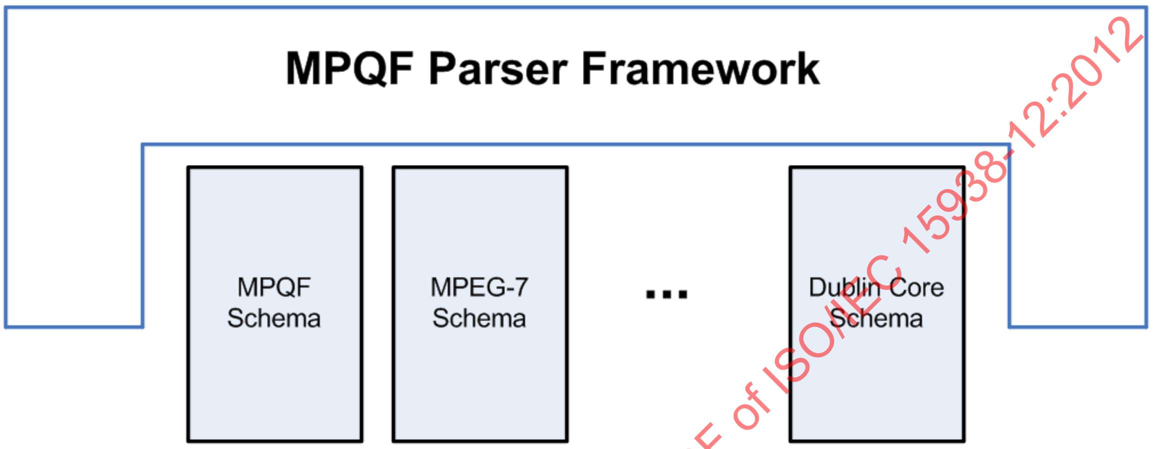


Figure 9 — MPQF Parser Framework

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

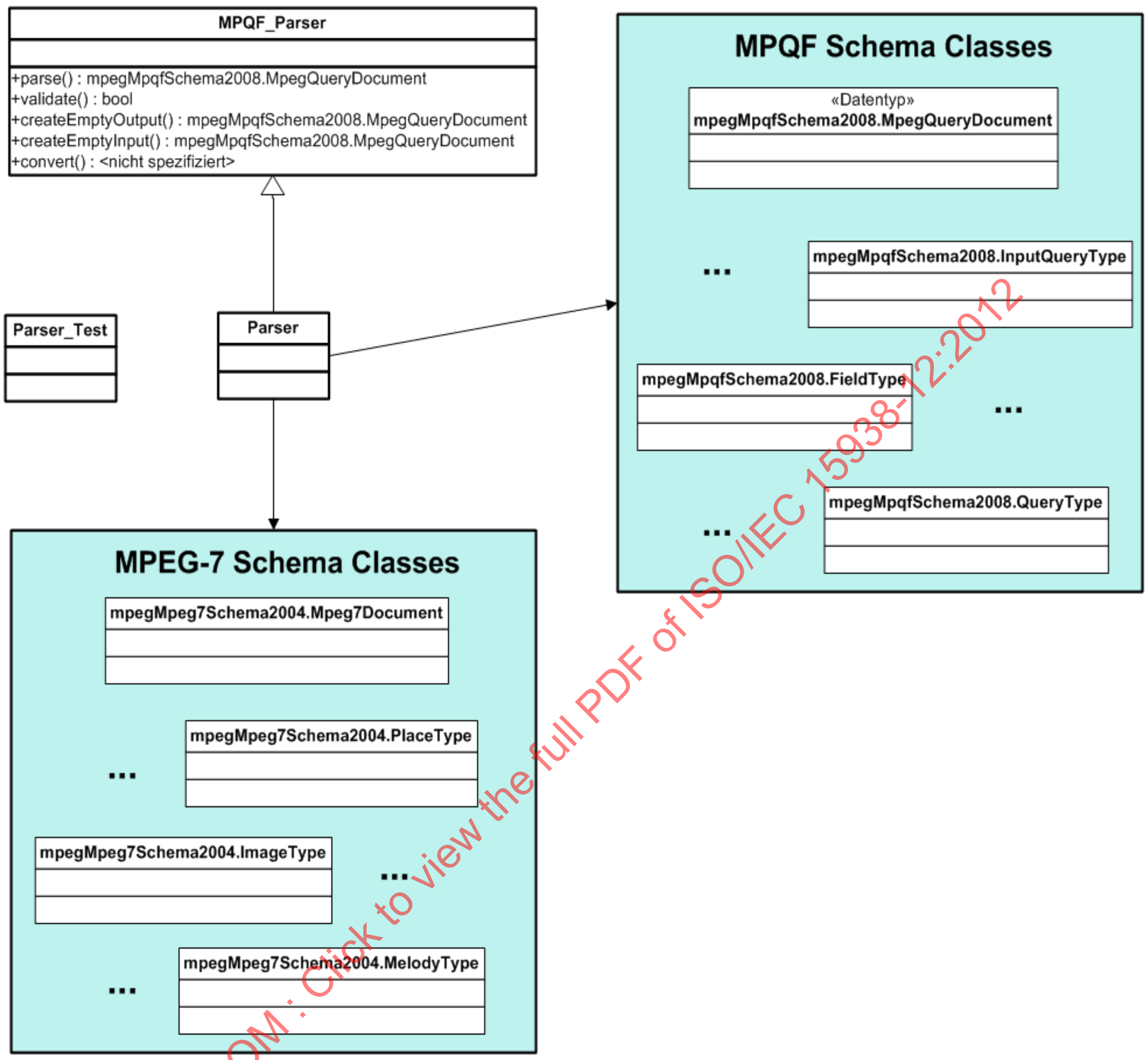

MPQF_Parser
+parse() : mpegMpqfSchema2008.MpegQueryDocument
+validate() : bool
+createEmptyOutput() : mpegMpqfSchema2008.MpegQueryDocument
+createEmptyInput() : mpegMpqfSchema2008.MpegQueryDocument
+convert() : <nicht spezifiziert>
Parser_Test
Parser
MPQF Schema Classes
«Datentyp»
mpegMpqfSchema2008.MpegQueryDocument
mpegMpqfSchema2008.InputQueryType
mpegMpqfSchema2008.FieldType
mpegMpqfSchema2008.QueryType
MPEG-7 Schema Classes
mpegMpeg7Schema2004.Mpeg7Document
mpegMpeg7Schema2004.PlaceType
mpegMpeg7Schema2004.ImageType
mpegMpeg7Schema2004.MelodyType

Figure 10 — MPQF Parser Class Hierarchy

Figure 10 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.

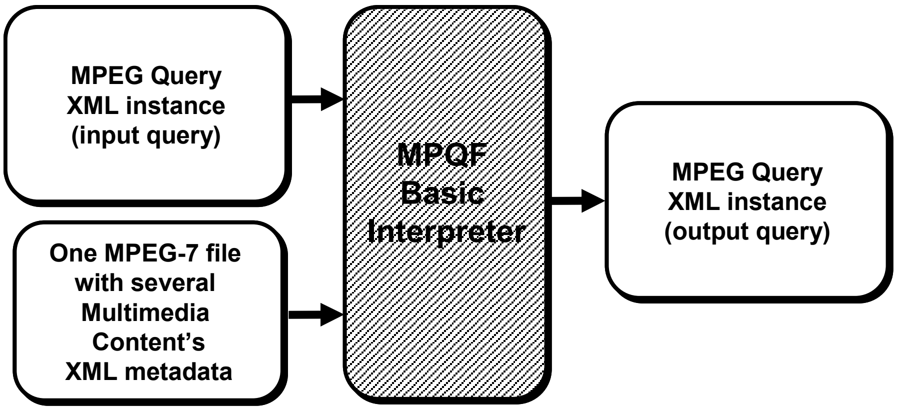

MPEG Query XML instance (input query)
One MPEG-7 file with several Multimedia Content's XML metadata
MPQF Basic Interpreter
MPEG Query XML instance (output query)

Figure 11 — 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

15.6.2.1 Example Utilization

mpqf.bat WEB-INF/classes/xml/test2_1_comparison_equal.xml imageDB/test/xml/mc_metadata1_mpeg7images.xml out.xml

Example input query (*test2_1_comparison_equal.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<MpegQuery xmlns="urn:mpeg:mpqf:schema:2008" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:mpeg:mpqf:schema:2008 mpqf.xsd"
mpqfID="http://www.mpqf.org/id1">
  <Query>
    <Input>
      <QueryCondition>
        <EvaluationPath>//Image</EvaluationPath>
          <Condition xsi:type="Equal">

<ArithmeticField>MediaInformation/MediaProfile/MediaFormat/FileSize</ArithmeticField>
```

```
            <LongValue>10000</LongValue>
          </Condition>
      </QueryCondition>
    </Input>
  </Query>
</MpegQuery>
```

Example MPEG-7 doc (*imageDB/test/xml/mc_metadata1_mpeg7images.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2004" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:mpeg:mpeg7:schema:2004 M7v2schema.xsd"
mediaTimeBase="/child::AA:AA[0]/child::AA:AA[0]" mediaTimeUnit="-P0DT0H0M0S0N0F"
timeBase="/child::AA:AA[0]/child::AA:AA[0]" xml:lang="en-us" timeUnit="-P0DT0H0M0S0N0F-
00:00Z">
  <Description xsi:type="ContentEntityType">
    <MultimediaContent xsi:type="ImageType">
      <Image>
        <MediaInformation>
          <MediaProfile>
            <MediaFormat>
              <Content href="http://www.someprovider.com"/>
              <FileFormat href="http://www.mpeg.org">
                <Name>avi</Name>
              </FileFormat>
              <FileSize>10000</FileSize>
              <VisualCoding>
                <Frame width="1284" height="1000"/>
              </VisualCoding>
            </MediaFormat>
            <MediaInstance>
              <InstanceIdentifier/>
              <MediaLocator>
                <MediaUri>http://www.test.com</MediaUri>
              </MediaLocator>
            </MediaInstance>
          </MediaProfile>
        </MediaInformation>
        <CreationInformation>
          <Creation>
            <Title>Image 1</Title>
            <Creator xsi:type="CreatorType">
              <Role href="http://www.roles.com/director"/>
              <Agent xsi:type="PersonType">
                <Name>
                  <GivenName>A.</GivenName>
                  <FamilyName>Scott</FamilyName>
                </Name>
              </Agent>
            </Creator>
          </Creation>
        </CreationInformation>
      </Image>
    </MultimediaContent>
  </Description>
  <Description xsi:type="ContentEntityType">
    <MultimediaContent xsi:type="ImageType">
      <Image>
```

```
        <MediaInformation>
          <MediaProfile>
            <MediaFormat>
              <Content href="http://www.someprovider.com"/>
              <FileFormat href="http://www.mpeg.org">
                <Name>avi</Name>
              </FileFormat>
              <FileSize>20000</FileSize>
              <VisualCoding>
                <Frame width="1280" height="1000"/>
              </VisualCoding>
            </MediaFormat>
            <MediaInstance>
              <InstanceIdentifier/>
              <MediaLocator>
                <MediaUri>http://www.test.com</MediaUri>
              </MediaLocator>
            </MediaInstance>
          </MediaProfile>
        </MediaInformation>
        <CreationInformation>
          <Creation>
            <Title>Image 2</Title>
            <Creator xsi:type="CreatorType">
              <Role href="http://www.roles.com/director"/>
              <Agent xsi:type="PersonType">
                <Name>
                  <GivenName>B.</GivenName>
                  <FamilyName>Smith</FamilyName>
                </Name>
              </Agent>
            </Creator>
          </Creation>
        </CreationInformation>
      </Image>
    </MultimediaContent>
  </Description>
</Mpeg7>
```

Example resulting output (*out.xml*):

```
<MpegQuery xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mpeg:mpqf:schema:2008" xsi:schemaLocation="urn:mpeg:mpqf:schema:2008 mpqf.xsd"
mpqfID="http://www.mpqf.org/id1">
    <Query>
        <Output currPage="1" totalPages="1" expirationDate="2008-05-30T09:00:00">
            <ResultItem recordNumber="1">
                <TextResult>Image 1</TextResult>
                <MediaResource>http://www.test.com</MediaResource>
            </ResultItem>
        </Output>
    </Query>
</MpegQuery>
```

15.6.3 Java Package org.iso.mpeg.mpqf (for programmatic utilization)

The provided query processor can be used also as embedded in another Java application. The software is divided in several packages, but only one is necessary to access the functionality of the query processor, the package org.iso.mpeg.mpqf. This package contains a set of generic public classes and interfaces which allow indexing content and metadata, and executing MPQF Query requests. If an application uses only org.iso.mpeg.mpqf classes and interfaces, it keeps decoupled from the internal implementation of the query engine, and the metadata and content indexes. This philosophy pursues the maximum interoperability, and allows combining a query engine, a metadata index, and a content index from three different third parties. Any one of the components can be replaced with minimum impact to the host application. The package org.iso.mpeg.mpqf is bundled along with a specific implementation of a query engine, and basic metadata and content indexes from third parties.

A javadoc with the API details has been generated and included in the application bundle.

15.6.3.1 Interface Hierarchy

- org.iso.mpeg.mpqf.**MPQFEngine**
- org.iso.mpeg.mpqf.**ContentIndex**
- org.iso.mpeg.mpqf.**XMLIndex**

15.6.3.2 Class Hierarchy

- java.lang.Object
    - org.iso.mpeg.mpqf.**MPQFEngineFactory**
    - org.iso.mpeg.mpqf.**MPQFOutput**
    - org.iso.mpeg.mpqf.**MPQFQuery**
    - org.iso.mpeg.mpqf.**ResultItem**
    - java.lang.Throwable (implements java.io.Serializable)
        - java.lang.Exception
            - org.iso.mpeg.mpqf.**MPQFException**
    - org.iso.mpeg.mpqf.**ContentIndexFactory**
    - org.iso.mpeg.mpqf.**XMLIndexFactory**

15.6.3.3 Interface MPQFEngine

This is the main component of the API. A particular class implementing the methods of the MPQFEngine interface can be obtained from the MPQFEngineFactory. By default, the UPC – BARCELONA TECH implementation will be used. The pubic methods of MPQFEngine are:

- MPQFQuery compileQuery(java.io.File mpqfFile)
- MPQFQuery compileQuery(java.io.InputStream mpqfQueryStream)
- MPQFQuery compileQuery(java.lang.String mpqfString)
- MPQFOutput executeQuery(MPQFQuery query)

- void setXMLIndex(XMLIndex index)
- void setContentIndex(ContentIndex contentIndex)

Usage steps:

1) Obtain an implementation through the MPQFEngineFactory
2) Register an XMLIndex and a ContentIndex through the setXMLIndex and setContentIndex methods
3) Index some content and metadata (see XMLIndex and ContentIndex interfaces)
4) Compile a query through one of the compileQuery methods
5) Execute the query through the executeQuery method

15.6.3.4 Interface XMLIndex

This interface allows connecting the query engine with external XML metadata databases. A specific class implementing the XMLIndex interface can be obtained from the XMLIndexFactory. The only pubic method necessary for basic usage is:

void indexMetadata(java.io.InputStream xmlStream)

Other methods are available for those who want to implement their own query processors and need to interact with the XML DB. These methods are documented in the javadoc.

15.6.3.5 Interface ContentIndex

This interface allows connecting the query engine with external content databases (video, still images, audio, etc.). A specific class implementing the ContentIndex interface can be obtained from the ContentIndexFactory. . The only pubic method necessary for basic usage is:

void indexContent(java.io.InputStream contentStream)

Other methods are available for those who want to implement their own query processors and need to interact with the content DB. These methods are documented in the javadoc.

15.6.4 Example Utilization

Example Java code

```
MPQFEngine mpqfEngine =
    MPQFEngineFactory.createMPQFEngine("org.barcelonatech.kaiko.MPQFEngineImplUPC");

//Setup XMLIndex
XMLIndex xmlIndex =
    xmlIndex = XMLIndexFactory.createXMLIndex("org.barcelonatech.kaiko.existdriver.
                XMLIndexImplExistEmbedded", "imageDB_index/blank/index-exist", true);
//Index metadata file
InputStream xmlStream = new FileInputStream(“example_mpeg7.xml”));
xmlIndex.indexMetadata(xmlStream);

//Register XMLIndex
mpqfEngine.setXMLIndex(xmlIndex);
```

```
//Query compilation
MPQFQuery mpqfQuery = mpqfEngine.compileQuery(new File(args[0]));

//Query execution
MPQFOutput mpqfOutput = mpqfEngine.executeQuery(mpqfQuery);

for (int i = 0; i<mpqfOutput.resultItemVector.size(); i++) {
    ResultItem ri = (ResultItem)mpqfOutput.resultItemVector.elementAt(i);
    System.out.println("ResultItem:");
    System.out.println(ri.textResult);
    System.out.println(ri.mediaResource);
    System.out.println();
}
```

15.7 MPQF Semantic Enhancement

Module name	/15938-12/ MPQFSE_to_SPARQL.zip
Description	- Transforms the components of the semantic enhancement specification into a valid SPARQL request - Supports the evaluation of the SPARQL request at an attached SPARQL repository - Transforms results of a SPARQL evaluation into a valid MPQF response
INPUT	- An MPQF query coping constructs of the semantic enhancement specification
OUTPUT	- An MPQF xml instance document
Programming Language(s)	Java version 1.5 or higher
Platform(s)	Any platform that supports the programming language
Dependencies	-
Details	-

15.7.1 Introduction

In ISO/IEC 15938-12:2008/PDAM 2 “Semantic enhancement” an amendment of MPQF (named MPQF-SE) is specified to address high level metadata for multimedia in the form of RDF descriptions.

This section describes the reference software for this part of MPQF which allows an automatic transformation of MPQF-SE requests to equivalent SPARQL queries. In series, the SPARQL query can be used for evaluation against RDF-data stores. Described is the MPQF-SE Parser framework, which provides means for a 1 to 1 mapping of MPQF-SE queries to SPARQL instances.

15.7.2 MPQF-SE Parser Framework

The basic functionality of the MPQF-SE-to-SPARQL application is the translation. The formal basis for the translation is described by the mapping rules.

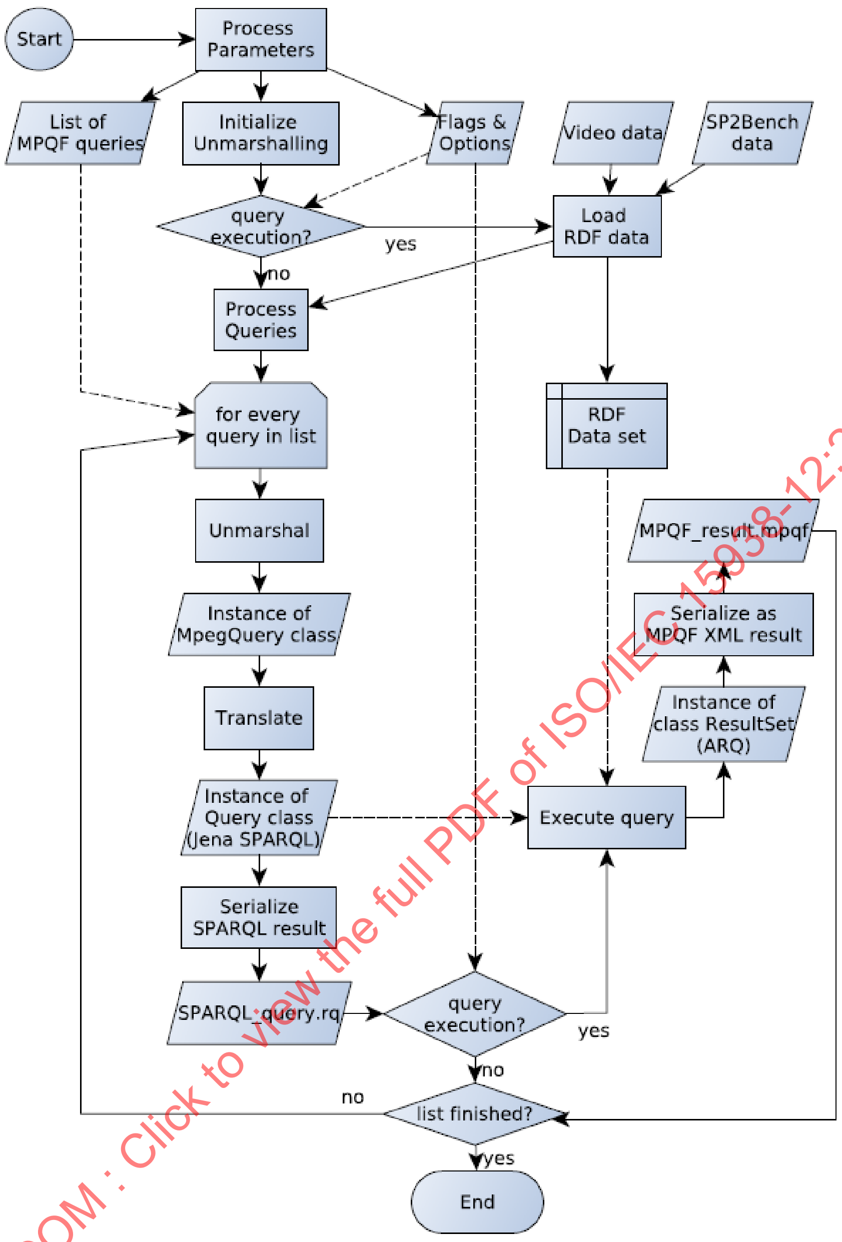


Figure 12 — Workflow of the MPQF-SE parser

Figure 12 provides a diagram showing MPQF-SE-to-SPARQL's basic processing steps. Basically, a given XML file with a semantic MPQF query is first parsed into Java objects. That is, these Java objects are instances of Java classes that represent the different building blocks of an MPQF XML file. This first preprocessing is performed using the JAXB framework and precompiled Java classes that represent MPQF query parts. The parsing of XML files into corresponding Java objects is called *marshalling* in the JAXB framework. The corresponding Java class for the root XML element of an MPQF query is the class MpegQuery. XML child elements of the XML element MpegQuery are represented by instance variables of MpegQuery Java objects. The actual translation which works in correspondence to the mapping rules is performed on the level of Java objects.

The translation takes an instance of class MpegQuery as input and creates an instance of class Query as output. Instances of class Query from the Jena framework represent SPARQL queries. They can be serialized to a String object or they can also be used for query processing. The ARQ framework provides a query engine for the Jena framework. That is, the classes of the ARQ framework complement the classes of the Jena framework so that instances of the Query class from Jena can be executed with class instances from the ARQ framework. Therefore, these two frameworks are frequently regarded as one unit named *Jena/ARQ*.

The main task of translating semantic MPQF queries to SPARQL queries is accomplished as soon as the instances of the class Query from the Jena framework is returned as String or in a file. This SPARQL query string could be used for query execution with any other framework than ARQ as well.

The parser comes with a test database for presenting the evaluation of transformed MPQF-SE requests in a Jena RDF database.

15.7.3 Mapping Concepts

A *semantic MPQF query* is a query which is specified with the new query constructs of the *MPQF Semantic Enhancement* or language constructs which can reasonably be used together with the language constructs of the *MPQF Semantic Enhancement*.

The syntactical structure of an MPQF query and its SPARQL counterpart are very different.

On the one hand, MPQF queries are defined with XML documents, which can have a deeply nested structure. The structure of MPQF query documents is formally described by an XML Schema document.

On the other hand, a SPARQL query is written in a syntax that is inspired by SQL. The structure of SPARQL queries is specified by a grammar file[1] which is written in an *Extended Backus-Naur Form (EBNF)* notation defined in [5]. This EBNF is W3C specific and differs from the EBNF defined in *ISO/IEC 14977:1996(E)*.

The formal mapping of MPQF-SE to SPARQL is described in [6] and bases on language constructs of *XPath 2.0* [10], the *XQuery 1.0 and XPath 2.0 Data Model (XDM)* [11] and functions of the *XQuery 1.0 and XPath 2.0 Functions and Operators* [12] are used. Furthermore, datatypes defined in *XML Schema Part 2: Datatypes* [13] are used.

15.7.4 Installation / Utilization

The MPQF-SE-to-SPARQL parser is delivered as ZIP archive file with the name MPQFSE_to_SPARQL.zip[2]. It contains the MPQF-SE-to-SPARQL application packed as JAR file with the name MPQF-SE-to-SPARQL<version>.jar. The ZIP archive file also includes libraries needed to execute the MPQF-SE-to-SPARQL application in the directory lib, MPQF query example files in the directory queryExamples, and two RDF data sets in the directory example-dataset.

The application uses several libraries from other frameworks. The most relevant libraries are related to the *Jena/ARQ framework* [7] and the *JAXB framework* [8]. Furthermore, the application uses data and test queries created by the *P2Bench* [9] SPARQL benchmark tools.

The application requires a Java Runtime Environment with version 6 or higher. Perform the following steps in order to launch the MPQF-SE-to-SPARQL application.

1. Unzip the file MPQF-SE-to-SPARQL.zip.
2. Open a terminal and change to the directory that contains the unzipped files.
3. Launch the application by typing the following command in the terminal:

```
java -jar MPQF-SE-to-SPARQL<version_number>.jar [-e] [MPQF_query_file(s)]
```

⇒ The application accepts one or more files containing a semantic MPQF query. This is indicated by [MPQF_query_file(s)].

1) http://www.w3.org/2001/sw/DataAccess/rq23/parsers/sparql.bnf.

2) Library can be obtained at http://www.dimis.fim.uni-passau.de/iris/mpqf/MPQF_SE_to_SPARQL.zip.

⇒ The *optional* parameter -e has the effect that a given semantic MPQF query is not only translated into a SPARQL query but the translated SPARQL query is also evaluated against the included example data set. If no test evaluation is desired this parameter should be omitted.

After successful transformation, the parser results in a *.rq file containing the equivalent SPARQL string of the input query.

In case of a desired evaluation (indicated by the –e parameter), a MPQF related XML document (extension .mpqf) is stored containing the results of the SPARQL query.