INTERNATIONAL STANDARD

ISO/IEC 15938-6

Second edition
2020-06

Information technology — Multimedia content description interface —

Part 6: Reference software

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

Partie 6: Logiciel de référence

Reference number
ISO/IEC 15938-6:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

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. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document 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 15938-6:2003), which has been technically revised. It also incorporates the Amendments ISO/IEC 15938-6:2003/Amd.1:2006, ISO/IEC 15938-6:2003/Amd.1:2006/Cor.1:2007, ISO/IEC 15938-6:2003/Amd.2:2007, ISO/IEC 15938-6:2003/Amd.3:2010 and ISO/IEC 15938-6:2003/Amd.4:2011. The main changes compared to the previous edition are as follows:

— all previous Amendments have been incorporated and the electronic attachments have been updated;

— minor editorial corrections have been made throughout the document to fully align with ISO/IEC Directives.

A list of all parts in the ISO/IEC 15938 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document provides a standardized set of technologies for describing multimedia content. It addresses a broad spectrum of multimedia applications and requirements by providing a metadata system for describing the features of multimedia content.

The software is available at: https://standards.iso.org/iso-iec/15938/-6/ed-2/en.

The following are specified in this document:

Description schemes (DS) describe entities or relationships pertaining to multimedia content. Description schemes specify the structure and semantics of their components, which can be description schemes, descriptors, or datatypes.

Descriptors (D) describe features, attributes, or groups of attributes of multimedia content.

Datatypes are the basic reusable datatypes employed by description schemes and descriptors.

Systems tools support delivery of descriptions, multiplexing of descriptions with multimedia content, synchronization, file format, and so forth.

This document contains simulation software for tools defined in ISO/IEC 15938-1, ISO/IEC 15938-2, ISO/IEC 15938-3, ISO/IEC 15938-4 and ISO/IEC 15938-5. This software has been derived from the verification models used in the process of developing this series.

Where multimedia content extraction or multimedia content description software is provided, attention is called to the fact that these software modules are provided for the purpose of creating bitstreams of descriptors and description schemes with normative syntax. The performance of these software tools is not indicative of that which can be obtained from implementations where quality and computational optimization are given priority. The techniques used for extracting descriptors or deriving description schemes are not specified by this document. This information can be found in the corresponding Part of the ISO/IEC 15938 series.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents.

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

Information technology — Multimedia content description interface —

Part 6: Reference software

1 Scope

The ISO/IEC 15938 series operates on and generates conformant bitstreams. This document provides a specific implementation that behaves in a conformant manner.

NOTE 1 Other implementations that conform to the ISO/IEC 15938 series are possible, which do not necessarily use the algorithms or the programming techniques of the reference software.

The software contained in this document is known as eXperimentation Model (XM) and is divided into five categories:

a) Binary format for MPEG-7 (BiM). This software converts DDL (XML) based descriptions to binary format and vice versa as explained in Clause 5.

b) DDL parser and DDL validation parser. The components of this software module are specified in Clause 6.

c) Visual descriptors. This software creates standard visual descriptions from associated (visual) media content as explained in Clause 7.

NOTE 2 The techniques used for extracting descriptors are informative, and the quality and complexity of these extraction tools has not been optimized.

d) Audio descriptors. This software creates standard descriptions from associated (audio) media content as explained in Clause 8.

NOTE 3 The techniques used for extracting descriptors are informative, and the quality and complexity of these extraction tools has not been optimized.

e) Multimedia description schemes. This software modules provide standard descriptions of multimedia description schemes as specified in Clause 9.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

No terms and definitions are listed in this document.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

4 Abbreviated terms

AV	audio-visual
CS	coding scheme
D	descriptor
Ds	descriptors
DCT	discrete cosine transform
DDL	description definition language
DS	description scheme
DSs	description schemes
MDS	multimedia description schemes
MPEG	moving picture experts group
MPEG-7	multimedia content description interface standard (the ISO/IEC 15938 series)
XML	eXtensible Markup Language

5 Copyright disclaimer for software modules

The source code for this document can be found at https://standards.iso.org/iso-iec/15938/-6/ed-2/en/.

Each source code module in this document contains a copyright disclaimer which shall not be removed from the source code module.

In the text of each copyright disclaimer, <MPEG standard> is replaced with a reference to its associated specification, e.g. MPEG-7 Systems (ISO/IEC 15938-1), MPEG-7 Visual (ISO/IEC 15938-3), MPEG-7 Audio (ISO/IEC 15938-4), MPEG-7 Multimedia description schemes (ISO/IEC 15938-5).

"This software module was originally developed by <FN1> <LN1> *(*<CN1>*) and edited by* <FN2> <LN2> *(*<CN2>*),* <FN3> <LN3> *(*<CN3>*) in the course of development of the* <MPEG standard>*. This software module is an implementation of a part of one or more* <MPEG standard> *tools as specified by the* <MPEG standard>*. ISO/IEC gives users of the* <MPEG standard> *free license to this software module or modifications thereof for use in hardware or software products claiming conformance to the* <MPEG standard>*. Those intending to use this software module in hardware or software products are advised that its use may infringe existing patents. The original developer of this software module and his/her company, the subsequent editors and their companies, and ISO/IEC have no liability for use of this software module or modifications thereof in an implementation. Copyright is not released for non* <MPEG standard> *conforming products.* <CN1> *retains full right to use the code for his/her own purpose, assign or donate the code to a third party and to inhibit third parties from using the code for non* <MPEG standard> *conforming products. This copyright notice must be included in all copies or derivative works. Copyright ©20xx-20xx".*

<FN>=*First Name,* <LN>=*Last Name,* <CN>=*Company Name*

6 XM software architecture

6.1 Block diagrams

This clause provides information about the XM software architecture. The block diagrams give short overviews and introduce individual components of the XM software.

The composing elements of the MPEG-7 reference software are characterized by their functionality and by their interfaces. They can be configured according to what here is referred as "key applications". From a functional point of view, they can be distinguished as follows:

— "extraction applications" (a description data base is built from a media database) as illustrated in Figure 1;

— "search and retrieval applications" (a description is compared with the descriptions in a database to find the one with the lowest distance) as illustrated in Figure 2;

— "transcoding applications" (a media data base is converted into another media database based on its description) as illustrated in Figure 3.

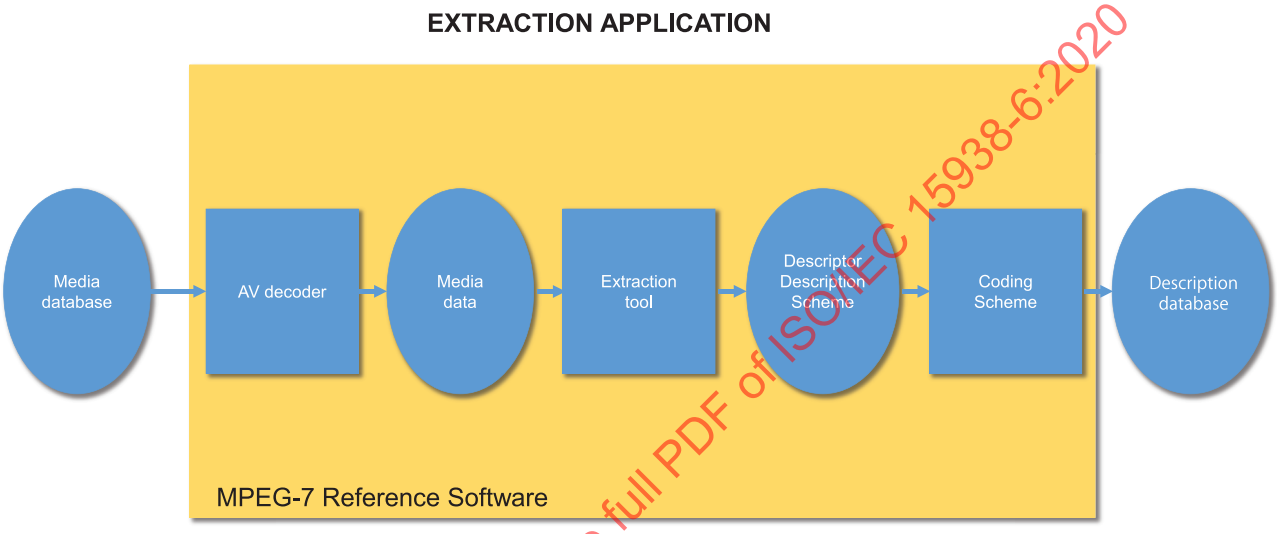

Figure 1 — An "extraction application" using the XM reference software modules, with boxes representing procedural parts and circles representing data structures

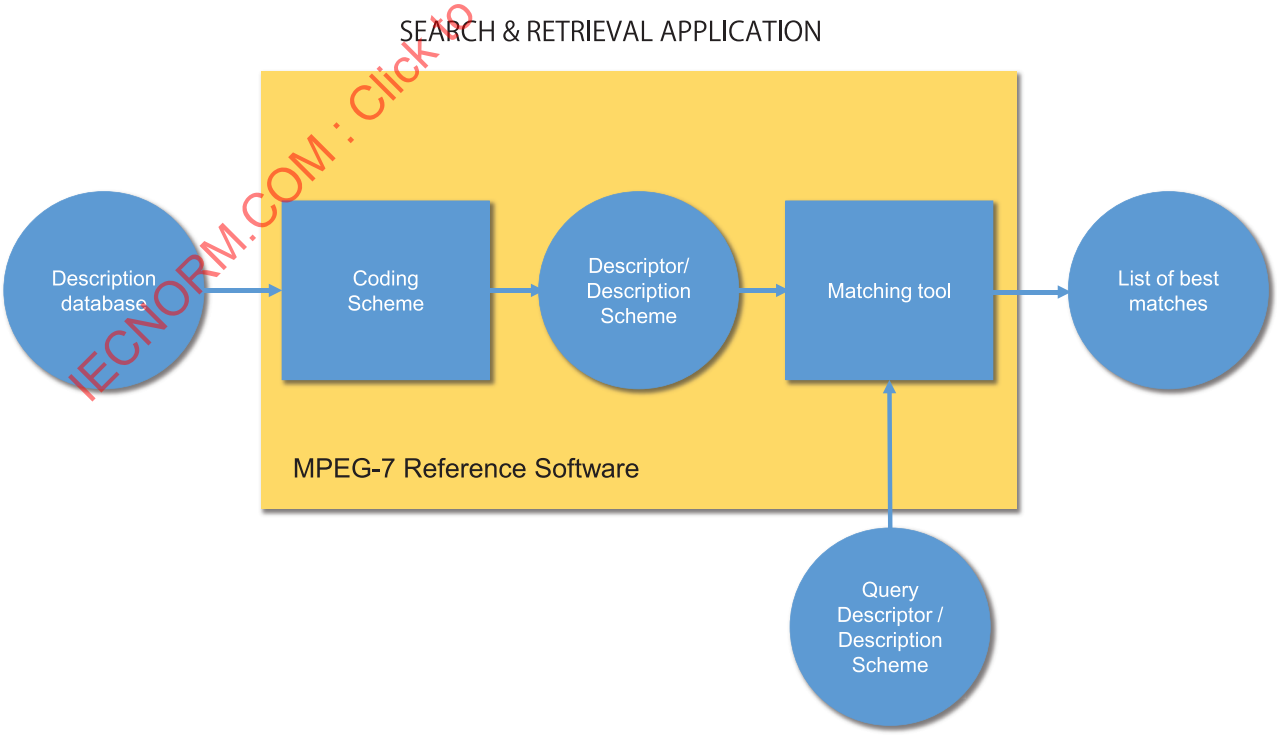

Figure 2 — A "search and retrieval application" using the XM reference software modules, with boxes representing procedural parts and circles representing data structures

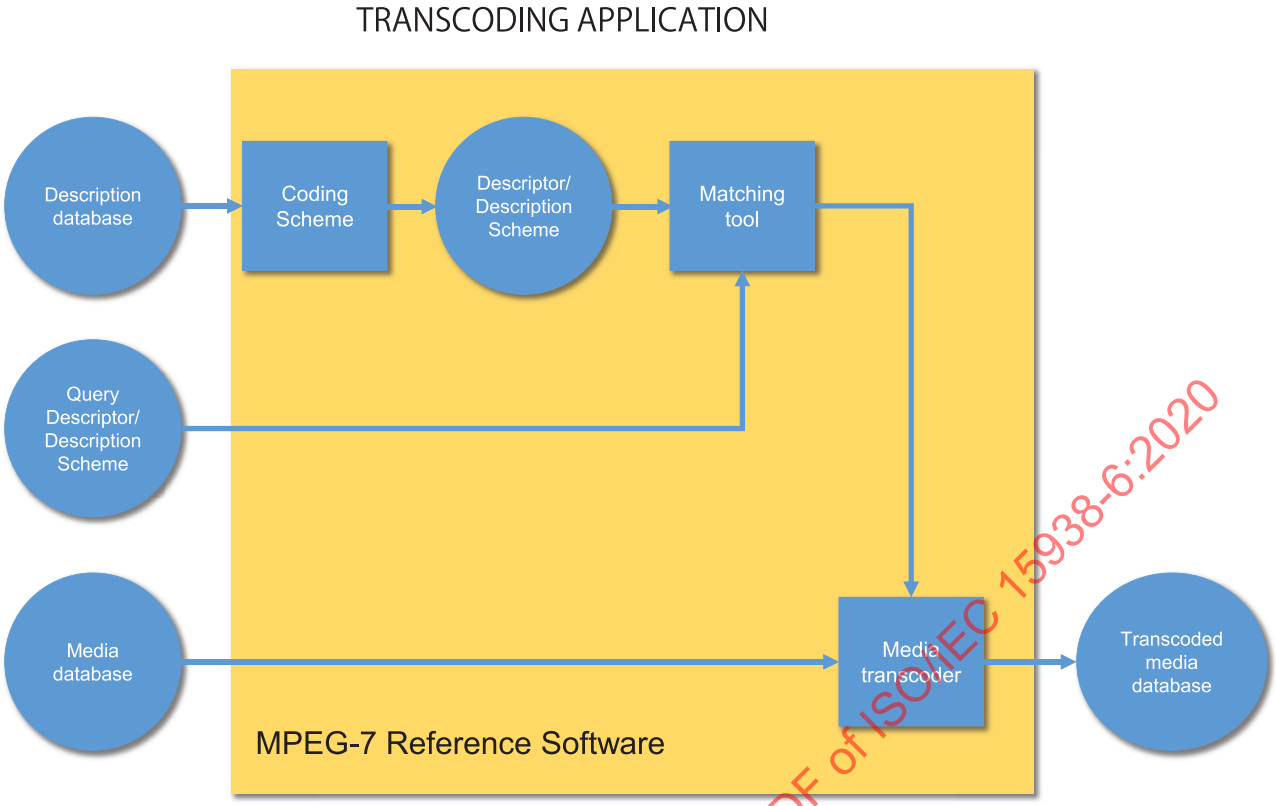

Figure 3 — A "transcoding application" using the XM reference software modules, with boxes representing procedural parts and circles representing data structures

6.2 Block descriptions

6.2.1 General

In subclauses 6.2.2 through 6.2.11, the blocks of the "key applications" are distinguished. For elements that are related to specific descriptors or description schemes, the interface is given using a DummyType example. This represents the XM integration template and not a normative descriptor or description scheme.

6.2.2 Media database

The media database contains media files, which are supported as input files by the AV decoders. The database file is read from a file and contains one media filename per line. From this media filename all additional input and output filenames can be derived.

6.2.3 AV decoders

The XM supports the following AV decoders:

— Still image decoders: ImageMagick®[1)] (Ver. 6.8.8 linked as external library, not included in the XM reference software distribution).

— MPEG-1, MPEG-2 video decoders: (XM directory: Decoders/MPEG2Dec).

1) ImageMagick® is the trademark of a product supplied by ImageMagick Studio LLC. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

— MPEG-1 video motion vector extractor: (XM directory: Decoders/MPEG2Dec). This extracts images and motion vectors.

— 3D Objects: (XM directory: Media). This reads a 3D object for 3D shape descriptors.

— Key Points: (XM directory: Media). This reads in a list of key points from a file.

— Audio decoders: (XM directory: Media). They read audio files.

6.2.4 Media data

This is the internal XM representation of the raw media data (one class with different structures depending on the media content type). The class description for media data can be found in the Media XM directory.

6.2.5 Extraction tools

Extraction tools are specific extraction methods defined for each descriptor and description scheme. All the source files are available in the ExtractionUtilities XM directory. The extraction tools extract the descriptions from media data. Because media data can be of significant size, the extraction is performed on time entities of the media, i.e., if the media is a video the extraction is done frame by frame. Some of the extraction tools may need OpenCV linked with XM as an external library. The interface of the DummyType extraction tool (implementation template) is:

```
//============================================================================
class DummyTypeExtractionTool: public DescriptorExtractor
{
  friend DummyTypeExtractionInterface;
public:
  // Null constructor
  DummyTypeExtractionTool();

  // Also connects the Descriptor (result memory) to the extraction
  // If set to "0" it automatically creates the descriptor
  DummyTypeExtractionTool(DummyTypeDescriptorInterfaceABC
           *DummyType);

  // ID of object type
  virtual const UUID& GetObjectID(void);
  // Object type name
  virtual const char *GetName(void);

// This informs the extractor where the source data comes from
  virtual int SetSourceMedia(MultiMediaInterfaceABC* media);

// Pointer where the description is stored
  virtual DummyTypeDescriptorInterfaceABC*
    GetDescriptorInterface(void);
  virtual int SetDescriptorInterface(DummyTypeDescriptorInterfaceABC
               *aDummyTypeDescriptorInterface);

  // initialize descriptor and extraction process (input media must be known)
  virtual unsigned long InitExtracting(void);

  // performs extraction from input media frame by input media frame
  virtual unsigned long StartExtracting(void);

  // collects descriptor data after all input media frames were processed
  virtual unsigned long PostExtracting(void);

  // Extraction object must not be used, only its interface is allowed
  // to be used. This function is to get the interface
  virtual DummyTypeExtractionInterfaceABC *GetInterface(void);

  // access is allowed only by class factories for this
  // object.  This avoids having to duplicate the
  // ID definition in multiple locations.  In the future, we may
```

```
  // have to do this.  PLEASE DO NOT USE THESE UNLESS YOU ARE
  // IMPLEMENTING A CLASS FACTORY GENERATING THIS OBJECT
  static const UUID myID;
  static const char * myName;
private:
  // Destructor is private to allow creation of
  // object only by using "new"
  virtual ~DummyTypeExtractionTool();

  DummyTypeExtractionInterface m_Interface;
  DummyTypeDescriptorInterfaceABC *m_DescriptorInterface;
  MultiMediaInterfaceABC* m_Media;

  // only used in this dummy type to show extraction function
  unsigned long m_FrameCnt;

#ifdef __HasSubTypes /*include this section if sub-descriptors exist,
             remove this section if no sub-descriptors exist*/
  SubDummyTypeAExtractionInterfaceABC *m_SubDummyTypeAExtraction;
  SubDummyTypeBExtractionInterfaceABC *m_SubDummyTypeBExtraction;
#endif /* __HasSubTypes*/
  int m_DummyExtractionParameter;
}; // End class
//============================================================================
```

6.2.6 Descriptors (Ds) and description schemes (DSs)

These modules implement the data structure of normative descriptors and description schemes. Low level video descriptors are using a dedicated C++ class. This class provides methods to access the elements of the normative descriptions. The source files are located in the Descriptors directory. All other normative Ds and DSs are using the GenericDS class located in the DescriptionSchemes directory. The GenericDS class does not implement the data structure in a dedicated way, but it is an interface to the XML parser library which controls the memory for the tree structure of the instantiated D or DS. The interface of the descriptors class is given for the DummyType descriptor (implementation template) below:

```
/============================================================================
class DummyTypeDescriptor: public Descriptor
{
friend DummyTypeDescriptorInterface;
public:
  DummyTypeDescriptor();

#ifdef __HasSubTypes /*include this section if sub-descriptors exist,
             remove this section if no sub-descriptors exist*/
  // constructor which also constructs and/or connects the descriptor object
  DummyTypeDescriptor(SubDummyTypeADecriptorInterfaceABC *aSubDummyTypeA,
            SubDummyTypeBDescriptorInterfaceABC *aSubDummyTypeB);
#endif /* __HasSubTypes*/

virtual const UUID& GetValueID(void);
  virtual const char* GetValueName(void);

  virtual const UUID& GetObjectID(void);
  virtual const char *GetName(void);

  // for reference counting
  virtual void addref();
  virtual void release();

#ifdef __HasSubTypes /*include this section if sub-descriptors exist,
             remove this section if no sub-descriptors exist*/
  /* only needed for manual connection with sub components*/
  virtual SubDummyTypeADescriptorInterfaceABC
    *GetSubDummyTypeADescriptorInterface(void);
  virtual unsigned long
    SetSubDummyTypeADescriptorInterface(SubDummyTypeADescriptorInterfaceABC
                *aSubDummyTypeADescriptorInterface);

  virtual SubDummyTypeBDescriptorInterfaceABC
```



```
    *GetSubDummyTypeBDescriptorInterface(void);
  virtual unsigned long
    SetSubDummyTypeBDescriptorInterface(SubDummyTypeBDescriptorInterfaceABC
                *aSubDummyTypeBDescriptorInterface);
#endif /* __HasSubTypes*/

  // actual descriptor methods, only in this dummy type example
  virtual long GetDummyContents(void);
  virtual void SetDummyContents(const long val);

  // transformation to GenericDS object (MDS implementaion style)
  virtual unsigned long
    ExportDDL(GenericDSInterfaceABC *aParentDescription);
  virtual unsigned long ImportDDL(GenericDSInterfaceABC *aDescription);

  // access is allowed only by class factories for this
  // object.  This avoids having to duplicate the
  // ID definition in multiple locations.  In the future, we may
  // have to do this.  PLEASE DO NOT USE THESE UNLESS YOU ARE
  // IMPLEMENTING A CLASS FACTORY GENERATING THIS OBJECT
  static const UUID myID;
  static const char * myName;

  virtual DummyTypeDescriptorInterfaceABC *GetInterface(void);

private:
  // private destructor to force reference counting mechanism
  virtual ~DummyTypeDescriptor();

  // reference counter
  unsigned long m_refcount;

  DummyTypeDescriptorInterface m_Interface;

  const bool m_isProprietary;
  static const char * valName;
  static const UUID valID;

#ifdef __HasSubTypes /*include this section if sub-descriptors exist,
            remove this section if no sub-descriptors exist*/
  SubDummyTypeADescriptorInterfaceABC *m_SubDummyTypeADescriptorInterface;
  SubDummyTypeBDescriptorInterfaceABC *m_SubDummyTypeBDescriptorInterface;
#endif /* __HasSubTypes*/

 // This is the actual data the D/DSType stores. In this particular
 // dummy example it's just a signed long called m_DummyContents
  long m_DummyContents;
};
/==========================================================================
```

6.2.7 Coding schemes (CSs)

Coding schemes are specific coding and decoding methods defined for individual descriptors (Ds) and description schemes (DSs). All the source files are available in the CodingSchemes directory. All normative coding schemes are located in the CodingSchemes directory. Coding schemes are available for the visual descriptors to encode or to decode a description into its binary representation. The alternative implementation of the coding scheme box, which is available for all Ds and DSs, allows encoding and decoding of descriptors into its DDL representation using the GenericDSCS which is an interface to the "write to file" and "read from file" functions of XML Xerces-C++ external parser library. The interface of the coding schemes is given below on the example of the DummyType coding scheme (implementation template):

```
//==========================================================================
class DummyTypeCS: public DescriptionCodingEngine
{
friend DummyTypeCSInterface;
public:
 DummyTypeCS();
```

```
// constructor which also constructs and/or connects the descriptor object
DummyTypeCS(DummyTypeDescriptorInterfaceABC
        *DummyType);

virtual const UUID& GetValueID(void);
 virtual const char* GetValueName(void);

 virtual const UUID& GetObjectID(void);
 virtual const char *GetName(void);

 // access is allowed only by class factories for this
 // object.  This avoids having to duplicate the
 // ID definition in multiple locations.  In the future, we may
 // have to do this.  PLEASE DO NOT USE THESE UNLESS YOU ARE
 // IMPLEMENTING A CLASS FACTORY GENERATING THIS OBJECT
 static const UUID myID;
 static const char * myName;

 virtual DummyTypeCSInterfaceABC *GetInterface(void);

 // accessor methods
 virtual EncoderFileIO *GetEncoderStreamBuffer(void);
 virtual int SetEncoderStreamBuffer(EncoderFileIO *aBuffer);
 virtual DecoderFileIO *GetDecoderStreamBuffer(void);
 virtual int SetDecoderStreamBuffer(DecoderFileIO *aBuffer);

 virtual DummyTypeDescriptorInterfaceABC
   *GetDescriptorInterface(void);
 virtual int SetDescriptorInterface(DummyTypeDescriptorInterfaceABC
                *aDummyTypeDescriptorInterface);

  //this function writes the description via the encoder buffer to a file
 virtual int StartEncode();
//this function reads the description via the decoder buffer from a file
 virtual int StartDecode();

private:
  // private destructor to allow construction of objects only by "new"
 virtual ~DummyTypeCS();

 DummyTypeCSInterface m_Interface;

 static const char * valName;
 static const UUID valID;

 // descriptor data
 EncoderFileIO *m_EncoderBuffer;
 DecoderFileIO *m_DecoderBuffer;
 DummyTypeDescriptorInterfaceABC *m_DescriptorInterface;
#ifdef __HasSubTypes /*include this section if sub-descriptors exist,
             remove this section if no sub-descriptors exist*/
 SubDummyTypeACSInterfaceABC *m_SubDummyTypeACS;
 SubDummyTypeBCSInterfaceABC *m_SubDummyTypeBCS;
#endif /* __HasSubTypes*/

};
//========================================================================
```


6.2.8 Search and matching tools

Search tools are specific search or matching methods defined for each descriptor (D) and description scheme (DS). All the source files are available in the SearchUtilities XM directory. Matching tools are not normative in the implementation, but they are depending on the specified application of the description.

The search tools can appear in two different ways: for computing distances between descriptions for the purpose of search and retrieval, and for searching in the descriptions based on a query for the purpose of transcoding. The interface of the matching tool in the case of distance computation is given with the example of the DummyType search tool. Here the whole description is processed in one step. The

interface of the search tool for search and retrieval is given below on the example of the DummyType search tool (implementation template):

```
//===========================================================================
class DummyTypeSearchTool: public Search
{
friend DummyTypeSearchInterface;
public:
  DummyTypeSearchTool();
  // constructor which also constructs and or connects the descriptor object
  DummyTypeSearchTool(DummyTypeDescriptorInterfaceABC
            *aQueryDescriptorInterface);

  virtual const UUID& GetObjectID(void);
  virtual const char *GetName(void);

  virtual DummyTypeSearchInterfaceABC *GetInterface(void);

  virtual int SetRefDescriptorInterface
    (DummyTypeDescriptorInterfaceABC
     *aDummyTypeDescriptorInterface);
  virtual DummyTypeDescriptorInterfaceABC*
    GetQueryDescriptorInterface(void);
  virtual int SetQueryDescriptorInterface
    (DummyTypeDescriptorInterfaceABC
     *aDummyTypeDescriptorInterface);

  // function to be called for computing the distance between the query and the reference
description
  virtual double GetDistance(void);

  static const UUID myID;
  static const char * myName;
private:
  // private destructor to force construction of objects only by using new
  virtual ~DummyTypeSearchTool();

  DummyTypeSearchInterface m_Interface;
  DummyTypeDescriptorInterfaceABC *m_RefDescriptorInterface;
  DummyTypeDescriptorInterfaceABC *m_QueryDescriptorInterface;
#ifdef __HasSubTypes /*include this section if sub-descriptors exist,
             remove this section if no sub-descriptors exist*/
  SubDummyTypeASearchInterfaceABC *m_SubDummyTypeASearch;
  SubDummyTypeBSearchInterfaceABC *m_SubDummyTypeBSearch;
#endif /* __HasSubTypes*/
};
//===========================================================================
```

The interface in case of transcoding applications is given with the example of the DummyType search tool (distinguished from the previous case with a define in the template file). Here media data is processed which can be very big, e.g., in the case of video data. Therefore, the search is performed on temporal sub-entities of the media, i.e., in the case of video data the search is performed on a frame by frame basis. The interface of the search tool for transcoding is given below on the example of the DSDummyType search tool (implementation template for MDS):

```
//===========================================================================
class DSDummyTypeSearchTool: public Search
{
friend DSDummyTypeSearchInterface;
public:
  DSDummyTypeSearchTool();
  // constructor allowing to connect also to the description
  DSDummyTypeSearchTool(GenericDSInterfaceABC
         *aQueryDescriptionInterface);

  virtual const UUID& GetObjectID(void);
  virtual const char *GetName(void);

  // pointer to my interface
```

```
  virtual DSDummyTypeSearchInterfaceABC *GetInterface(void);

  virtual int SetRefDescriptionInterface
    (GenericDSInterfaceABC *aDSDummyTypeDescriptionInterface);
  virtual GenericDSInterfaceABC* GetQueryDescriptionInterface(void);
  virtual int SetQueryDescriptionInterface
    (GenericDSInterfaceABC *aDSDummyTypeDescriptionInterface);

  // set the media for transcoding
  virtual int SetMedia(MultiMediaInterfaceABC* media); /* needed ,e.g.,
        to read the time of the image*/
  // called before the first media frame is processed (media must be set)
  virtual double InitSearch(void);
  // called for each media frame
  virtual double StartSearch(void);
  // called after the last media frame was processed
  virtual double PostSearch(void);

  static const UUID myID;
  static const char * myName;
private:
  // private destructor to force construction of objects only by using new
  virtual ~DSDummyTypeSearchTool();

  DSDummyTypeSearchInterface m_Interface;
  GenericDSInterfaceABC *m_RefDescriptionInterface;
  GenericDSInterfaceABC *m_QueryDescriptionInterface;
MultiMediaInterfaceABC* m_Media;

#ifdef __HasSubTypes /*include this section if sub-descriptors exist,
             remove this section if no sub-descriptors exist*/
  SubDSDummyTypeBSearchInterfaceABC *m_SubDSDummyTypeBSearch;
  SubDSDummyTypeBSearchInterfaceABC *m_SubDSDummyTypeBSearch;
#endif
};
//=============================================================================
```

6.2.9 Media transcoders

These procedural blocks are part of the functionality of specific application modules. They are not represented by dedicated module classes in the XM software. They need to be integrated in the XM when implementing a specific transcoding application.

6.2.10 Applications

Applications are expressed by the classes combining the modules of a descriptor or a description scheme including modules of their sub-Ds and -DSs. The resulting class implements one of the three key applications specified above. The source files are located in the applications XM directory. Applications creating a database of the descriptor or description scheme under test (DUT/DSUT), which are of the extraction application type, are called server applications. Applications using the DSUT data base (search and retrieval and transcoding) are called client applications.

6.2.11 Interface structure

The components of the reference software, which are corresponding to a descriptor or description scheme, are implemented using a specific interface mechanism. Besides using private and public functions, all classes have an individual interface class which interfaces the public methods of the class itself. This is done to increase the reusability of the code. For example, by making all destructors private it is possible to force a dedicated way of instantiating objects of this class. Furthermore, the interface function has a pure virtual representation by its InterfaceABC class (ABC = Abstract Base Class), which is always used to access the elements of the classes mentioned in the previous sections.

For the reuse of classes, two mechanisms are implemented. In the case of descriptors implemented with a C++ class (i.e., for visual descriptors) not only the data structure with its methods can be reused, but also the description data itself (e.g., multiple visual colour descriptions share the same colour space

description). In this case a reference counting mechanism is implemented to allow correct destruction of the reused description objects.

In the case of the other modules (i.e. the coding scheme classes, the extraction classes, and the search classes), reuse of the objects is not required. Therefore, these classes do not use a reference counting mechanism. Anyway, these classes use the reference counting mechanism of the descriptor class to manage the memory of the description data.

7 Systems reference software (BiM)

7.1 General

Table 1 shows the reference software components for ISO/IEC 15938-1.

Table 1 — Reference software components for ISO/IEC 15938-1

Name of the tool in ISO/IEC 15938-1	Name of the tool in the XM software
bitstream encoder/decoder	SystemTools/BiM
access unit navigation	SystemTools/TextualAccessUnit

7.2 BiM reference software

7.2.1 General

The BiM reference software is the set of sources, libraries, examples and documentation related to the encoding, decoding and handling of the binary XML format defined by MPEG-7. The package contains different tools, and an intuitive GUI application to control the binarization process of a generic XML source; it is then run independently from the XM reference software.

7.2.2 Package content

The BiM reference software contains:

— the implementation sources for the encoding/decoding algorithms;

— the set of the external libraries for building and running the programs (i.e., xerces.jar, xerces-c_1_4.dll, gnu-regexp-1.1.3.jar);

— a set of command line tools and GUI applications to apply the binarization on different sources;

— a set of example files:

  — MPEG-7 XML schema and a set of MPEG-7 files;

  — a set of XML files and associated XML schemas for the binarization of generic files;

— a .dll and its sources containing the functions necessary to encode a textual path in a binary path and to decode a binary path from the bitstream into a textual path;

— short documentation.

7.2.3 Installation and execution

Prerequisites: the Java Sun™[2)] JDK (version 1.2.2 or later) is needed to use the BiM reference software. The software can be unzipped anywhere in the target machine. Some libraries (xerces-c_1_4.dll,

2) Jave Sun™ is the trademark of a product supplied by Sun Microsystems, Inc. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

navigation_path.dll, included) need to be installed in the machine in order to complete the installation. Refer to the readme file for the detailed instruction for your platform.

The execution of the encoding, decoding and GUI tools is controlled by scripts included in the distribution.

7.2.4 Software overview

7.2.4.1 XML schema parsing and validation

The BiM compression algorithm is based on the knowledge of the schema underlying the XML stream. A central part of the reference software is then dedicated to the parsing and validation of XML schema, and to the building of the objects and automatas used during the encoding/decoding.

The com.expway.schema hierarchy contains the main classes for the validation and internal structure building. Among them, com.expway.schema.GeneralSchemaHandler contains the main entry points for the schema handling.

7.2.4.2 Binary encoding

The encoding process uses the information built during the schema parsing to scan the input file and to encode the structure of the document and its leaves. The main class for the encoding process is "**com.expway.binarisation.GeneralBinaryHandler**": it behaves like a SAX ContentHandler, and all the binarization is driven by the input XML.

In the encoding process, simple XML schema types (xsd:string, xsd:enumeration) have built-in encoding rules, but dedicated encoders can also be associated to particular XML schema types to get a fine-grain control over the compression performances of a known XML schema.

The "**com.expway.ref**" package offers an entry point to the command line tools present in the reference software: BiMDecoder, BiMEncoder, BiMGUI.

7.2.4.3 Binary decoding

The decoding process is somewhat simpler because it does not need to parse the XML schema that had been used for the encoding. The decoding uses the decoderConfig information produced in the schema parsing phase.

The decoders entry points are in "**com.expway.binarisation.GeneralDecompression**".

7.2.5 Main packages

The **com.expway.schema**, **com.expway.schema.instance** contain the main algorithms for the schema static analysis and structure building.

The encoding phase is handled by the automata-based algorithm implemented mainly in the **com.expway.tools.automata** and **com.expway.tools.compression** packages.

The **com.expway.tools.expression** package contains the encoding infrastructure for XML schema related types.

The **com.expway.ref** package contain the entry points for the command line tools.

7.3 Access unit navigator

7.3.1 General

The access unit navigation software builds a separate executable, which is called from the XM process.

It is located in the SystemTools/TextualAccessUnit directory of the reference software source tree. For installing the access unit navigation you need to have the Xerces™ [3] 1.6.0 XML parser to be installed (see http://xerces.apache.org/xerces-c). The xercesc-1_6_0.dll file location should be specified while compiling the encoder/decoder modules in their respective project workspaces.

7.3.2 Textual access unit encoder module

This module assumes that you have the necessary fragments already available with their respective path (FUContext) information. An input parameter file (input.par) has to be generated by the user which is passed as a command line argument to the module for the encoding to take place.

The format of the input.par file, which is a text file, is as follows.

Example:

```
DecoderInit                    → Signifies the start of the DecoderInit.
SchemaReference                → Contains a URI reference to the Schema
                                  followed by an optional second URI for
                                  locationHint.

InitialDescription             → Specifies the first Default AccessUnit
SystemsProfLileLevelIndication→ Gives the SystemsProfLileLevelIndication if
                                  present is followed by the level number on
                                  the next line.

AddNode                        → Signifies the Initial Update Command
/                              → Signifies the navigation path (root)
node_main.xml                  → Specifies the node file name stored on disk.
AccessUnit                     → Signifies the start of the subsequent AU
AddNode                        → Signifies the Update Command
/Semantic/                     → Signifies the navigation path
node_01.xml                    → Specifies the node file name stored on disk.



AddNode
/Semantic/                     → Signifies the navigation path
node_02.xml
AccessUnit                     → Signifies the start of a new AU
AddNode
/Semantic/
<beginPayload>                 → Instead of providing a node file location,
                                  include it inline for small payloads.
<SemanticBase id="soccerstadium-obj">
   <Label> <FreeTerm> Soccer stadium </FreeTerm> </Label>
   <MediaOccurrence>
   </MediaOccurrence>
</SemanticBase>
</beginPayload>                → Payload end indicator
```

An output file called TeMAccessUnit.xml would be generated which can be fed to the decoder module.

A sample input.par file is provided in the Sample_InputFiles directory of the encoder module reference software.

7.3.3 Textual access unit decoder module

For this module two parameters are passed. The first one being the encoded file (e.g., TeMAccessUnit.xml) and the second parameter being the name of the file you want the decoded information to be stored in.

3) Xerces™ is the trademark of a product supplied by The Apache Software Foundation. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

The module takes care of the following:

a) It is able parse through the navigation path and traverse through the DOM tree being built. The modules for this are present in the xpathtype.cpp and xpathtype.hpp files.

b) It is able to add, replace and delete nodes in the DOM tree based on the FUContext information. The manner in which this is indicated in the path, e.g., /Semantic/SemanticBase[2] means the second child node of semantic which is SematicBase.

It is also able to parse in terms of relative addressing, e.g.:

./ □ from the current node onwards.

../ □ upward traversal through the tree structure.

Provides for add/delete/replace of attributes in element nodes.

c) It takes care of validating the document being created (i.e., schema validation) with the schema. The schema files have to be present in the same folder as the input files and the reference to the schema file is made in the encoded document itself.

Qname support is available, i.e., namespace support is provided for.

Provision has been provided for reset operation.

Based on these features the decoder is able to parse through an AccessUnit file and generate an output description file.

Some sample encoded files (TeMAccessUnit.xml, TeMAccessUnit01.xml and TeMCompatAU.xml) are provided in the Sample_InputFiles folder in the SystemTools/TextualAccessUnits/Decoder folder.

8 Description definition language (DDL) reference software

Table 2 shows the reference software components for ISO/IEC 15938-2, which are implemented using the external XML parser library.

Table 2 — Reference software components for ISO/IEC 15938-2

Name of the tool in ISO/IEC 15938-2	Name of the tool in the XM software
DDL parser	External XML parser supporting the DOM API and XML schema (e.g., Xerces XML parser from Apache)
DDL validation parser	External XML parser supporting the DOM API and XML schema (e.g., Xerces XML parser from Apache)

9 Video reference software

Table 3 lists the reference software components of ISO/IEC 15938-3. Most of the components of this section include a server application and a client application. The normative descriptors are implemented using a C++ class. All modules have a binary coding scheme according to the binary syntax of the corresponding visual descriptors, and an interface to the XML parser-based implemented description schemes of ISO/IEC 15938-5. Thus the descriptions may be stored in a binary bitstream file following the binary syntax of each visual descriptor, or in an XML file, which may also be converted to a binary XML formal using the BiM tools. The detailed usage instructions for these modules are located in the doc/video directory of the reference software source tree.