INTERNATIONAL STANDARD

ISO/IEC 15938-6

First edition
2003-07-01

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 15938-6:2003(E)

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

Published in Switzerland

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.

ISO/IEC 15938-6 was prepared by Technical Committee ISO/IEC/TC JTC1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

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*

Introduction

This International Standard, also known as “Multimedia Content Description Interface”, provides a standardized set of technologies for describing multimedia content. International Standard addresses a broad spectrum of multimedia applications and requirements by providing a metadata system for describing the features of multimedia content.

The following are specified in this International Standard:

- **Description schemes (DS)** describe entities or relationships pertaining to multimedia content. Description Schemes specify the structure and semantics of their components, which may 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 International Standard is subdivided into eight parts:

Part 1 – Systems: specifies the tools for preparing descriptions for efficient transport and storage, compressing descriptions, and allowing synchronization between content and descriptions.

Part 2 – Description definition language: specifies the language for defining the standard set of description tools (DSs, Ds, and datatypes) and for defining new description tools.

Part 3 – Visual: specifies the description tools pertaining to visual content.

Part 4 – Audio: specifies the description tools pertaining to audio content.

Part 5 – Multimedia description schemes: specifies the generic description tools pertaining to multimedia including audio and visual content.

Part 6 – Reference software: provides a software implementation of the standard.

Part 7 – Conformance testing: specifies the guidelines and procedures for testing conformance of implementations of the standard.

Part 8 – Extraction and use of MPEG-7 descriptions: provides guidelines and examples of the extraction and use of descriptions.

This part of ISO/IEC 15938 contains simulation software for tools defined in parts 1, 2, 3, 4 and 5 of ISO/IEC 15938. This software has been derived from the verification models used in the process of developing the International Standard.

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 bit streams of descriptors and description schemes with normative syntax. The performance of these software tools should not be taken as 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 sections of part 1-5.

Information technology — Multimedia content description interface —

Part 6: Reference software

1 Scope

This International Standard operates on and generates conformant bit streams. This International Standard provides a specific implementation that behaves in a conformant manner. In general, other implementations that conform to ISO/IEC 15938 are possible that do not necessarily use the algorithms or the programming techniques of the reference software.

The software contained in this part of ISO/IEC 15938 is known as experimentation software (XM) and is divided into five categories:

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

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

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

2 Symbols and abbreviated terms

For the purposes of this part of ISO/IEC 15938, the following symbols and abbreviated terms apply:

AV: Audio-visual

CS: Coding Scheme

D: Descriptor

Ds: Descriptors

DCT: Discrete Cosine Transform

DDL: Description Definition Language

DS: Description Scheme

DSs: Description Schemes

ISO: International Organization for Standardization

MDS: Multimedia Description Schemes

MPEG: Moving Picture Experts Group

MPEG-7: Multimedia Content Description Interface Standard (see ISO/IEC 15938)

XML: Extensible Markup Language

3 Copyright disclaimer for software modules

Each source code module in this specification contains 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 System (ISO/IEC 15938-1), MPEG-7 Video (ISO/IEC 15938-3), MPEG-7 Audio (ISO/IEC 15938-4), MPEG-7 Multimedia Description Scheme (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 ©200_".

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

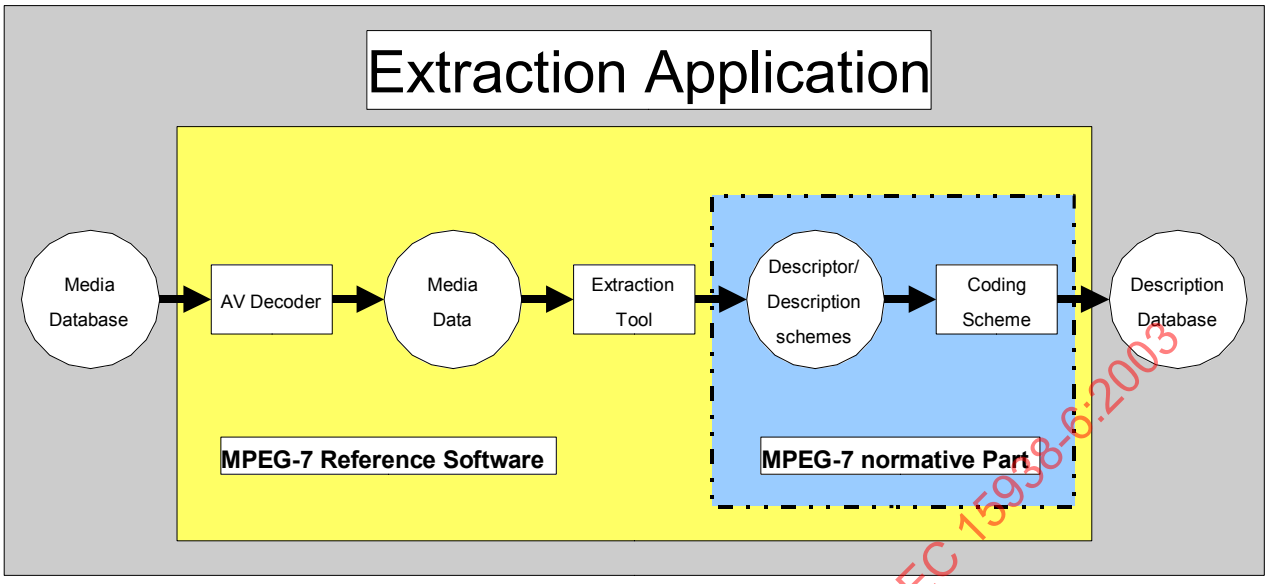
4 XM software architecture

4.1 Block diagrams

In this section you will find some information about the XM software architecture. The block diagrams give short overviews, and introduce individual components of the XM software. The section also provides a list of the directory location for each module.

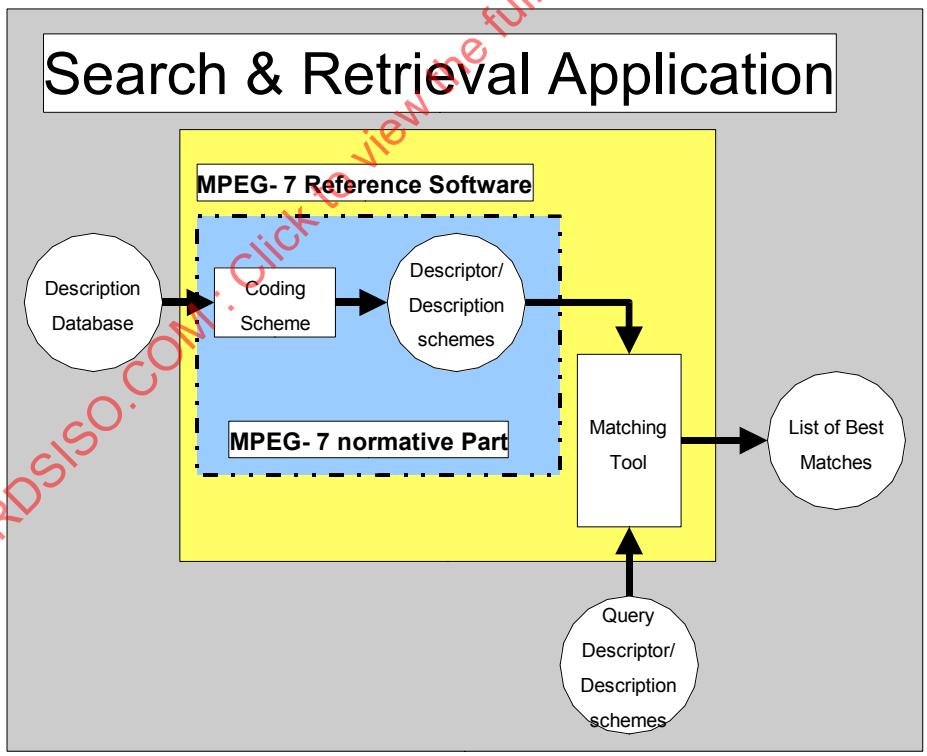
The composing elements of the MPEG-7 Reference Software are characterized by their functionality and by their interfaces. They can be configured according to what here is referred as "Key Applications". We can distinguish from the functional point of view:

- "Extraction Applications" (a description data base is built from a media data base)
- "Search and Retrieval Applications" (a description is compared with the descriptions in a database to find the one with the lowest distance)
- "Transcoding Applications" (a media data base is converted into another media data base basing on its description)



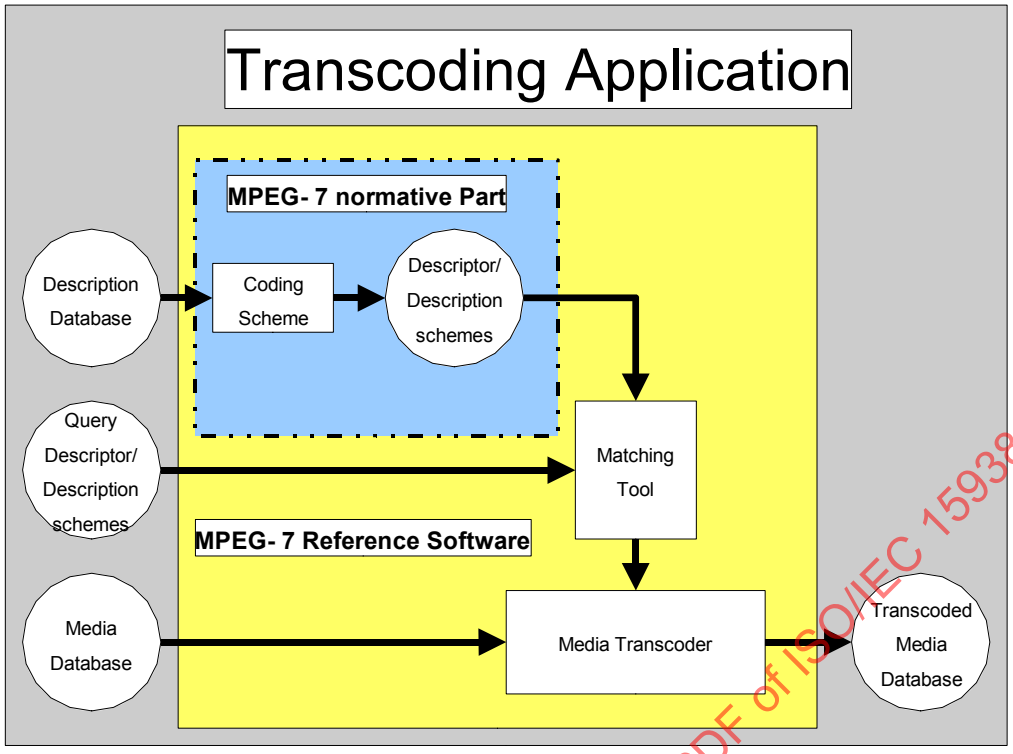
NOTE In the block diagram, boxes represent procedural parts, circles represent data structures.

Figure 1 — Schematic diagram of an "Extraction Application" using the XM reference software modules



NOTE In the block diagram, boxes represent procedural parts, circles represent data structures.

Figure 2 — Schematic diagram of a "Search and Retrieval Application" using the XM reference software modules



NOTE In the block diagram, boxes represent procedural parts, circles represent data structures.

Figure 3 — Schematic diagram of a "Transcoding Application" using the XM reference software modules

In the following, the blocks of the "Key Applications" are distinguished. For elements, that are related to a specific descriptors or description schemes, the interface is given with the example of the DummyType (the XM integration template and not a normative descriptor).

4.1.1 Media database

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

4.1.2 AV decoders

The XM supports the following AV decoders:

- Still image decoders: ImageMagick (Ver.4.*-5.* linked as external library, not included in the XM reference software distribution)
- MPEG-1, MPEG-2 video decoders: (XM directory: Decoders/MPEG2Dec)
- MPEG-1 video motion vector extractor: (XM directory: Decoders/MPEG2Dec) (It can extract images and motion vectors)
- 3D Objects: (XM directory: Media) (It reads a 3D object for 3D shape descriptors)
- Key Points: (XM directory: Media) (It reads in a list of key points from a file).

4.1.3 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.

4.1.4 Extraction tools

Extraction tools are specific extraction methods defined for each Descriptor and Description Scheme. All these source file are available in the ExtractionUtilities XM directory. Extraction tools are not normative in the implementation but they must provide a valid description. The extraction tools extract the descriptions from media data. Because media data can be very big, 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. The interface of the DummyType extraction tool (implementation template) is given below:

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

  // Also connects the Descriptor (result memnory) 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);

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

  // performs extraction form 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 no be used, only its interface is allowd to
  // 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
//=========================================================================
```

4.1.5 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 classes provide 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;
};
/=============================================================================
```

4.1.6 Coding Schemes (CSs)

Coding Schemes are specific coding and decoding methods defined for individual Descriptors and Description Schemes. All these source file are available in the CodingSchemes directory. If an individual coding schemes is available, it represents a normative part of the standard. Coding schemes are available for the visual descriptors to encode or to decode a description into its binary representation. Coding schemes are not available for Ds and DSs which are implemented using the GenericDS class. In their case, the coding scheme box is implemented using the GenericDSCS which is an interface to the "write to file"– and "read from file"– functions of XML 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*/

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

4.1.7 Search & Matching Tools

Search tools are specific search, or matching methods defined for each Descriptor and Description Scheme. All these source file 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 indexing, 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 tools. Here the whole description is processed in one step. The interface of the search tool for indexing 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 indexing 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 teh 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
  };
  //=========================================================================
```


4.1.8 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.

4.1.9 Applications

Applications are expressed by the classes combining the modules of a Descriptor or a Descriptions Scheme including modules of their sub-Ds and -DSs. The resulting class implements one of the three key applications specified above in this section. 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 & Retrieval and Transcoding) are call Client Applications.

4.1.10 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. Thus, in case of code reuse, the way of destructing the object is fixed. 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 that are 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 color description share the same Color 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.

5 Systems reference software (BiM)

This table shows the Reference Software components for Part 1 of ISO/IEC 15938

Name of the Tool in Part 1	Name of the Tool in the XM software
Bit Stream Encoder/Decoder	SystemTools/BiM
Access Unit Navigation	SystemTools/TextualAccessUnits

5.1 BiM reference software

5.1.1 Introduction

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.

5.1.1.1 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: xerces.jar, xerces-c_1_X.so/.dll (version 1.6 is known to compile successfully), and 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.so/.dll and its sources (SIEMENS) containing the functions necessary to encode a textual path in a binary path and to decode a binary path from the bit stream into a textual path.
- A short documentation

5.1.1.2 Installation and execution

Prerequisites: the Java Sun 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, 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.

5.1.2 Software overview

The reference software functionality can be grouped as follows:

- 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.

- 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 binarisation is driven by the input XML.

In the encoding process, simple XML-Schema types (xsd:string, xsd:enumeration) have build-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 structure of the document is encoded as in the Mpeg7-system FDIS.

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

- 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 informations produced in the schema parsing phase.

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

5.1.2.1 Main packages

As explained before, 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.

*As stated before, the **com.expway.ref** package contain the entry points for the command line tools.*

5.2 Access unit navigator

5.2.1 Introduction

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

It is located in the SystemUtilities/AccessUnitNavigator directory of the Reference Software source tree. For installing the access unit navigation you need to have the Xerces XML parser (version 1.6 is know to work fine) to be installed. (xml.apache.org, see http://xml.apache.org/xerces-c/install.html). The xerces1_X.so/.dll file location should be specified while compiling the Encoder/Decoder Modules in their respective Project Workspaces.

5.2.2 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 AUN_input.par file is provided in the Parfiles directory of the reference Software.

5.2.3 Textual Acces 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.

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 is for eg. /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 SystemUtilities/AccessUnit/Navigator/Decoder folder.

6 Systems reference software (DDL)

These Systems components are implemented using the external XML parser library.

Name of the tool in Part 2	Name of the Tool in the XM software
DDL Parser	External XML parser supporting the DOM API (e.g., “xerces” XML parser from Apache)
DDL Validation parser	External XML parser supporting the DOM API (e.g., “xerces” XML parser from Apache)

7 Video reference software

This section lists the reference software components of Part 3 of ISO/IEC 15938. Most of the components of this section include a server- and a client application. The normative descriptors are implemented using a C++ class. All modules have a binary coding scheme, and an interface to the XML parser based implemented description schemes of part 5. Thus the descriptions may be stored in a binary bit stream file or in XML file. The detailed usage instructions for these modules are located in the Doc/Video directory of the Reference Software source tree.

Name of the Tool in Part 3	Clause in Part 3	Name of the Tool in the XM software
Grid layout	5.2	GridLayout
Time series	5.3	TimeSeries
Multiple view	5.4	MultiView
Spatial 2D coordinates	5.5	Spatial2Dcoordinates
Temporal interpolation	5.6	TemporalInterpolation
Color space	6.2	ColorSpace
Color quantization	6.3	ColorQuant
Dominant color	6.4	DominantColor
Scalable color	6.5	ScalableColor
Color layout	6.6	ColorLayout
Color structure	6.7	ColorStructure
GoF/GoP Color	6.8	GoFGoPColor
Homogeneous texture	7.2	HomoTexture
Texture browsing	7.3	TextureBrowsing
Edge histogram	7.4	EdgeHist
Region shape	8.2	RegionShape
Contour shape	8.3	ContourShape
Shape 3D	8.4	3DShapeSpectrum
Camera motion	9.2	CameraMotion
Motion trajectory	9.3	MotionTrajectory
Parametric motion	9.4	ParametricObjectMotion
Motion activity	9.5	MotionActivity
Region locator	10.2	RegionLocator
Spatio-temporal locator	10.3	SpatioTemporalLocator
Face recognition	11.2	FaceRecognition

8 Audio reference software

This section lists the reference software components of Part 4 of ISO/IEC 15938. Not all types are implemented explicitly in the XM. Rather, the functionality of several types may exist in one (set of) file(s). The extraction tools, if existing for the modules, may be implemented using MatLab or C++. The search tools, if existing, are implemented using C++ code. This modules do not have a binary coding scheme besides the BiM format thus usually no normative coding schemes exist. Thus, the descriptions are stored in XML files.The detailed usage instructions for these modules are located in the Doc/Audio directory of the Reference Software source tree.

Name of the tool in Part 4	Clause in Part 4	Name of the tool in the XM software
ScalableSeriesType	5.2	MatLabat different locations in: AudioMatLab
SeriesOfScalarType		
SeriesOfScalarBinaryType		
SeriesOfVectorType		
SeriesOfVectorBinaryType		
AudioLLDScalarType	5.3.2	Virtual type – not implemented
AudioLLDVectorType	5.3.3	
AudioWaveformType	5.3.4	MatLab: AudioMatLab/AudioWaveformD
AudioPowerType	5.3.5	AudioMatLab/AudioPowerD
AudioSpectrumEnvelopeType	5.3.7	AudioMatLab/AudioSpectrumEnvelopeD
AudioSpectrumCentroidType	5.3.8	AudioMatLab/AudioSpectrumCentroidD
AudioSpectrumSpreadType	5.3.9	AudioMatLab/AudioSpectrumSpreadD
AudioSpectrumFlatnessType	5.3.10	C++-code: AudioSpectralFlatness MatLab: AudioMatLab/AudioSpectrumFlatness/*.m
AudioSpectrumBasisType	5.3.11	MatLab: AudioMatLab/AudioBasisD
AudioSpectrumProjectionType	5.3.12	AudioMatLab/AudioProjectionD
AudioHarmonicityType	5.3.13	MatLab: AudioMatLab/AudioHarmonicityD
LogAttackTimeType	5.3.15	MatLab: AudioMatLab/LogAttackTimeD
HarmonicSpectralCentroidType	5.3.16	AudioMatLab/HarmonicSpectralCentroidD
HarmonicSpectralDeviationType	5.3.17	AudioMatLab/HarmonicSpectralDeviationD
HarmonicSpectralSpreadType	5.3.18	AudioMatLab/HarmonicSpectralSpreadD
HarmonicSpectralVariationType	5.3.19	AudioMatLab/HarmonicSpectralVariationD
SpectralCentroidType	5.3.20	AudioMatLab/SpectralCentroidD
TemporalCentroidType	5.3.21	AudioMatLab/TemporalCentroidD File names correpond to D's acronym, e.g. InstantaneousHarmonicSpectralCentroid=IHSC, a function file starts with an F, hence Hisc.m
SilenceHeaderType	5.4	C-code: Silence
SilenceType		

AudioSignatureType	6.2.2	C++-code: AudioSignature*.cpp
InstrumentTimbreType	6.3	C++-code: HarmonicInstrumentTimbre PercussiveInstrumentTimbre
HarmonicInstrumentTimbreType		
PercussiveInstrumentTimbreType		
SoundModelType	6.4	MatLab: AudioMatLab/SoundRecognitionModelD
SoundClassificationModelType		
SoundModelStatePathType		
SoundModelStateHistogramType		
SpokenContentHeaderType	6.5	C++-code: SpokenContent
SpeakerInfoType		
SpokenContentIndexEntryType		
ConfusionCountType		
WordType		
PhoneType		
WordLexiconIndexType		
PhoneLexiconIndexType		
LexiconType		
WordLexiconType		
PhoneticAlphabetType		
PhoneLexiconType		
SpokenContentLatticeType		
SpokenContentLinkType		
MelodyType	6.6	C++-code: Melody
MeterType		
ScaleType		
KeyType		
MelodyContourType		
ContourType		
BeatType		

9 Multimedia description scheme reference software

This section lists the reference software components of Part 5 of ISO/IEC 15938. The components of this section may include a server- or client application, or both. The normative description schemes are implemented based on an XML parser. Therefore, the coding schemes do not exist. However, the writing of the in-memory representation of the descriptions to files and vice versa is implemented using the GenericDSCS module. The description files are always XML files. The detailed usage instructions for these modules are located in the Doc/MDS directory of the Reference Software source tree.