

MPEG-7 Library

MPEG-7 C++ API Implementation

DATE Oct. 5, 2005
ABSTRACT Description of the MPEG-7 library
AUTHOR Hermann Fürntratt, Helmut Neuschmied, Werner Bailer
KEYWORDS MPEG-7 API, XML, Visual, Audio, Multimedia Description Schemes, C++
RELATED ITEMS

DOCUMENT HISTORY

Release	Date	Reason of change	Status	Distribution
0.1	2002-12-06	Document setup	Living	Confidential
0.8	2003-07-14	Mp7Jrs v0.9	Living	Confidential
1.0	2003-09-05	Mp7Jrs v1.0	Living	Restricted
1.1	2003-10-30	Mp7Jrs v1.1	Living	Restricted
1.11	2003-11-17	Mp7Jrs v1.11	Living	Restricted
1.12	2003-12-01	Mp7Jrs v1.12	Living	Public
1.13	2004-04-01	Mp7Jrs v1.13d	Living	Public
1.14	2004-05-11	Mp7Jrs v1.14	Living	Public
1.14b	2004-07-26	Mp7Jrs v1.14b	Living	Public
1.14f	2005-03-22	Mp7Jrs v1.14f	Living	Public
1.2	2005-10-05	Mp7Jrs v1.2	Living	Public

Table of Contents

1	Introduction	4
1.1	Objectives	4
1.2	Benefits	4
1.3	Technical specification.....	4
1.4	About this Document.....	4
2	The Concept of the MPEG-7 library.....	6
2.1	Design	6
2.1.1	Overview	6
2.1.2	Representation of XML Types	6
2.1.3	Factory Pattern	8
2.1.4	Handling of “Conformance under Extension”	8
2.1.5	Easy Extensibility at Runtime	8
2.1.6	Access to Patterns.....	8
2.2	Implementation Features	8
2.2.1	Basic Data Types.....	9
2.2.2	Serializing and Parsing.....	9
2.2.3	Smart Pointers	10
2.2.4	Using MPEG-7 library in Multithreaded Applications	10
2.2.5	XPath support.....	10
3	Using MPEG-7 library	12
3.1	The MPEG-7 library Distribution	12
3.2	Development Environment	12
3.3	Setting up a Project Using MPEG-7 library	12
3.3.1	Setting up a Project Using MPEG-7 library	12
3.3.2	Include Everything	13
3.4	MPEG-7 library sample code.....	13
3.4.1	Simple	13
3.4.2	Creating Nodes.....	14
3.4.3	Node Types and Abstract Nodes Types.....	14
3.4.4	Working with Strings	15
3.4.5	Working with Collections	15
3.4.6	Modifying a MPEG-7 Representation	16
3.4.7	CreateDocument.....	17
4	Extending the MPEG-7 library.....	18
4.1	Extension Concept.....	18
4.2	Creating Extensions.....	18
4.3	Using Extensions	18
5	FAQ.....	19
6	Glossary	21

7 **References** **22**

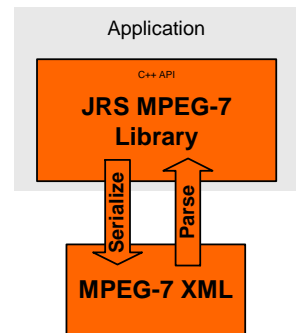
8 **Appendix A: Extensions** **23**

9 **Appendix B: Revision History** **25**

1 Introduction

1.1 Objectives

The MPEG-7 library is a set of C++ classes, implementing the MPEG-7 standard (ISO/IEC 15938:2001). With this library application developers are able to create multimedia content descriptions, manipulate it, serialize it to XML and de-serialize it – with validation – from XML. Target operating systems are Windows and UNIX systems. One major design goal was to simplify extending single classes to allow the developer to enrich interface functionality for certain descriptors. Furthermore documentation on concept and source code level improves the learning curve for the programmer.



1.2 Benefits

All application developers who want to deal with MPEG-7 metadata, and who do not want to struggle with complex XML DOM programming. MPEG-7 metadata, which are stored in a hierarchical XML representation, are made available to a developer in an object oriented hierarchical class tree. The MPEG-7 library avoids the time consuming task of implementing hundreds of necessary MPEG-7 classes before implementation of MPEG-7 functionality can start.

1.3 Technical specification

- Support for Part 3 (Visual), 4 (Audio) and 5 (Media Description Schemes) of the MPEG-7 (version 1, 2001) standard (about 1200 classes)
- Serializes the whole tree or parts of it
- Serializes to string, console output or file
- Schema validation
- Patterns (e.g. MediaTimePoint and Duration) are extracted to numbers – no inefficient and time consuming string processing necessary
- Works with UNICODE and standard character encodings (ISO Latin-1, UTF-8, ...)
- Extension mechanism is very easy to use
- Class functionality can be added at runtime (late binding)
- Creation and destruction of the classes is done by factories
- Handles 'Conformance under extension' described in MPEG-7 standard (Part 7)
- Platform independent
- Future proof – automatic code generation allows fast library update when future MPEG-7 versions are released
- Compiles together with Microsoft components using MS XML

1.4 About this Document

This document is not an API documentation, but shall give an overview about the concepts and of the library. Chapter 2 describes the concepts behind MPEG-7 library in detail. An API documentation comes with the library in HTML format. Together with Chapter 4, which provides information on how to use this library and presents some examples, this should be sufficient information to work with the library.

Chapter 3 discusses how to extend the library and may be skipped when first reading this document and Appendix A lists some inline extension which come with the library and may be useful when working with some of the classes.

A FAQ is provided in this document. For up-to-date information and a list of known issues please consult the MPEG-7 library pages at <http://iis.joanneum.at/MPEG-7>.

2 The Concept of the MPEG-7 library

The MPEG-7 library is a set of classes for creating, manipulating, and serializing information according to the MPEG-7 standard.

The MPEG-7 standard contains more than 800 global types, called descriptors (D) and description schemes (DS). Working with these types in an application requires either working with such generic objects like DOM nodes or implementing classes for all types.

2.1 Design

2.1.1 Overview

The MPEG-7 library represents the MPEG-7 XML document as a tree structure. Each element or attribute, which has an XML type defined in the MPEG-7 schema, is represented by a node in this tree.

2.1.2 Representation of XML Types

2.1.2.1 Mapping between MPEG-7 XML Types and Classes

Basically, each XML type is mapped to a class. This applies both for global types (those defined in the schema as a named simple or complex type) and local types (elements of other types which do not have a named type but are defined where they are used). There is one exception for this mapping rule: XML enumerations are directly mapped to C++ enumerations to reduce additional overhead.

Sequences and choices with have a maximum occurrence attribute larger than 1 and lists are mapped to collection classes.

2.1.2.2 Generic Node Type

There is one generic node type, which defines the common interface of all node classes. All generated classes (i.e. all classes that represent XML types are derived from the generic node type).

Each node provides member functions to access its type name (i.e. the name of the MPEG-7 type it represents) and the class type using runtime type information (RTTI). This is especially useful when extending the MPEG-7 library (cf. Chapter 4).

2.1.2.3 Naming Conventions

The following naming conventions are used:

- Upper case / lower case writing of names will be used as defined in the schema.
- Global types will have the same name as defined in the schema, with the prefix `Mp7Jrs`.
- Local type names will have the following form:
 - prefix `Mp7Jrs`
 - The name of the global type where they are defined.

- The name of the element which defines the local type. If there are nested definitions of local types, this will be more than one element name. The element name(s) will be separated from the global type name (and among one another) by underscores.
 - The suffix `_LocalType`.
 - If there is more than one local type declaration found within the same named element, numbers starting at 0 will be appended to the local types (starting with the second, the first one will never be numbered).
 - If the elements of a collections are unnamed types, they are defined as local types. They will have the same name as the collection, just with `Collection` replaced by `Local`.
- Collection type names will have the following form:
 - prefix `Mp7Jrs`
 - The name of the global type where they are defined.
 - The name of the element which defines the local type. If there are nested definitions of local types, this will be more than one element name. The element name(s) will be separated from the global type name (and among one another) by underscores.
 - The suffix `_CollectionType`.
 - If there is more than one collection found within the same named element, numbers starting at 0 will be appended to the collection types (starting with the second, the first one will never be numbered).
- Getter and setter functions will have the name `Get` (or respectively `Set`), followed by the name of the element. If there are different local types within the element, the functions will be named `Get / Set`, followed by the name of the local type. Collections will can also be accessed via the `Get / Set` function using the corresponding local type name.

Naming Example: VideoSegmentType (Part 5, MDS)

The type itself will be mapped to a class `Mp7JrsVideoSegmentType`. The type contains three choices and two other elements.

One of the elements is `MultipleView`, which is mapped to member variable of type `Mp7JrsMultipleViewType`. The other is an unbounded set of mosaic elements. It will be mapped to a collection called `Mp7JrsVideoSegment_Mosaic_CollectionType`, with elements of `Mp7JrsMosaicType`.

The first of the choices may only occur once, it will therefore be directly included into the class using two member variables, one of type `Mp7JrsMediaTime` and the other of type `Mp7JrsTemporalMask`. The set methods will make sure, that only one the two may be set.

The two other choices may occur for an unbounded number of times. It is therefore necessary to map the content of the choices to local types, which will be called `Mp7JrsVideoSegmentType_LocalType` and `Mp7JrsVideoSegmentType_LocalType0` respectively. For both there will exist collections (`Mp7JrsVideoSegmentType_CollectionType` and `Mp7JrsVideoSegmentType_CollectionType0`), which will have elements of the corresponding local type.

2.1.2.4 Inheritance of XML Types

The inheritance hierarchy defined in the MPEG-7 XML schema (extension, restriction) mapped to an inheritance hierarchy of interfaces. The interfaces have the prefix `I` and the same name as the XML type. Each class implements the corresponding interface but is only

derived from `Mp7JrsNode`, as otherwise extensibility of the library (cf. Chapter 4) would be limited. Instead, each class contains a reference to its base class, which can be accessed using the `GetBase()` member function of each node. The member functions of the base class are copied and will call the base class' implementation of the function.

2.1.3 Factory Pattern

The factory pattern has been heavily used in the MPEG-7 library. For each class which is generated, a factory class is generated, which handles creating and destroying instances of the class.

There is a certain factory type (`Mp7JrsFactory`) which defines the interface of factory classes and serves as a registry for all other factories. A new instance can be created by calling a method of this type registry with the type name of the desired type as argument. Instances will also be deleted by calling a method of the factory.

This strategy is very advantageous when working with DLLs, as it guarantees, that the instances are always destroyed on the heap of the right DLL.

2.1.4 Handling of "Conformance under Extension"

MPEG-7 part 7 defines that MPEG-7 documents containing proprietary types which are extensions of MPEG-7 types do conform the standard. MPEG-7 library supports this concept by wrapping all unknown types it encounters during parsing into a special type of node. It can be handled like any other node in the tree and will contain the unknown subtree, which can be serialized again without modification.

2.1.5 Easy Extensibility at Runtime

The MPEG-7 library can be easily extended by replacing the classes of the core library with other classes providing additional (application specific) functionality. This can be done at runtime (late binding).

Extending the MPEG-7 library is described in Chapter 4.

2.1.6 Access to Patterns

MPEG-7 uses a number of patterns to represent structured data as strings. For commonly used types, such as media time points and durations, MPEG-7 library provides functionality to access fields using the appropriate (numeric) data type. This saves a lot of tedious string processing in the application.

2.2 Implementation Features

Currently only a C++ implementation of the MPEG-7 library is available, but the API has been designed with easy portability to C# and Java in mind. Therefore C++ specific concepts, such as multiple inheritance and templates have not been used.

2.2.1 Basic Data Types

The following mapping to C++ data types is used for the basic data types in the MPEG-7 schema:

W3C Type used in schema	C++ Type
integer	int
positiveInteger	unsigned
nonNegativeInteger	unsigned
float	float
double	double
decimal	double
boolean	bool
string	XMLCh *
token	XMLCh *
NMTOKEN	XMLCh *
anyURI	XMLCh *
ID	XMLCh *
IDREF	XMLCh *
language	XMLCh *
lang	XMLCh *
QName	XMLCh *
NCName	XMLCh *
hexBinary	XMLCh *
base64Binary	XMLCh *

XMLCh* is a pointer to a 16 bit buffer and represents a Unicode string. Internally, the MPEG-7 library uses only Unicode.

Other types are currently not handled.

2.2.2 Serializing and Parsing

The serialization and parsing capabilities of the MPEG-7 library are based on Xerces [Xerces]. This makes MPEG-7 library platform independent as Xerces is supported on many platforms. The MPEG-7 library uses Xerces macros to deal with compiler/platform specific details. Measures have been taken to avoid conflicts with Microsoft components (such as DirectShow) which use MS XML, which potentially conflicts with Xerces.

(De-)Serialization can be done from/to file, buffer and to console. Both whole MPEG-7 XML documents as well as fragments can be (de-)serialized. Both MPEG-7 library and Xerces use Unicode internally. When (de-)serializing, the following encodings may be used:

- UTF-8
- ISO-8859-1 (Latin-1)
- UTF-16

(De-)Serializing is done via a class called `Mp7JrsArchive`. This class also handles settings like encoding, local schema locations, etc. All public methods can be used with both 8 and 16 bit strings.

When parsing from a buffer or file, Xerces' validation capabilities will be used. This includes everything except for checking the number of occurrences of elements.

NOTE: When using Xerces validation, the default `xml-1998.xsd`, that comes with the MPEG-7 schema files has to be replaced. It maps the namespace (where the attributes `lang` and `space` are defined) <http://www.w3.org/XML/1998/namespace> to the prefix `x`, although W3C defines a fixed mapping between this namespace and the prefix `xml`. Also the document which can be found under the namespace URL follows this definition and will work with Xerces validation.

2.2.3 Smart Pointers

MPEG-7 library uses pointers to nodes as type of parameters and return values. This avoids copying if it is not necessary. For each type in MPEG-7 library, there is an associated pointer type, which has the suffix `Ptr` in its name (this suffix replaces the suffix `Type`, if present, otherwise it is just appended).

`Mp7JrsNodePtr` is the basic type and all pointers can be cast to this type. It is used as parameter or return value, whenever this flexibility is required. It also provides cast operators to cast to any specific type using dynamic casts to ensure type safety. If casts fail or it is tried to dereference `NULL` pointers an exception of `Mp7JrsPtrException` is thrown.

To set a smart pointer to `NULL`, either assign it the value `(void*)0` or a newly constructed smart pointer of the same type (`NULL` is the default value for the pointer after construction). For example, to set the `Mp7JrsVideoSegmentPtr` `videoSegment` to `NULL`, use

```
videoSegment = Mp7JrsVideoSegmentPtr();
```

2.2.4 Using MPEG-7 library in Multithreaded Applications

MPEG-7 library provides a locking mechanism to coordinate write access to an MPEG-7 representation. After registering, a client will receive an identifier, which can be used to request locks on nodes. When a lock is granted, the client may perform modifications (it must pass its identifier with every set function). After doing the modifications, the client must unlock the node.

There are two different types of locks: Normal locks allow a client to do any operation, but they block the whole subtree below the locked node. Locks4Add allow the client only to add or insert elements and other operations may be performed in the subtree below the locked node.

Each client using the locking mechanism must implement an interface to receive notifications from any of the nodes. This will be used to inform a client if a node cannot be granted immediately but later. The same mechanism can be used, if a client needs to know, when a certain node has been updated.

Locking and notifications can be switched on and off globally by calling the appropriate methods of `Mp7JrsClientManager`. By default, both are turned off.

NOTE: The locking and notification mechanism is not fully implemented in the current version.

2.2.5 XPath support

The MPEG-7 library supports absolute XPath statements, i.e. XPath expressions containing a full and unambiguous path to a node in the MPEG-7 representation. Nodes provide a

method to get their absolute XPath and vice versa it is possible to get a reference to a node by querying the root node with an absolute XPath expression.

3 Using MPEG-7 library

3.1 The MPEG-7 library Distribution

The distribution of the MPEG-7 library is organized as follows:

Subdirectory	Contents
/bin	DLLs
/doc	This document and a link to the index page of the HTML API documentation.
/doc/html	Browsable API documentation.
/include	Headers of manually coded files
/include/generated	Headers of automatically generated classes
/include/extensions	Headers of inline extensions
/lib	LIB files
/samples	Sample projects using MPEG-7 library.

3.2 Development Environment

To use the MPEG-7 library, the following development environment is required:

- The MPEG-7 library requires Xerces 2.2.0 [Xerces]. The default settings of the sample project assume that the Xerces root directory is on the same level as the MPEG-7 library root directory.
- MS Visual C++ 6.0 (Service Pack 5 recommended) or MS Visual C++ .NET 2003 (7.1).

3.3 Setting up a Project Using MPEG-7 library

3.3.1 Setting up a Project Using MPEG-7 library

The following steps are necessary to use the MPEG-7 library in your project:

- Add the following directories (relative to the MPEG-7 library root) to the include path:
 - /include
 - /include/extensions
 - /include/generated
- Add the path to the Xerces include directory to your include path.
- Add the following directories to your library input path:
 - <MPEG-7 library root>/lib
 - <Xerces root>/lib
- Link against the following libraries:
 - xercesc-2.lib (or xercesc-2D.lib respectively)

- mp7Jrs.lib (or mp7JrsD.lib, or the libraries named _VC6 respectively)

IMPORTANT: You have to enable runtime type information (RTTI) in all projects using the MPEG-7 library. You should use the multithreaded DLL runtime libraries.

For a sample project setup, see any of the sample programs. They are ready to compile and run.

3.3.2 Include Everything

There is a special include file called `Mp7JrsEverything.h` which includes everything defined in the MPEG-7 library. This may be convenient if your project works with a large number of different classes. However, use this include file with care, as it significantly increases compilation time.

3.4 MPEG-7 library sample code

3.4.1 Simple

This simple sample program shows how to parse and serialize an MPEG-7 document from/to a file. For this sample, the source code and the project is available in the samples directory.

```
// Xerces utilities
#include <xercesc/util/PlatformUtils.hpp>
// archive class used for (de-) serializing
#include "Mp7JrsArchive.h"

// factory registry and defines
#include "Mp7JrsFactory.h"
#include "Mp7JrsFactoryDefines.h"
// include the type which is the target type of the cast
#include "Mp7JrsMpeg7_LocalType.h"

int main(int argc, char** argv) {

    // initializes Xerces
    XMLPlatformUtils::Initialize();
    // initialize JRS MPEG-7
    Mp7JrsFactory::Initialize();

    // the following code is put into a separate block to make
    // sure that all smart pointers are deleted, before the factories
    // are deleted by calling Mp7JrsFactory::Terminate
    {
        // create archive instance
        Mp7JrsArchive archive;
        // set encoding and validation mode
        archive.SetEncoding("UTF-8");
        archive.SetDoValidation(true);

        // load and validate MPEG-7 XML file
        // returns the root node of the document
        Mp7JrsNodePtr root = archive.FromFile("test.xml");

        // cast the root element of the document to the appropriate type
```

```

    Mp7JrsMpeg7_LocalPtr mpeg7 = root;

    // serialize the root node again
    archive.ToFile("test2.xml", (Mp7JrsNodePtr)mpeg7);
}

// cleanup JRS MPEG-7
Mp7JrsFactory::Terminate();
// cleanup Xerces
XMLPlatformUtils::Terminate();

return 0;
}

```

The node you get back from a deserialization method is the root node of the MPEG-7 document of fragment you parse.

The type of the root node of an MPEG-7 document is `Mp7JrsMpeg7_LocalType`. It is a local type because the root element does not use any type defined in the schema but locally defines a complex type derived from `Mpeg7Type`.

NOTE: It is important to make sure that all objects are deleted before `Mp7JrsFactory::Terminate` is called. Otherwise the objects referenced by the smart pointers cannot be deleted. This means that you should make sure that all smart pointers are out of scope or are set to NULL before `Mp7JrsFactory::Terminate` is called.

3.4.2 Creating Nodes

Basically, nodes are created and deleted via factories. Assume we want to create a node of type `VideoSegmentType`. The code would be

```

Mp7JrsVideoSegmentPtr videoSegment =
    Mp7JrsFactory::CreateObject(
        Mp7JrsFactory::GetTypeIndex(X("VideoSegmentType")) );

```

This calls a static method of the factory registry, which directs the call to the factory responsible for the type specified as parameter¹. The returned node will be of type `Mp7JrsNodePtr` and must therefore be cast to the correct type before assigning.

Fortunately, these calls are similar for all types and therefore macros are generated, which handle this for you. All you have to do is to include `Mp7JrsFactoryDefines.h`. The call will then look as follows:

```

Mp7JrsVideoSegmentPtr videoSegment = CreateVideoSegmentType;

```

Objects are deleted when they are no longer referenced by any smart pointer.

3.4.3 Node Types and Abstract Nodes Types

In many cases abstract types are used in the schema. In these cases, `Mp7JrsNodePtr` will be used as the type of parameters and return values. The smart pointers support conversion from specialized types to `Mp7JrsNodePtr`s and vice versa, which is done implicitly during assignment.

The MPEG-7 library smart pointers support the concept of NULL pointers to indicate elements or attributes which are not set. Elements can be deleted by setting them to NULL.

¹ The argument `X("VideoSegmentType")` is just the name of the class as string, `X(...)` is a utility macro that converts between `char*` and Xerces `XMLCh*`.

3.4.4 Working with Strings

Internally, MPEG-7 library uses the Xerces `XMLCh*` type for representing strings. Strings set to MPEG-7 classes will be adopted, i.e. if you need the string separately or pass a constant string, you have to copy the string before passing it as argument.

For working with `XMLCh*`, the utility class `XMLString` can be used (cf. Xerces documentation). For example, strings can be copied using `XMLString::replicate` and converted between char and wide char using `XMLString::transcode`.

3.4.5 Working with Collections

Collections are used to represent list constructs (space separated simple types as content of an element) and any type of constructs with `maxOccurs` attributes larger than 1. The content of a collection can be any type defined in MPEG-7 library.

Collections are usually normal member variables of the element containing them. The collection class provide methods for adding, inserting, removing, accessing, and counting elements etc. The type of the elements of a collection depends on the collection, it can be a named type from the MPEG-7 standard or, in most cases, a local type with the same name as the collection (only with other suffix). This is the case if the content of the collection is implicitly defined, e.g. by defining the content of a collection to be a choice.

The following example shows how to iterate through the visual descriptors of a `VideoSegment` (which are inherited from `SegmentType`).

```
#include "Mp7JrsVideoSegmentType.h"
#include "Mp7JrsVideoSegmentType_CollectionType.h"
#include "Mp7JrsEnumerator.h"
#include "MP7JrsVideoSegmentType_LocalType.h"

void collection(Mp7JrsVideoSegmentPtr videoSegment) {
    // get the collection
    Mp7JrsVideoSegmentType_CollectionPtr visualDColl=
        videoSegment->GetVideoSegmentType_LocalType();
    // get the enumerator
    Mp7JrsNodeEnum enumVis = visualDColl->GetElements();
    // iterate through the elements
    while (enumVis.HasNext()) {
        Mp7JrsVideoSegmentType_LocalPtr vsLocal = (Mp7JrsVideoSegmentType_LocalPtr)
            enumVis.GetNext();

        // do something
    }
}
```

The enumerator contains a snapshot of the collection's elements.

It is also possible to iterate only through the elements of a certain type. In the following example, the enumerator contains all descriptions of an MPEG-7 document (i.e. those contained in a collection of the root node) that have the type `ContentEntityType`:

```
// load the root node
Mp7JrsMpeg7_LocalPtr root = _archive.FromFile("mp7.xml");
// get the collection
Mp7JrsMpeg7_Description_CollectionPtr descrColl = root->GetDescription();
// get the enumerator containing elements of the desired type
Mp7JrsNodeEnum contEntityTypeEnum = descrColl ->GetElementsOfType
    (Mp7JrsFactory::GetTypeIndex(X("ContentEntityType")));
// iterate through the elements
while (contEntityTypeEnum.HasNext()) {
```

```

        Mp7JrsContentEntityPtr ce =(Mp7JrsContentEntityPtr) contEntityEnum.GetNext();
        // ...
    }

```

The elements of some collections are not named types, but locally defined choices or collections. As these constructs are modelled as local types in this library, the method `GetElementsOfType` could only be used to check whether the collection element is the appropriate local type or not. For these kind of collections, a method called `GetElementsContainingType` is provided, which allows introspection into the local type, by checking whether one of the elements in the local type has a certain type:

The following example shows to get all `ColorLayoutType` descriptors from a `VideoSegment`'s collection of visual descriptors:

```

#include "Mp7JrsVideoSegmentType.h"
#include "Mp7JrsVideoSegmentType_CollectionType.h"
#include "Mp7JrsEnumerator.h"
#include "Mp7JrsColorLayoutType.h"

void collection(Mp7JrsVideoSegmentPtr videoSegment) {
    // get the collection
    Mp7JrsVideoSegmentType_CollectionPtr visualDColl=
        videoSegment->GetVideoSegmentType_LocalType();
    Mp7JrsNodeEnum collLayoutEnum = visualDColl->GetElementsContainingType
        (Mp7JrsFactory::GetTypeIndex(X("ColorLayoutType")));
    // iterate through the elements
    while (collLayoutEnum.HasNext()) {
        Mp7JrsColorLayoutPtr cl =(Mp7JrsColorLayoutPtr) collLayoutEnum.GetNext();
        // do something
    }
}

```

3.4.6 Modifying a MPEG-7 Representation

When modifying a representation, the MPEG-7 locking mechanism must be used, as it guarantees thread safety, if multiple components are accessing the representation. The following code show how to delete an element and add one to a collection.

The component that registers for modifying, needs to implement the MPEG-7 library `NotificationListener` interface.

```

void AClass::Notify(Mp7JrsNotificationPtr notification) {
    // .. do something
}

void AClass::Modify(Mp7JrsVideoSegmentPtr videoSegment) {
    // register to get a client ID
    Mp7JrsClientID myID = Mp7JrsClientManager::GetNewID(this);
    // lock the node to be modified
    if (videoSegment->Lock(myID)) {
        // delete media information element by setting an empty pointer
        Mp7JrsMediaInformationPtr mediaInfo;
        videoSegment->SetMediaInformation(mediaInfo);
        // unlock
        videoSegment->Unlock(myID);
    }
    // for the following operation, a lock4add is sufficient. It should be used
    // whenever possible in order not to block the tree
    if (videoSegment->Lock4Add(myID)) {
        // create a CreationInformation/Creation and add a title
    }
}

```



```
Mp7JrsCreationInformationPtr creationInfo = CreateCreationInformationType;
Mp7JrsCreationPtr creation = CreateCreationType;
creationInfo->SetCreation(creation);
Mp7JrsCreationType_Title_CollectionPtr titleColl =
    CreateCreationType_Title_CollectionType;
creation->SetTitle(titleColl);
// create the title and set its content (using Unicode)
Mp7JrsTitlePtr title = CreateTitleType;
title->SetContent(XMLString::replicate(L"My title"));
titleColl->addElement(title);
// unlock
videoSegment->Unlock(myID);
}}
```

3.4.7 CreateDocument

This is a more advanced sample, that parses a document, removes all descriptions and creates them again. The goal of this sample is to demonstrate how to create MPEG-7 tree structures from scratch using the JRS MPEG-7 Library.

The source code and project file for this sample can be found in the samples directory of your installation.

4 Extending the MPEG-7 library

4.1 Extension Concept

As an automatically generated library, MPEG-7 library provides very generic functionality. In many applications, it will be useful to add specific functionality to certain descriptors or description schemes, e.g. to convert between the MPEG-7 representation and specific internal data types.

Because of the extensive use of the factory pattern in MPEG-7 library, each class of the library may be exchanged at runtime. The extension will then replace the original type. Application components that know about the extension can use its additional functionality while other components can use it just like the original type.

4.2 Creating Extensions

To extend an existing class, you have to create two new classes:

- A class derived from the one you want to replace. Here you will implement additionally functionality.
- A new factory, derived from the factory for the original class. It will look just the same like the one you derive from, except that the type of the original class is replaced by your new one.

For convenience of the users, you should also define a smart pointer type and provide a create macro like one of those defined in `Mp7JrsFactoryDefines.h`. It will create an instance of your new type and cast it to the correct type, so that the additional functionality is accessible. These declaration should be at the end of the h file of your new class and should look like the following:

```
#define CreateMyExtendedType(Mp7JrsFactory::CreateObject( \
Mp7JrsFactory::GetTypeIndex(X("IDofOriginalClass"))))

typedef Mp7JrsPtr< MyExtendedType > MyExtendedPtr;
```

4.3 Using Extensions

Using extensions is very easy:

- Register the extension. This is simply done by creating an instance of the factory of the new type. It is recommended to call `SetToDelete(true)` on the new factory, so that `Mp7JrsFactory` will destroy the factory when `Mp7JrsFactory::Terminate` is called.
- From now on, your new type replaces the original one. Applications can work with instances of the new type just like with instances of the original one, if they do not know about the extension.

5 FAQ

For most up-to-date information please look at the MPEG-7 library web pages at <http://iis.joanneum.at/mpeg-7>

Q: Can the library also be used with MS Visual C++ .NET?

A: Yes, the library can be used with both MS VC++ 6.0 and .NET 2003 for unmanaged C++ applications.

Q: Are there any special compiler or linker settings I have to enable when using the MPEG-7 library?

A: In all projects where you use the MPEG-7 library you have to enable the Run Time Type Information (RTTI) in the compiler settings. You should link against multithreaded DLL runtime libraries.

Q: Do I have to use any special libraries in connection with the MPEG-7 library?

A: The MPEG-7 library requires Xerces 2.2.0.

Q: I have problems validating MPEG-7 XML documents, the parser reports that namespace <http://www.w3.org/XML/1998/namespace> must have the prefix xml.

A: The XML specification defines that the namespace <http://www.w3.org/XML/1998/namespace> is by default mapped to the prefix xml. This rule is obeyed by the definitions for this namespace available at the W3C site (both 1998 and 2001 version). xml-1998.xsd contains the definition of this namespace used in the MPEG-7 schemas. There this namespace is internally called x, violating the XML specification. Xerces does not accept this, but requires the W3C definition. Change the prefix in this file back to xml.

Q: When using Win32 API includes, I keep getting compilation errors because of redefinition of DOM classes?

A: A number of Win32 API includes use the MSXML header file, which contains forward declaration (without a namespace) that cause conflicts with Xerces types. Make sure to include the MPEG-7 library and/or Xerces header files before the Win32 API header files.

Q: I'm using the JRS MPEG-7 Library under MS Visual Studio .NET 2003, using the .NET 2003 version of the JRS MPEG-7 library. When I try to run the debug build of the test application, I get the message that the runtime libraries msvcrt.dll and msvcrtd.dll are not found.

A: The problem is caused by the Xerces XML parser. The Xerces 2.2.0 Binary distribution has been compiled using MSVC 6, which requires these runtime libraries. This version of Xerces cannot be compiled with VS .NET 2003, as it uses no longer supported parts of the C API.

Solution: Please copy the required DLLs from an installation of MS VC++ 6.0 to Windows/System32 directory.

Q: The code I have written using version 1.1x of the MPEG-7 Library does not compile with version 1.2. The compiler complains about Mp7JrsClass.

A: In order to make the library more extensible and flexible, the concept of a static enumeration of class IDs has been replaced by a dynamic list of type IDs. The type ID can be queried from the factory class using the type name:

```
Mp7JrsFactory::GetTypeIndex(X("MyType"))
```

This method call replaces the access to the class ID enumeration:

```
Mp7JrsClass::MyType.
```

Q: I've downloaded the library from your website, but when I try to open the setup launcher, I get a message saying "Error reading setup initialization file".

A: The problem with reading the initialization files may have one of the following causes:

- 1) There is no space left in your temp directory. Clean out the Temp directory.
- 2) You may have insufficient permissions on the machine. Make sure you have the correct permissions. For installing the JRS MPEG-7 Library, you should have administrator rights on your machine.
- 3) The said symptom occurs if the user does not have access privileges for the following registry keys:

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Interface
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\TypeLib

6 Glossary

API	Application Programming Interface.
MPEG-7	ISO/IEC standard 15938: Multimedia content description interface. An extensive framework to describe multimedia content.
MSVC	Microsoft Visual C++ Compiler.
MS XML	Microsoft XML Parser.
RTTI	Run time type information.
XML	Extensible Markup Language, http://www.w3c.org/XML/
XML Schema	XML Schema provide a means for defining the structure, content and semantics of XML documents. An XML schema definition (XSD) is itself formulated as a XML document.
XPath	XPath is a language for addressing parts of an XML document. http://www.w3c.org/TR/xpath
Win32	Microsoft Windows 32bit operating systems, i.e. Windows 95/98/Me and Windows NT/2000/XP.

7 References

[Xerces] Xerces C++ <http://xml.apache.org/xerces-c/>

8 Appendix A: Extensions

This chapter describes inline extensions of the MPEG-7 library, i.e. additional methods that are available besides the automatically generated code.

NOTE: These extensions are currently not documented in the HTML API documentation. The documentation will be added in one of the next versions.

Class	Description
Mp7JrsColorLayoutType	Init: DC coefficients, and 6 Y and 3 C AC coefficients are initialized
	Get/Set number of coefficients
	Get/Set coefficients values as arrays
Mp7JrsControlledTermUseType	Get/Set term reference using URN or AnyURI
Mp7JrsEdgeHistogramType	Init: Create BinCount list, consisting of 80 unsigned 3 bit integers.
Mp7JrsHierarchicalSummaryType	Init: Create list of segment groups and set type to independent.
	Add a segment group.
Mp7JrsmediaDurationType	Get/Set total number of fractions. NOTE: The date is currently ignored!!
Mp7JrsmediaTimePointType	Get/Set total number of fractions. NOTE: The date is currently ignored!!
	Calculate duration as difference of two time points.
	Add duration to time point.
Mp7JrsmediaRelTimePointType	Get/set the relative time point as absolute time point with respect to the mediaTimeBase attribute of the in the current scope.
Mp7JrsMediaTimePointType (NOTE: references not supported)	Init: initializes start time and duration
	Calculates end time from start time and duration
	Get/Set begin/end time
	Check if time point is inside
Mp7JrsSummarySegmentGroupType	Init: name and caption collections and fidelity
	Add Segment/SegmentGroup
	Get number of Segments/SegmentGroups
	Add name/caption
	Get/Set fidelity
Mp7JrsSummarySegmentType	Init: Create name and keyframe collections
	Add Keyframe, get number of keyframes
	Set/Get key video clip
	Add name

Mp7JrsVideoSegmentType	Get/Set begin/end time
------------------------	------------------------

9 Appendix B: Revision History

Version 1.14g to 1.2

- **IMPORTANT:** The static enumeration of type IDs in `Mp7JrsClass` has been replaced by a dynamic concept of type IDs. Use `Mp7JrsFactory::GetTypeIndex(X("MyType"))` instead of `Mp7JrsClass::MyType`. See also the FAQ entry on this issue.
- Support for schema extensions.
- Fixed parsing bug in `ClassificationSchemeBaseType`
- Fixed serialization bug of optional attributes
- Fixed serialization problem with boolean values
- Fixed missing initialization for some members
- Fixed problem serializing `ScalableColorType`
- Fixed problem parsing `xml:lang` attribute
- Fixed bug when parsing fragments
- Fixed bug parsing attributes that contain lists of elements
- Improved documentation of inherited member functions
- Fixed serialization bug of boolean values
- Fixed bug when accessing local type elements by XPath

Version 1.14f to 1.14g

- Fixed memory leaks in `Archive` and `SchemaLocationMap`

Version 1.14b to 1.14f

- Fixed `mediaDuration` serialization bugs (missing D and serializing empty elements)
- Workaround for problems parsing some pattern types
- Fixed bug when serializing `StructuredAnnotationType`
- Added workaround for side effect of `DirectShow` header file (`windowsx.h`)
- Fixed collection parsing problem
- Fixed bug with missing flag for some attributes
- Fixed problem parsing from UTF-16 buffers
- Fixed problem parsing fragments

Version 1.14 to 1.14b

- Added sample program `CreateDocument`
- Fixed minor linker problem under MSVC .NET 2003
- Fixed bug in attribute setter
- Fixed problem when cloning subtrees

Version 1.13d to 1.14

- Fixed bugs in XPath implementation
- Fixed bugs in collection serialisation

- Fixed memory problem in VC++ .NET 2003 version
- Fixed problem in string collections
- Fixed problem when setting parents of newly inserted nodes
- Corrected serialization of coordinate lists and media time elements without durations
- Added prefix to file ptrtypes.h
- Fixed wrong setter implementation for choices of sequences
- Fixed signed/unsigned mismatch in enumerator

Version 1.12 to 1.13d

- **NOTE:** There is a known issue in 1.13d concerning the serialization of lists of simple types (each element gets an own named tag). This will be fixed in 1.14.
- Fixed parsing bugs (elements getting lost when parsing, nested tags in collections)
- Fixed bug in fragment serialisation that caused node names to be wrong
- Fixed compatibility problems with VC++ .NET 2003 compiler
- Improved usability of TermReferenceType extension
- Removed dependency from some Xerces pre-processor defines
- Inline extension documentation included in HTML documentation

Version 1.11 to 1.12

- Global switch to turn on/off locking and notifications
- Fix of serialization bug for some collection types
- Fix of parsing of unions of patterns restricted strings
- Fix of bug that caused limitation of number of elements in collections
- Fix of memory de-allocation problem in SchemaLocationMap

Version 1.1 to 1.11

- Smart pointers are used in methods of archive and factory, create macros have been modified accordingly
- Implementation of basic locking and notification functionality
- Fixed access violation in enumerators
- Bugfixes in collection: insertAt method
- Fix of incorrect parsing of collections if xsi:type attribute is present
- Fix of parsing of IntegerMatrixType

Version 1.0 to 1.1

- Implementation of smart pointer and reference counting completed
- Implementation of interface inheritance hierarchy
- Bug in extension of ColorLayoutType fixed (wrong number of coefficients returned)
- Bug in Mp7JrsEnumType fixed (RefVectorOf from the Xerces API is now used)
- Bug fixes in access to collections