INTERNATIONAL STANDARD

ISO/IEC 21000-10

First edition
2006-01-01

AMENDMENT 1
2006-12-15

Information technology — Multimedia framework (MPEG-21) —

Part 10: Digital Item Processing

AMENDMENT 1: Additional C++ bindings

Technologies de l'information — Cadre multimédia (MPEG-21) —

Partie 10: Traitement d'élément numérique

AMENDEMENT 1: Liaisons C++ additionnelles

Reference number
ISO/IEC 21000-10:2006/Amd.1:2006(E)

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

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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

Amendment 1 to ISO/IEC 21000-10:2006 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This Amendment defines normative C++ bindings for Digital Item Base Operations, informative security and platform dependence considerations, an informative example of a safe DIP profile and an entry to be appended to the Bibliography.

Information technology — Multimedia framework (MPEG-21) —

Part 10:
Digital Item Processing

AMENDMENT 1: Additional C++ bindings

In Clause 1, Scope, second paragraph, replace the text:

three normative annexes

with:

four normative annexes

In Clause 1, Scope, add at the end:

— C++ bindings for Digital Item Base Operations:

Annex E specifies the C++ bindings for the Digital Item Base Operations described in 5.4.

In Clause 2, Normative references, insert the following normative reference before the reference to ISO/IEC 16262:2002:

ISO/IEC 14882:2003, *Programming languages — C++*

Add a new subclause 5.6:

5.6 Security and platform dependence considerations (informative)

5.6.1 Security considerations

5.6.1.1 Execute DIBO

The use of the `DIP.execute` DIBO can potentially result in security issues, because the DIBO provides means to execute arbitrary code.

5.6.1.2 C++ bindings

The use of the C++ DIBO bindings can potentially result in security issues.

5.6.1.3 DOM Load and Save

The DOM Load and Save API exposes potential security issues, because they provide access to the file system.

5.6.2 Platform dependence considerations

5.6.2.1 Execute DIBO

The use of the `DIP.execute` DIBO can result in the use of platform dependent code. It is possible to avoid those issues by creating a profile removing `DIP.execute`. This can be done using profiles as demonstrated in subclause Annex I.8.

5.6.2.2 C++ bindings

The use of the C++ DIBO bindings will result in the use of platform dependent code. It is possible to avoid those issues by creating a profile removing the C++ bindings. This can be done using profiles as demonstrated in subclause Annex I.8.

Adjust numbering clauses:

"5.6 Digital Item eXtension Operations" numbering change to "5.7 Digital Item eXtension Operations" numbering.

And change following (sub)clauses numbering accordingly.

Add a new Annex E:

Annex E
(normative)

C++ bindings for Digital Item Base Operations

E.1 Introduction

C++ bindings for DIBOs are specified so that C++ executables can interact with the DIP environment. The way in which C++ executables are executed and the reference to the bindings are obtained, is done in an implementation specific way.

E.2 C++ data type bindings for DIML object types

E.2.1 DIPError

See 5.4.3.2

```
#ifndef DIPERROR_H
#define DIPERROR_H

/**
 * C++ interface for the DIPError.
 */

class DIPError {
  public:
        /**
         * General DIP error not covered by other defined error codes.
         * The value of this property is 1.
         */
        static const int GENERAL_EXCEPTION;

        /**
         * A parameter provided to a DIBO or other DIP function is invalid.
         * The value of this property is 2.
         */
        static const int INVALID_PARAM;

        /**
         * Permission to execute this operation is unavailable in the host environment.
         * The value of this property is 3.
         */
        static const int INVALID_PERMISSION;

        /**
         * Something needed to complete the operation is not found.
         * The value of this property is 4.
         */
        static const int NOT_FOUND;

        /**
         * An error occurred during an attempt to adapt a resource.
         * The value of this property is 5.
         */
        static const int ADAPTATION_FAILED;

        /**
         * An error occurred during an attempt to play.
         * The value of this property is 6.
         */
        static const int PLAYBACK_FAILED;

        /**
         * An error occurred during an attempt to execute.
         * The value of this property is 7.
         */
        static const int EXECUTE_FAILED;

        /**
         * An error occurred during an attempt to print.
         * The value of this property is 8.
         */
        static const int PRINT_FAILED;

        /**
         * Returns the code of an error caused by an exception.
         * @return int value representing DIPErrorCode or other value specified in ISO/IEC
         *          21000.
         */
        virtual int getDIPErrorCode() const = 0;
};

#endif
```

E.2.2 ObjectMap

See 5.4.3.3

```
#ifndef OBJECTMAP_H
#define OBJECTMAP_H

#include <DOMElement.h>

/**
 * C++ interface for the ObjectMap DIML Object Type.
 */
class ObjectMap {
  public:
       /**
        * Returns a pointer to succeeding char pointers representing the
        * Arguments Types of an argument list.
        * @param index The index of the Argument list in the Object Map.
        * @return array of char pointers or null pointers if no such index exists. The
        *          last element of the array must be a null pointer.
        */
       virtual char** getArgumentList(int index) const throw (DIPError) = 0;

       /**
        * Returns the number of unique argument lists with arguments in a specific order.
        * @return int value indicating the number of unique argument lists.
        */
       virtual int getArgumentListCount() const = 0;

       /**
        * Returns the number of DIMs taking arguments of given Argument Types.
        * @param argumentList An array of char pointers representing the Arguments names.
        *                     The last element of this array shall be a null pointer.
        * @return int value indicating the number of DIMs.
        */
       virtual int getMethodCount(char** argumentList) const throw (DIPError) = 0;

       /**
        * Returns a pointer to a DOMElement representing Components containing the DIM
        * declarations of DIMs taking arguments of given Argument Types.
        * @param argumentList An array of char pointers representing the Arguments names.
                             The last element of this array shall be a null pointer.
        * @return pointer to DOMElement.
        */
       virtual DOMElement* getMethodWithArgs(char** argumentList) const
                                                                    throw (DIPError)= 0;

       /**
        * Returns an array of pointers to DOMElements representing Components containing
        * the DIM declaration of a DIM taking arguments of given Argument Types.
        * @param argumentList An array of char pointers representing the Arguments names.
                             The last element of this array shall be a null pointer.
        * @param index An int value indicating the index of the DIM in the list of DIMS
                     that accept the char pointers of the argumentList parameter as
                     parameters.
        * @return array of pointers to DOMElements. The last element of this array shall
        *          be a null pointer.
        */
       virtual DOMElement** getMethodsWithArgs(char** argumentList, int index) const
                                                                    throw (DIPError)= 0;

       /**
        * Returns a DID object corresponding to the given Object Type and the index.
        * @param objectType A char pointer containing the Object Type of the wanted DID
                           object.
        * @param index The index in the array to DID objects corresponding
                       to the Object Type.
```

```
         * @return pointer to DOMElement or null pointer if no such DID object exists.
         */
        virtual DOMElement* getObjectOfType(char* objectType, int index) const
                                                                    throw (DIPError)= 0;

        /**
         * Returns an array of DID objects corresponding to the given Object Type.
         * @param objectType A char pointer containing the Object Type of the wanted DID
                             objects.
         * @return array of pointers to DOMElements. The last element of the array must
         *         be a null pointer.
         */
        virtual DOMElement** getObjectsOfType(char* objectType) const
                                                                    throw (DIPError) = 0;

        /**
         * Returns the number of objects corresponding to a certain Object Type.
         * @param objectTypeName A char pointer containing the name of the Object Type in
                                 the Object Map.
         * @return int value representing the number of objects.
         */
        virtual int getObjectsOfTypeCount(char* objectTypeName) const
                                                                    throw (DIPError) = 0;

        /**
         * Returns the number of Object Types defined in the Object Map.
         * @return int value representing the number of Object Types.
         */
        virtual int getObjectTypeCount() const = 0;

        /**
         * Returns a char pointer representing the Object Type name.
         * @param index The index of the Object Type in the Object Map.
         * @return char pointer or null pointer if no such index exists.
         */
        virtual char* getObjectTypeName(int index) const throw (DIPError) = 0;
};

#endif
```

E.2.3 PlayStatus

See 5.4.3.4

```
#ifndef PLAYSTATUS_H
#define PLAYSTATUS_H

/**
 * C++ interface for the PlayStatus DIML Object Type.
 */
class PlayStatus {
   public:
        /**
         * Indicates that the associated resource is not currently playing.
         * The value of this property is 0.
         */
        static const int RELEASED;

        /**
         * Indicates that the associated resource is currently playing. Time based state
         * information related to playing the resource, if relevant, is paused for a
         * STATICPLAY resource.
```

```
         * The value of this property is 1.
         */
        static const int STATICPLAY;

        /**
         * Indicates that the associated resource is currently playing. Time based state
         * information related to playing the resource, if relevant, is advancing for a
         * TIMEPLAY resource.
         * The value of this property is 2.
         */
        static const int TIMEPLAY;

        /**
         * Returns the current status of a played instance of a resource associated with
         * this PlayStatus object.
         * @return int value representing the current status.
         */
        virtual int getStatus() const = 0;
};

#endif
```

E.3 C++ DIBO factory interface

This subclause specifies the C++ interface for the C++ DIBO factory. An MPEG-21 environment providing C++ bindings to DIBOs shall provide an implementation of `CppDIBOFactory`.

The C++ DIBO factory is used in a C++ executable to obtain an instance of an object that implements the C++ binding for a DIBO. The interface of a C++ DIBO factory is defined below.

```
#ifndef CPPDIBOFACTORY_H
#define CPPDIBOFACTORY_H

#include "DIPError.h"

/**
 * CppDIBOFactory interface is used to create new C++ DIBO classes.
 */
class CppDIBOFactory {
   public:
        /**
         * Returns the implementation for the given DIML object interface.
         * defining the set of C++ DIBO interfaces bound to the required DIBO. This method
         * is implemented by the C++ DIBO implementation provider.
         * @param objectName A char pointer containing the name of the DIML object for
         *                   which the implementation is requested.
         * @return void pointer representing the implementation for the given DIML object
         *          interface.
         */
        virtual void* getCppDIBOObject(char* objectName) const throw (DIPError) = 0;
};

#endif
```

E.4 C++ global environment interface

A reference to the global environment (i.e., `GlobalEnv` object) should be provided to a C++ executable, enabling access to the DIP environment in the C++ executables.

```
#ifndef GLOBALENV_H
#define GLOBALENV_H

#include <DOMDocument.h>
#include "CppDIBOFactory.h"

/**
 * C++ interface defining a mechanism for C++ executables to query the platform for
 * environment settings.
 */

class CppDIBOFactory;

class GlobalEnv {
   public:
         /**
          * Returns the instance of the CppDIBOFactory which in turn is used to
          * instantiate C++ DIBOs. This call must not fail.
          * @return pointer to CppDIBOFactory.
          */
         virtual CppDIBOFactory* getCppDIBOFactory() const = 0;

         /**
          * Returns the instance of the Current DIDL document.
          * @return pointer to DOMDocument.
          */
         virtual DOMDocument* getCurrentDIDDocument() const = 0;
};

#endif
```

The platform implementation for a particular C++ DIBO will be obtained from the `CppDIBOFactory` object (see subclause E.3 for the specification of the `CppDIBOFactory` interface) which shall be queried from the `GlobalEnv` object.

E.5 C++ interface bindings for DIBOs

E.5.1 Introduction

This clause specifies the C++ interface bindings for the corresponding DIBOs as specified in the subclause 5.4.

E.5.2 DIDL document access and manipulation

In DIML base operations for accessing and manipulating the DIDL document objects are those specified by the DOM Level 3 Core API as defined by W3C.

Any W3C DOM conformant C++ language binding supporting the Core module can be used, provided:

- The Core module interfaces of the W3C DOM specification are supported;
- The binding specifies an interface called “DOMDocument.h” which supports the Document interface from the Core module;
- The binding specifies an interface called “DOMElement.h” which supports the Element interface from the Core module.

NOTE For example, the C++ language binding for the DOM Level 3 Core API from Xerces – C++ 2.4 [1] can be used. It provides the C++ bindings for these access and manipulation operations.

E.5.3 DIDL document loading and saving

In DIML base operations for loading and saving a DIDL document are those specified by the DOM Level 3 Load and Save API as defined by W3C.

Any W3C DOM conformant C++ language binding supporting the Load and Save module can be used, provided the Load and Save module interfaces of the W3C DOM specification are supported.

NOTE For example, the C++ language binding for the DOM Level 3 Load and Save API from Xerces – C++ 2.4 [1] can be used. It provides the C++ bindings for these loading and saving operations.

E.5.4 DIA related operations

See 5.4.2.4.

```
#ifndef DIA_H
#define DIA_H

#include <DOMElement.h>
#include "DIPError.h"

/**
 * C++ interface bindings for the DIA related DIBOs.
 */
class DIA {
   public:
        /**
         * Requests the adaptation of the given resource.
         * @param element A pointer to a DOMElement representing the Component or
         *                Descriptor to be adapted.
         * @param metadata An array of pointers to DOMElements representing additional
         *                information or null pointers. The last element of the array
         *                must be a null pointer.
         * @return pointer to DOMElement representing the adapted DIDL element or null
         *          pointer if the element was not adapted.
         */
        virtual DOMElement* adapt(DOMElement* element, DOMElement** metadata)
                                                                        throw (DIPError) = 0;
};

#endif
```

E.5.5 DID related operations

See 5.4.2.5.

```
#ifndef DID_H
#define DID_H

#include <DOMElement.h>
#include "DIPError.h"

/**
 * C++ interface bindings for the DID related DIBOs.
 */
class DID {
   public:
        /**
```

```
         * Returns whether or not the conditions in the given DOMElement are satified.
         * @param element A pointer to a DOMElement representing the DIDL element for wich
         *                conditions will be tested.
         * @return boolean value of true if conditions of DOMElement are satisfied, false
         *          if not.
         */
        virtual bool areConditionsSatisfied(DOMElement* element) throw (DIPError) = 0;

        /**
         * Configures the choice in the given DOMElement.
         * @param choice A pointer to a DOMElement representing the DOMElement to be
         *               configured.
         * @return boolean value of true if choice configuration was modified, false if
         *          choice configuration was not modified.
         */
        virtual bool configureChoice(DOMElement* choice) throw (DIPError) = 0;

        /**
         * Sets the state of the selection of the given DOMElement to the given
         * state.
         * @param selection A pointer to a DOMElement representing the Selection.
         * @param state A char pointer to the state to set the Selection ("true", "false",
         *              or "undecided").
         * @return void.
         */
        virtual void setSelection(DOMElement* selection, char* state)
                                                                    throw (DIPError) = 0;
};

#endif
```

E.5.6 DII related operations

See 5.4.2.6.

```
#ifndef DII_H
#define DII_H

#include <DOMDocument.h>
#include <DOMElement.h>
#include "DIPError.h"

/**
 * C++ interface bindings for the DII related DIBOs.
 */
class DII {
   public:
        /**
         * Retrieves DIDL elements identified by using a DII identifier.
         * @param sourceDID A pointer to a DOMDocument from which to retrieve the
         *                  elements.
         * @param value A char pointer to a DII Identifier identifying elements to be
         *              retrieved.
         * @return array of pointers to DOMElements representing the retrieved DIDL
         *          elements. The last element of the array must be a null pointer.
         */
        virtual DOMElement** getElementsByIdentifier (DOMDocument* sourceDID,
                                                      char* value) const throw (DIPError) = 0;

        /**
         * Retrieves DIDL elements identified by using RelatedIdentifiers.
         * @param sourceDID A pointer to a DOMDocument from which to retrieve the
```

```
         *                   elements.
         * @param value A char pointer to a DII RelatedIdentifier identifying the elements
         *              to be retrieved.
         * @return array of pointers to DOMElements representing the retrieved DIDL
         *          elements. The last element of the array must be a null pointer.
         */
        virtual DOMElement** getElementsByRelatedIdentifier(DOMDocument* sourceDID,
                                                char* value) const throw (DIPError) = 0;

        /**
         * Retrieves DIDL elements identified by using DII Type. The last element of this
         * array must be a null pointer.
         * @param sourceDID A pointer to a DOMDocument from which to retrieve the
         *                  elements.
         * @param value A char pointer to a DII Type identifying the type of the elements
         *              to be retrieved.
         * @return array of pointers to DOMElements representing the retrieved DIDL
         *          elements. The last element of the array must be a null pointer.
         */
        virtual DOMElement** getElementsByType(DOMDocument* sourceDID, char* value) const
                                                                        throw (DIPError) = 0;
};

#endif
```

E.5.7 DIP related operations

See 5.4.2.7.

```
#ifndef DIP_H
#define DIP_H

#include <DOMDocument.h>
#include <DOMElement.h>

#include "ObjectMap.h"
#include "PlayStatus.h"
#include "DIPError.h"

/**
 * C++ interface bindings for the DIP related DIBOs.
 */
class DIP {
  public:
        /**
         * Alerts the User of some circumstance via a given message.
         * @param message A char pointer containing the message to be displayed.
         * @param messageType an int value indicating the generic nature of the message
         *                    (MSG_INFO, MSG_WARNING, MSG_ERROR, and MSG_PLAIN).
         * @return void.
         */
        virtual void alert(char* message, int messageType) const = 0;

        /**
         * Requests to execute the given resource.
         * @param element A pointer to the DID object, being a DOMElement, reflecting the
         *              DIDL Resource element representing the resource.
         * @param arguments An array of void pointers to arguments given to the
         *                  executable. The last element of the array shall be a null
         *                  pointer.
         * @return boolean value of true if the resource execution was successfully
         *          initiated, or false if resource execution was not initiated.
```

```
     */
    virtual bool execute(DOMElement* element, void** arguments) const
                                                              throw (DIPError) = 0;

    /**
     * Requests the User to choose resources located external to the Digital Item.
     * @param mimetypes An array of arrays of char pointers specifying the required
     *                  media formats. The last element of the array must be a null
     *                  pointer.
     * @param requestMessages An array of either char pointers or null pointers. The
     *                        last element of the array must be a null pointer.
     * @return array of char pointers specifying the URLs that describe the location
     *         of the resources. The last element of the array must be a
     *         null pointer.
     */
    virtual char** getExternalData(char*** mimeTypes, char** requestMessages) const
                                                              throw (DIPError) = 0;

    /**
     * Retrieves the Object Map from a DIDL Document.
     * @param document A pointer to a DOMDocument representing the DID instance
     *                 document containing the Object Type information.
     * @return pointer to ObjectMap object representing Object Map of the DID instance
     *         document.
     */
    virtual ObjectMap* getObjectMap(DOMDocument* document) const
                                                              throw (DIPError) = 0;

    /**
     * Requests the User to choose available DID objects of the given Object Types
     * from the current DID instance document.
     * @param objectTypeNames An array of char pointers specifying the Object Type
     *                        names. The last element of the array must be a null
     *                        pointer.
     * @param requestMessages An array of char pointers or null pointers. The last
     *                        element of the array must be a null pointer.
     * @return array of pointers to DOMElements or null pointers. A DOMElement
     *         object represents the selected element of the corresponding Object
     *         Type as specified in the objectTypeNames array. The last element of
     *         the array must be a null pointer.
     */
    virtual DOMElement** getObjects(char** objectTypeNames,
                                    char** requestMessages) const throw (DIPError) = 0;

    /**
     * Requests the User to supply values for boolean values, char pointers, or int
     * data.
     * @param dataTypes An Array of char pointers specifying the data type of each
     *                  datum. The last element of the array must be a null pointer.
     * @param requestMessages An array of char pointers or null pointers. The last
     *                        element of the array must be a null pointer.
     * @return rray of void pointers representing boolean values, char
     *         pointers or int data, corresponding to the data type specified in the
     *         dataTypes array. The last element of the array must be a null pointer.
     */
    virtual void** getValues(char** dataTypes, char** requestMessages) const
                                                              throw (DIPError) = 0;

    /**
     * Requests to play the given resource.
     * @param element A pointer to the DOMElement that reflects the DIDL Resource
     *                element describing the media resource.
     * @param async If this boolean value is true, play the resource asynchronously,
     *              else synchronously.
     * @return pointer to PlayStatus object to identify the playing resource.
     */
    virtual PlayStatus* play(DOMElement* element, bool async)
                                                              throw (DIPError) = 0;
```

```
        /**
         * Requests to print the given resource.
         * @param element A pointer to the DOMElement that reflects the DIDL Resource
         *                element representing the media resource.
         * @return boolean value of true if the element was successfully printed, or
         *         false if it was not printed.
         */
        virtual bool print(DOMElement* element) throw (DIPError) = 0;

        /**
         * Requests that playing of a resource is stopped and any associated state
         * information be released.
         * @param playStatus A pointer to a PlayStatus object associated with the playing
         *                   resource.
         * @return void.
         */
        virtual void release(PlayStatus* playStatus) throw (DIPError) = 0;

        /**
         * Runs an identified DIM.
         * @param itemIdType A char pointer indicating the type of identifier (DII
         *                   Identifier or URI) that is given by the itemID parameter.
         * @param itemId A char pointer identifying the ITEM that contains the DIM
         *               declaration of the DIM to be run or null pointer.
         * @param componentIdType A char pointer indicating the type of identifier (DII
         *                        Identifier or URI) that is given by the componentID
         *                        parameter.
         * @param componentId A char pointer identifying the COMPONENT that contains the DIM
         *               declaration of the DIM to be run or null pointer.
         * @param arguments An array of void pointers that are to be the arguments to be
         *                  passed on to the invoked DIM. The last element of the array
         *                  must be a null pointer.
         * @return void.
         */
        virtual void runDIM(char* itemIdType, char* itemId, char* componentIdType,
                            char* componentId, void** arguments) throw (DIPError) = 0;

        /**
         * Runs an identified J-DIXO.
         * @param itemIdType A char pointer indicating the type of identifier (DII
         *                   Identifier or URI) that is given by the itemID parameter.
         * @param itemId A char pointer identifying the ITEM that contains the DIM
         *               declaration of the DIM to be run or null pointer.
         * @param componentIdType A char pointer indicating the type of identifier (DII
         *                        Identifier or URI) that is given by the componentID
         *                        parameter.
         * @param componentId A char pointer identifying the COMPONENT that contains the DIM
         *               declaration of the DIM to be run or null pointer.
         * @param className A char pointer representing the fully qualified class name of
         *                  the J-DIXO to be run.
         * @param arguments An array of void pointers that are to be the arguments to be
         *                  passed on to the invoked DIM. The last element of the array
         *                  must be a null pointer.
         * @return void.
         */
        virtual void runJDIXO(char* itemIdType, char* itemId, char* componentIdType,
               char* componentId, char* className, void** arguments) throw (DIPError) = 0;

        /**
         * Pauses the executing of the invoking DIM.
         * @param timeInterval An int value indicating the time in milliseconds for the
         *                     wait operation to pause execution of the DIM.
         * @return void.
         */
        virtual void wait(int timeInterval) = 0;
};
```

```
#endif
```

E.5.8 REL related operations

See 5.4.2.8.

```
#ifndef REL_H
#define REL_H

#include <DOMElement.h>
#include "DIPError.h"

/**
 * C++ interface bindings for the REL related DIBOs.
 */
class REL {
  public:
        /**
         * Requests to retrieve any licenses associated with the given resource.
         * @param resource A pointer to a DOMElement representing the DIDL Resource.
         * @return array of pointers to DOMElements representing any licenses. The last
         *          element of the array must be a null pointer.
         */
        virtual DOMElement** getLicense(DOMElement* resource) const throw (DIPError) = 0;

        /**
         * Requests to check for the existence of an authorization proof for an
         * authorization request.
         * @param license A pointer to a DOMElement representing license information.
         * @param resource A pointer to a DOMElement representing DIDL Resource.
         * @param rightNs A char pointer representing the namespace of the right to be
         *                checked or a null pointer.
         * @param rightLocal A char pointer to the Localname of the right to be checked or
         *                   the value of the definition attribute of xs:rightUri,
         *                   depending on whether rightNs is a char pointer or a null
         *                   pointer, respectively.
         * @param additionalInfo An array of pointers to DOMElements representing
         *                       additional information that can be considered. The last
         *                       element of the array must be a null pointer.
         * @return boolean value of true if a corresponding authorization proof is found,
         *          false if a corresponding authorization proof does not exist or could
         *          not be found.
         */
        virtual bool queryLicenseAuthorization(DOMElement* license, DOMElement* resource,
                            char* rightNs, char* rightLocal, DOMElement** additionalInfo)
                                                                        throw (DIPError) = 0;
};

#endif
```