INTERNATIONAL STANDARD

ISO/IEC 21000-10

First edition
2006-01-01

Information technology — Multimedia framework (MPEG-21) —

Part 10: Digital Item Processing

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

Partie 10: Traitement d'élément numérique

Reference number
ISO/IEC 21000-10:2006(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

Contents

Page

Foreword

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

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

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

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

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

ISO/IEC 21000 consists of the following parts, under the general title *Information technology — Multimedia framework (MPEG-21)*:

— *Part 1: Vision, Technologies and Strategy* [Technical Report]

— *Part 2: Digital Item Declaration*

— *Part 3: Digital Item Identification*

— *Part 4: Intellectual Property Management and Protection Components*

— *Part 5: Rights Expression Language*

— *Part 6: Rights Data Dictionary*

— *Part 7: Digital Item Adaptation*

— *Part 8: Reference Software*

— *Part 9: File Format*

— *Part 10: Digital Item Processing*

— *Part 11: Evaluation Tools for Persistent Association Technologies* [Technical Report]

— *Part 12: Test Bed for MPEG-21 Resource Delivery* [Technical Report]

— *Part 16: Binary Format*

The following parts are under preparation:

— *Part 14: Conformance Testing*

— *Part 15: Event Reporting*

— *Part 17: Fragment Identification of MPEG Resources*

— *Part 18: Digital Item Streaming*

Introduction

Today, many elements exist to build an infrastructure for the delivery and consumption of multimedia content. There is, however, no “big picture” to describe how these elements, either in existence or under development, relate to each other. The aim for ISO/IEC 21000 (MPEG-21) is to describe how these various elements fit together. Where gaps exist, MPEG-21 will recommend which new standards are required. ISO/IEC JTC 1/SC 29/WG 11 (MPEG) will then develop new standards as appropriate while other relevant standards may be developed by other bodies. These specifications will be integrated into the multimedia framework through collaboration between MPEG and these bodies.

The result is an open framework for multimedia delivery and consumption, with both the content creator and content consumer as focal points. This open framework provides content creators and service providers with equal opportunities in the MPEG-21 enabled open market. This will also be to the benefit of the content consumers, providing them access to a large variety of content in an interoperable manner.

The vision for MPEG-21 is to define a multimedia framework *to enable transparent and augmented use of multimedia resources across a wide range of networks and devices* used by different communities.

A key concept of the multimedia framework is the Digital Item. In MPEG-21 a Digital Item is a structured digital object with a standard representation, identification, and metadata. An equally important concept in the multimedia framework is the notion of the User. In MPEG-21 a User is any entity that interacts with the multimedia framework and as such includes all members of the value chain (e.g., creator, rights holders, distributors and consumers of Digital Items) and include, for example, individuals, consumers, communities, organizations, corporations, consortia, and governments.

Part 2 of MPEG-21 specifies the mechanism for declaring the structure and makeup of Digital Items. Such Digital Item Declarations are static by nature. This 10th part of MPEG-21 specifies tools enabling Users to provide suggested interactions with Digital Items, thereby enabling the inclusion of a dynamic aspect to the static declaration of Digital Items.

Information technology — Multimedia framework (MPEG-21) —

Part 10:
Digital Item Processing

1 Scope

This Part of ISO/IEC 21000, entitled Digital Item Processing (DIP), specifies the syntax and semantics of tools that may be used to process Digital Items. The tools provide a normative set of tools that specify the processing of a Digital Item in a predefined manner.

This technology is specified in one normative clause and three normative annexes:

— Digital Item Methods:

  Digital Item Methods (Clause 5) specifies the set of tools enabling Digital Item Users to include sequences of instructions for adding predefined functionality to a Digital Item. Such a sequence of instructions is a Digital Item Method. Digital Item Methods are authored with the Digital Item Method Language (see 5.2) which includes bindings to Digital Item Base Operations (see 5.4). For extended functionality, Digital Item eXtension Operations (see 5.6) allow such processing to be implemented more efficiently in a higher level programming language. Tools for integrating Digital Item Methods into Digital Item Declarations are also specified (see 5.3).

— ECMAScript bindings for Digital Item Base Operations:

  Annex A specifies the ECMAScript bindings for the Digital Item Base Operations described in 5.3.

— Java bindings for Digital Item Base Operations:

  Annex B specifies the Java bindings for the Digital Item Base Operations described in 5.4.

— Calling Java based DIXOs from Digital Item Methods:

  Annex C specifies the mechanism for calling Java based Digital Item eXtension Operations. Digital Item eXtension Operations are described in 5.6.

2 Normative references

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

ISO/IEC 16262:2002, *Information technology — ECMAScript language specification*

ISO/IEC 21000 (all parts), *Information technology — Multimedia framework (MPEG-21)*

IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, 1996

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, 2005

W3C REC-DOM-Level-3-Core-20040407, *Document Object Model (DOM) Level 3 Core Specification, Version 1.0*, W3C Recommendation 07 April 2004

W3C REC-DOM-Level-3-LS-20040407, *Document Object Model (DOM) Level 3 Load and Save Specification, Version 1.0*, W3C Recommendation 07 April 2004

W3C REC-xml-20040204, *Extensible Markup Language (XML) 1.0 (Third Edition)*, W3C Recommendation 04 February 2004

W3C REC-xml-names-19990114, *Namespaces in XML*, World Wide Web Consortium 14 January 1999

W3C REC-xmlschema-1-20041028, *XML Schema Part 1: Structures Second Edition*, W3C Recommendation 28 October 2004

W3C REC-xmlschema-2-20041028, *XML Schema Part 2: Datatypes Second Edition*, W3C Recommendation 28 October 2004

W3C REC-xpath-19991116, *XML Path Language (XPath), Version 1.0*, W3C Recommendation 16 November 1999

3 Terms, definitions, and abbreviated terms

For the purposes of this part of ISO/IEC 21000, the following terms, definitions and abbreviations apply:

3.1
Argument Type
type of the Digital Item Method Argument specified by an `Argument` element of the associated Digital Item Method declaration

NOTE 1 Argument Type is part of the Object Map allowing mapping of DIM Arguments to DID Objects.

NOTE 2 For further information see 5.3.5.

3.2
Digital Item
DI
structured digital object, including a standard representation, identification and metadata within the MPEG-21 framework

NOTE This entity is the fundamental unit of distribution and transaction within the multimedia framework as a whole.

[ISO/IEC TR 21000-1:2004, definition 2.3]

3.3
DIA
Digital Item Adaptation as specified by ISO/IEC 21000-7

3.4
Digital Item Base Operation
DIBO
base operation providing access to functionality implemented within an MPEG-21 environment and used in authoring a Digital Item Method

NOTE For further information see Clause 5.4.

3.5
Digital Item Base Operation implementation
manner in which a particular implementer of a Digital Item Base Operation chooses to implement the normative semantics of the Digital Item Base Operation

3.6
Digital Item Declaration
DID
declaration of the resources, metadata and their interrelationships of a Digital Item specified by ISO/IEC 21000-2

3.7
Digital Item Declaration Language
DIDL
XML-based language including validation rules specified by ISO/IEC 21000-2 for the standard representation in XML of a Digital Item Declaration

3.8
Digital Item Declaration Language document
a document using the Digital Item Declaration Language to declare a Digital Item in a standard representation in XML specified by ISO/IEC 21000-2

3.9
Digital Item Declaration Language element
XML element of the Digital Item Declaration Language specified by ISO/IEC 21000-2

3.10
Digital Item Declaration Model
set of abstract terms and concepts specified by ISO/IEC 21000-2 forming a model for declaring Digital Items

3.11
Digital Item Declaration Model entity
entity of the Digital Item Declaration Model specified by ISO/IEC 21000-2

3.12
Digital Item Declaration Object
Object
object representation in the Digital Item Method Language of a Digital Item Declaration element and associated with an Object Type

NOTE 1 A Digital Item Declaration Object has an Object Type that allows it to be processed in a Digital Item Method according to the Object Type. A Digital Item Declaration element is mapped to an Object Type by the Object Map.

NOTE 2 An Object Type is associated with a Digital Item Declaration element by an `ObjectType` element contained in a DESCRIPTOR-STATEMENT child of the Digital Item Declaration element. For further information see 5.3.

NOTE 3 The capitalized term Object is used in this part of ISO/IEC 21000 to mean a Digital Item Declaration Object. Other uses of the term object without an initial uppercase letter is used for an object as understood in the context of object-oriented programming.

3.13
Digital Item eXtension Operation
DIXO
operation allowing extended functionality to be invoked from a Digital Item Method

NOTE For further information see Clause 5.6.

3.14
Digital Item eXtension Operation Language
DIXL
programming language in which a Digital Item eXtension Operation is defined

3.15
DII
Digital Item Identification as specified by ISO/IEC 21000-3

3.16
Digital Item Method
DIM
tool for expressing the suggested interaction of a User with a Digital Item at the level of the Digital Item Declaration

NOTE 1 For further information see Clause 5.

NOTE 2 A Digital Item Method is composed of a Digital Item Method definition and its declaration.

3.17
Digital Item Method Argument
argument to a Digital Item Method as represented in the Digital Item Method Language

3.18
Digital Item Method declaration
declaration of the Digital Item Method as being part of a particular Digital Item

NOTE For further information see Clause 5.3.

3.19
Digital Item Method definition
code written in the Digital Item Method Language that defines the Digital Item Method and that is either embedded inline with the Digital Item Method declaration or located separately and referenced from the Digital Item Method declaration

NOTE Whether the Digital Item Method definition is embedded inline or referenced from a separate location, it is the Digital Item Method definition itself that is the *resource* (in terms of the Digital Item Declaration Model).

3.20
Digital Item Method Language
DIML
language providing the syntax and structure for authoring a Digital Item Method utilizing the Digital Item Base Operations

NOTE For further information see Clause 5.2

3.21
Digital Item Processing engine
component within an MPEG-21 environment that supports ISO/IEC 21000-10 and is responsible for providing such supporting functionality (including execution of Digital Item Methods)

3.22
DOM
Document Object Model (see W3C REC-DOM-Level-3-Core-20040407)

3.23
End User
User taking the role of consumer, i.e. being at the end of a value or delivery chain

EXAMPLE A human consumer or an agent operating on behalf of a human consumer, etc.

[ISO/IEC TR 21000-1:2004, definition 2.4]

3.24
GUI
Graphical User Interface

3.25
IPMP
Intellectual Property Management and Protection as specified by ISO/IEC 21000-4

3.26
JPEG
Joint Photographic Experts Group

3.27
MIME
Multipurpose Internet Mail Extensions (see IETF RFC 2046)

3.28
MP3
MPEG-1/2 layer 3 (audio coding)

3.29
MPEG
Moving Picture Experts Group

3.30
Object Map
map of Digital Item Declaration elements in a Digital Item Declaration to Digital Item Declaration Objects with their associated Object Types

EXAMPLE An object map might map several ITEM elements to an Object Type of “music track”, and another ITEM element to an Object Type of “album information”.

NOTE For further information see 5.3.5.

3.31
Object Type
type of the Digital Item Declaration Object specified by an `ObjectType` descriptor of the associated DIDL element

NOTE 1 Object Type is part of the Object Map allowing mapping of DID Objects to DIM Arguments.

NOTE 2 For further information see Clause 5.3.5.

3.32
Peer
device or application that compliantly processes a Digital Item

[ISO/IEC TR 21000-1:2004, definition 2.7]

3.33
RDD
Rights Data Dictionary as specified by ISO/IEC 21000-6

3.34
REL
Rights Expression Language as specified by ISO/IEC 21000-5

3.35
URI
Uniform Resource Identifier (see IETF RFC 3986)

3.36
URL
Uniform Resource Locator (see IETF RFC 3986)

3.37
User
entity that interacts in the MPEG-21 environment or makes use of Digital Items

NOTE This includes all members of the value chain (e.g., creator, rights holders, distributors and consumers of Digital Items).

[ISO/IEC TR 21000-1:2004, definition 2.9]

3.38
W3C
World Wide Web Consortium

3.39
XML
Extensible Markup Language (see W3C REC-xml-20040204)

4 Overview and Conventions

4.1 Overview of Digital Item Processing

The Digital Item Declaration Language described in ISO/IEC 21000-2 is for creating a static declaration. Digital Item Processing assists processing of a Digital Item by providing tools allowing a User to add User specified functionality to a Digital Item Declaration. The standardization of Digital Item Processing enables interoperability at the processing level.

A key component of Digital Item Processing is the Digital Item Method (see 5). A Digital Item Method is the tool whereby a User (as defined in 21000-1) specifies suggested interactions with the Digital Item. As such, Digital Item Methods provide a way for a User to specify a selection of suggested procedures for processing a Digital Item at the level of the Digital Item itself.

EXAMPLE A Digital Item representing a music album can contain a Digital Item Method to add a new music track to the album. Such a Digital Item Method can be used to ensure that the new music track is added to the Digital Item while maintaining a suggested format for the Digital Item Declaration of such a music album Digital Item (i.e., elements added in the correct place in the Digital Item Declaration structure, correct Descriptors are included, etc.).

A Digital Item Method is expressed using the Digital Item Method Language (see 5.2) which includes a binding for Digital Item Base Operations (see 5.4). Digital Item eXtension Operations (see 5.6) provide a mechanism that allows the functionality provided by the standard set of Digital Item Base Operations to be extended.

Digital Item Methods, and the Digital Item Base Operations and Digital Item eXtension Operations called by them, can be considered as requests to the Digital Item Processing engine to process the Digital Item in some manner, or to execute some action.

The interface through which a User interacts with a Digital Item using Digital Item Processing is implementation dependent. Some implementations might support specification of aspects of the interface by metadata included in the Digital Item. Some possible scenarios follow.

— On receipt of a Digital Item Declaration, a list of Digital Item Methods (contained or referenced from within the DIDL document representing the Digital Item) can be presented to the User. The User can choose a Digital Item Method and then the Objects on which it operates. The Digital Item Processing engine then executes the chosen Digital Item Method with the chosen Objects as arguments;

— On receipt of a Digital Item Declaration, a list of Objects is presented based on the presence of Identifiers of the DII XML Namespace. The User chooses one or more of these Object(s). A list of Digital Item Methods that takes as arguments the (set of) Object(s) is then presented to the User. The User selects a Digital Item Method that is then executed by the Digital Item Processing engine.

4.2 Relation of Digital Item Processing with other parts of ISO/IEC 21000

Digital Item Processing is related to ISO/IEC 21000-2 by providing normative tools that enable functionality to be included in a Digital Item.

Implementations of DIBOs might have requirements or choices of implementation related to other parts of ISO/IEC 21000. ISO/IEC 21000-4, for instance, is expected to require that DIBO implementations accessing governed resources are required to check for permissions before doing so. In such cases, DIBO implementations would check for permissions and, in so doing, may take advantage of information compliant with ISO/IEC 21000-5 and ISO/IEC 21000-6. DIBO implementations may also make use of information compliant with ISO/IEC 21000-7, if appropriate to the DIBO semantics.

NOTE Annex G provides guidance on how to support DIP while maintaining a level of interaction with a Digital Item that is consistent with the available rights.

Overall processing of a Digital Item remains largely at the discretion of an application. Digital Items are intended to be used throughout the delivery chain, and thus different applications and different Users will perform different overall processing of a Digital Item. Digital Item Methods can be regarded as a 'menu' of User interaction possibilities. Digital Item Methods can then be utilized during processing of Digital Items to understand the Digital Item Method author's suggested manner of User interaction with a Digital Item. Different Digital Item Methods can be authored such that they provide different suggested interactions appropriate for different Users at various junctures in the delivery chain. This part of ISO/IEC 21000 specifies how to author Digital Item Methods and integrate them in a Digital Item Declaration. It does not specify how to restrict access to a Digital Item Method. This can be achieved, by utilizing other parts of ISO/IEC 21000 such as ISO/IEC 21000-4 and ISO/IEC 21000-5.

4.3 Documentation conventions

Literal machine-readable character sequences are shown in `fixed width font`.

References to DID Model entity names are shown in *italics*.

References to DIDL element names are shown in FIXED WIDTH SMALL CAPS FONT.

Normative syntax for DIP tools specified by XML Schema declarations and definitions are shown in this document using a separate font and background as follows.

EXAMPLE

```
<complexType name="ExampleType">
   <simpleContent>
      <extension base="string"/>
   </simpleContent>
</complexType>
```

XML Schema declarations and definitions as in the above example are to be considered fragments of a complete schema within an XML schema wrapper as described in 4.4.

Normative semantics for XML Schema declarations and definitions are set out in a table as follows.

EXAMPLE

Semantics of `ExampleType`:

Name	*Definition*
`ExampleType`	Example type semantics.

4.4 Schema wrapper

XML Schema declarations and definitions provided as XML fragments are to be understood as fragments of a complete schema and contained within an XML Schema `schema` element as follows.

```
<?xml version="1.0"?>
<!-- Digital Item Processing ISO/IEC 21000-10 -->
<schema
    targetNamespace="urn:mpeg:mpeg21:2005:01-DIP-NS"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:dip="urn:mpeg:mpeg21:2005:01-DIP-NS"
    elementFormDefault="qualified" attributeFormDefault="unqualified">

</schema>
```

4.5 Use of namespace prefixes

Qualified Names are written with a namespace prefix followed by a colon followed by the local part of the Qualified Name as shown in the following example.

EXAMPLE `dip:ObjectType`

For clarity, consistent namespace prefixes as listed below are used in this part of ISO/IEC 21000.

Table 1 — Mapping of prefixes to namespaces

Prefix	**Namespace**
`dia`	`urn:mpeg:mpeg21:2003:01-DIA-NS`
`didl`	`urn:mpeg:mpeg21:2002:02-DIDL-NS`
`dii`	`urn:mpeg:mpeg21:2002:01-DII-NS`
`dip`	`urn:mpeg:mpeg21:2005:01-DIP-NS`
`xsd`	`http://www.w3.org/2001/XMLSchema`
`xsi`	`http://www.w3.org/2001/XMLSchema-instance`
`xi`	`http://www.w3.org/2001/XInclude`
NOTE The prefixes `xml` and `xmlns` are normatively defined by *Namespaces in XML (see* W3C REC-xml-names-19990114*)*. All other prefixes are not normative and are used by convention for consistency in this part of ISO/IEC 21000.	

For informative examples provided as XML fragments without namespace declarations, the default namespace by convention in this part of ISO/IEC 21000 is defined as `urn:mpeg:mpeg21:2002:02-DIDL-NS` and the different prefixes are bound to the namespaces as listed above.

5 Digital Item Methods

5.1 Introduction

Digital Item Methods (DIMs) provide for a programmatic invocation of Digital Item Base Operations (DIBOs). Thereby a suggested interaction of the User with the Digital Item is provided. DIMs should thus be viewed from a User perspective; they are intended to be related to User interaction with a Digital Item.

DIMs are intended for working with parts of a DI at the DID level. DIMs are not intended to be utilized for implementing the processing of media resources themselves.

EXAMPLE DIMs are not intended to be used for implementing transcoding of media resources. However DIMs might be used for adaptations of the DID at the DID level.

NOTE While the intention is that media resource processing is not implemented directly within a DIM definition, DIBO implementations, such as the `adapt` DIBO (see 5.4.2.4.2), might have semantics that can lead to media resource processing. In addition media resource processing might take place in a DIXO (see 5.6).

Arguments to DIMs are DID Objects representing DIDL elements. The relationship between DIDL elements, DID Objects and DIM Arguments is specified by the Object Map (for further information on the Object Map see 5.3.5).

Digital Item Method tools include the following.

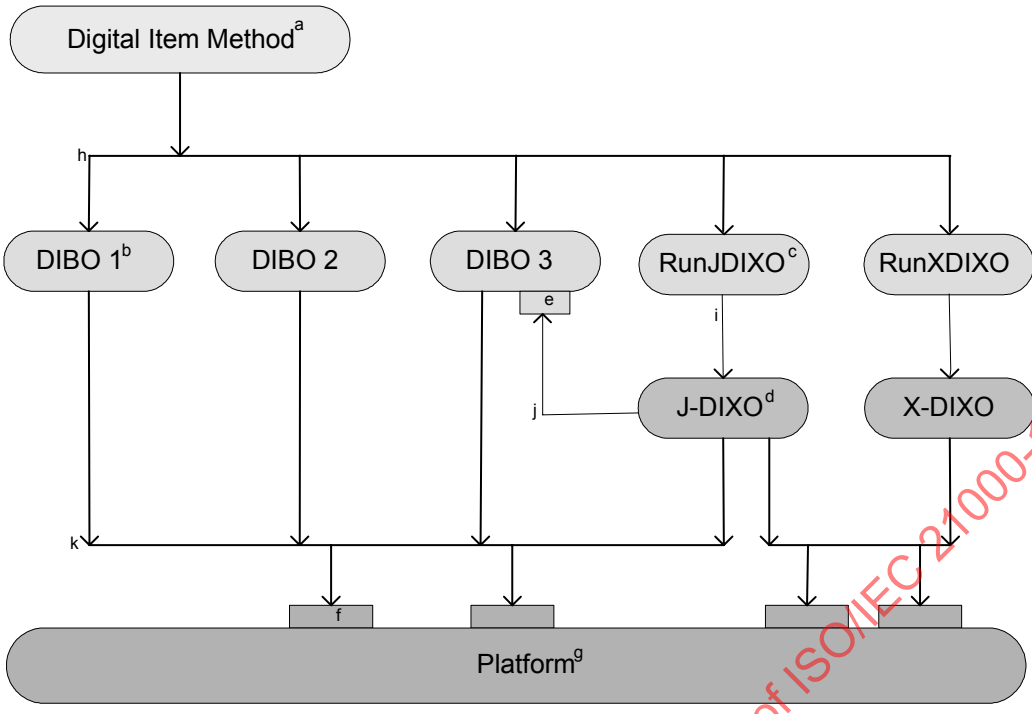
— Digital Item Base Operations (DIBOs) – the Digital Item Base Operations specify a high level normative interface to the basic types of interaction with a Digital Item. The set of normatively defined DIBOs have general application across a wide range of resources, applications, etc.;

— Digital Item Method Language (DIML) – the Digital Item Method Language specifies the normative language for defining interoperable DIMs and from which the DIBOs are able to be called. The ECMAScript binding of the DIBOs is a normative part of the DIML;

— Digital Item Method linkage with DID – this clause also specifies the normative mechanisms for including of Digital Item Methods in a DID;

— Digital Item Method execution – this clause also specifies the execution environment of a DIM; and

— Digital Item eXtension Operations (DIXO) – Digital Item eXtension Operations specify a normative mechanism for enabling functionality that extends beyond the basic functionality of the normative set of DIBOs in an efficient way.



a Represents a Digital Item Method definition. The DIM definition is defined using the Digital Item Method Language and may call DIBOs and DIXOs (via a defined DIBO to call DIXOs defined in a given DIXO Language).

b These components represent Digital Item Base Operation implementations. DIBO implementations may call other modules provided by the platform.

c These components represent certain DIBOs that are defined to call DIXOs defined in a given DIXO Language. Currently only a RunJDIXO DIBO is defined for calling DIXOs using Java as the DIXO Language.

d These components represent Digital Item eXtension Operation definitions. DIXO definitions may call DIBOs via the bindings to the DIBOs in the DIXO Language as well as other modules provided by the platform.

e This represents a binding to a DIBO in the DIXO Language used for the DIXO definitions.

f This represents the API(s) to other modules provided by the platform which the DIBO implementations and DIXO definitions may call. This could include modules providing functionality related to other parts of ISO/IEC 21000.The API(s) available to DIBO implementations and DIXO definitions might be different.

g The platform refers to the environment in which the DIBOs are implemented and the DIMs are executed.

h This indicates calls from the DIM definition to DIBOs, including the DIBOs to call DIXOs defined in a given DIXO Language.

i This indicates calls from the implementation of the defined DIXO calling DIBOs to the DIXO being called.

j This indicates calls from a DIXO definition to a DIBO via the binding to the DIBO in the DIXO Language.

k This indicates calls from the DIBO implementations and DIXO definitions to modules provided by the platform.

Figure 1 — Digital Item Method components

5.2 Digital Item Method Language

5.2.1 Core language

The DIML specification normatively includes the ECMAScript Language Specification defined by ISO/IEC 16262:2002. This provides a standardized core language specification for DIML, including specification of the following features of DIML.

— Lexical conventions and syntactical structure;

— Flow control structures (i.e., if/then, while, etc.);

— Primitive data types (String, Boolean and Number);

— Composite data types (Objects and arrays);

— Standard arithmetic, logical and bitwise operators;

— Exception handling;

— Error definitions; and

— Support for Regular Expressions.

NOTE 1 ISO/IEC 16262:2002 is equivalent to ECMA-262 (edition 3).

NOTE 2 ISO/IEC 16262:2002 defines an ***object*** as an unordered collection of properties each of which contains a primitive value, object, or function. This part of ISO/IEC 21000 specifies DIML as an extension of ECMAScript by specifying additional objects. The DIBOs specified by this part of ISO/IEC 21000 are specified as function properties of these additional objects, hence in this part of ISO/IEC 21000 the term DIBO is used equivalently for the function properties of the objects specified by DIML.

5.2.2 Digital Item Base Operations

The DIML specification includes the normative set of DIBOs specified in 5.4.

5.2.3 Digital Item eXtension Operations

The DIML specification allows the calling of DIXOs as specified in 5.6.

5.3 Digital Item Method linkage with DID

5.3.1 Introduction

This clause specifies how DIMs are incorporated into a DID.

5.3.2 XML Schema for DIP

DIMs are incorporated into a DID using DIP defined XML (W3C REC-xml-20040204) elements that are incorporated at appropriate locations within the DIDL of the DID. W3C XML Schema (W3C REC-xmlschema-1-20041028, W3C REC-xmlschema-2-20041028) is used to define these DIP elements and the complete XML Schema for DIP is found in Annex E.

5.3.3 Digital Item Method declaration

5.3.3.1 Introduction

A Digital Item Method declaration is contained in a DIDL COMPONENT element which shall be constructed such that:

— The COMPONENT should contain a DIM information descriptor represented by a DIP `MethodInfo` element contained in a DIDL DESCRIPTOR-STATEMENT. If the DIM does not have any arguments, the `MethodInfo` element need not be present;

— The COMPONENT may contain a flag, represented by a DIP `Label` element contained in a DIDL DESCRIPTOR-STATEMENT, indicating the DIM declaration is to be processed by the DIP engine; and

— The Digital Item Method definition is referenced or embedded by a RESOURCE child of the COMPONENT.

NOTE 1 A DIM declaration can be included from an external DIDL document by utilizing provisions for document modularity as specified in ISO/IEC 21000-2.

NOTE 2 An identifier can be associated with the DIM declaration using a DII Identifier (as specified by part 3 of ISO/IEC 21000). This could be located at the level of the COMPONENT, or at the level of the ITEM containing the COMPONENT (perhaps depending on the particular application of the DI). The `RunDIM` DIBO (see 5.4.2.7.11) supports both levels of identification.

NOTE 3 Since the `MethodInfo` element need not be present if the DIM does not have any arguments, it is not intended to be used to locate DIMs in a DID. For DIMs intended to be processed by the DIP engine, the `Label` element is used. In other cases the `mimeType` attribute of the RESOURCE can be used.

5.3.3.2 MethodInfo syntax

```
<!--

################################################
#  Definition of MethodInfo                    #
################################################
-->
<element name="MethodInfo" type="dip:MethodInfoType"/>
<complexType name="MethodInfoType">
    <sequence>
        <element name="Argument" type="anyURI" minOccurs="0"
                 maxOccurs="unbounded"/>
    </sequence>
    <attribute name="autoRun" type="boolean" use="optional"
               default="false"/>
    <attribute name="profileCompliance"
               type="dip:ProfileComplianceType" use="optional"/>
</complexType>
<!--

################################################
# Definition of ProfileComplianceType          #
################################################
-->
<simpleType name="ProfileComplianceType">
  <list itemType="QName"/>
</simpleType>
```

5.3.3.3 MethodInfo Semantics

Semantics of `MethodInfo`:

Name	*Definition*
`MethodInfo`	Content of this element is a sequence of `Argument` child elements. A DIDL COMPONENT representing a DIM declaration should contain a DIP `MethodInfo` element contained in a DIDL DESCRIPTOR-STATEMENT. There shall be one `Argument` child element for each argument of the DIM definition. The sequence of `Argument` child elements shall match the sequence of arguments of the DIM definition. If the DIM definition has no arguments and the `MethodInfo` element is still present, it shall contain zero `Argument` child elements.
`Argument`	Content of this element is a URI (IETF RFC 3986) indicating the Object Type associated with the corresponding argument of the DIM definition. There shall be a one-to-one mapping between the arguments of the DIM definition and a corresponding `Argument` element associating an Object Type with the

Name	*Definition*
	argument. The `Argument` elements shall be listed in the same sequence as the arguments of the DIM definition.
`autoRun`	This optional attribute allows the DIM author to identify an auto run DIM. If present and the value is true, then the declared DIM is an auto run DIM. For further information on auto run DIM see 5.7.
`profileCompliance`	This optional attribute allows the DIM author to signal a profile or profiles to which the DIM conforms. If present, each member in the list is a Qualified Name representing one such profile. If present, this attribute need not provide a complete list of profiles. Processors may ignore this attribute. If they do process it, they may ignore any members of the list. NOTE Resolving the namespace prefix of the Qualified Name representing a profile to a namespace name URI reference is done using standard XML processing. That is, using an XML namespace declaration.

5.3.3.4 Label syntax

```
    <!--

    ################################################
    # Definition of Label                          #
    ################################################
    -->
    <element name="Label" type="dip:LabelType"/>
    <complexType name="LabelType">
        <simpleContent>
            <extension base="anyURI"/>
        </simpleContent>
    </complexType>
```

5.3.3.5 Label semantics

Semantics of `Label`:

Name	*Definition*
`Label`	Content of this element is a URI (IETF RFC 3986). The presence of a DIP `Label` element contained in a DIDL DESCRIPTOR-STATEMENT indicates the parent element of the DESCRIPTOR is to be processed by the DIP engine. If present, such a DESCRIPTOR shall be the child of a COMPONENT representing a DIM declaration or DIXO declaration. For a DIM declaration that is to be processed by the DIP engine, the `Label` element shall be present and the value of the URI shall be `urn:mpeg:mpeg21:2005:01-DIP-NS:DIM`. For a DIXO declaration that is to be processed by the DIP engine, the `Label` element shall be present and the value of the URI shall be of the form `urn:mpeg:mpeg21:2005:01-DIP-NS:DIXO:`*x*, where *x* is unique for the DIXO language of the DIXO definition.

Name	*Definition*
	NOTE 1 The URI value for a `Label` of a J-DIXO is given in C.3.2.4. If a DIM declaration or DIXO declaration is not intended to be processed by the DIP engine, the `Label` element should be absent. NOTE 2 Processing by the DIP engine of a COMPONENT as a DIM declaration (or DIXO declaration) makes the DIM (or DIXO) available to the User for interacting with the DI (or calling from a DIM). If a DI is intended to be used, for example, to store or transport DIM declarations (and/or DIXO declarations), then these declarations can exclude a `Label`. When such a DI is processed, the declarations will not be processed by the DIP engine.

5.3.3.6 Digital Item Method declaration examples

EXAMPLE The following example shows several different DIM declarations. In the example the DIM declarations are grouped together in a single DIDL ITEM, however this is not mandated by this part of ISO/IEC 21000.

The first DIM declaration provides an example of an empty `MethodInfo` element for a DIM that has no arguments. This DIM declaration also shows an example of an external reference to the DIM definition by referencing the location of the DIM definition using the `ref` attribute of the DIDL RESOURCE element. In addition, a second DIDL RESOURCE element referencing an equivalent DIM definition at an alternate location is provided.

The second DIM declaration provides an example of a `MethodInfo` element for a DIM that has two arguments. In this second DIM declaration, the DIM definition is embedded inline in the DIDL RESOURCE element (complete DIM definition not included for this example). This second DIM declaration also shows the use of a DIDL CONDITION element for conditional availability of this DIM declaration.

Both DIM declarations also show an example of including a plain text DIDL DESCRIPTOR-STATEMENT for containing a short human readable description of each DIM.

The third DIM declaration shows an example of signalling a DIM that is compliant with the `acme:basic` profile.

A fourth DIM declaration shows an example of using the provisions for document modularity specified in ISO/IEC 21000-2 to include elements of a DIM declaration contained in an external document.

Each of the DIM declarations discussed above contain a DIDL DESCRIPTOR-STATEMENT containing a DIP `Label`, indicating that these DIM declarations are to be processed by a DIP engine. Hence the DIMs declared by these DIM declarations will be available for User interaction with the DI containing this ITEM.

The DIM declaration in the external document otherDI.xml does not contain a DIP `Label`. In this example the otherDI.xml is being used only to store DIM declarations, and these DIM declarations are not intended to be processed by a DIP engine on processing of the otherDI.xml DI itself.

```
…
<Item id="DIMs">
    <Descriptor>
        <Statement mimeType="text/plain">DIM Declarations</Statement>
    </Descriptor>
    <Choice>
        <Selection select_id="selection1"/>
        <Selection select_id="selection2"/>
    </Choice>
    <Component id="dim_01">
        <Descriptor>
            <Statement mimeType="text/plain">Description of DIM 1</Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIM</dip:Label>
            </Statement>
        </Descriptor>
```

```
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:MethodInfo/>
            </Statement>
        </Descriptor>
        <Resource mimeType="application/mp21-method"
            ref="http://www.somewhere.near/DIM#dim_01"/>
        <Resource mimeType="application/mp21-method"
            ref="http://www.somewhere.far/DIM#dim_01"/>
    </Component>
    <Component id="dim_02">
        <Condition require="selection1"/>
        <Descriptor>
            <Statement mimeType="text/plain">Description of DIM 2</Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIM</dip:Label>
            </Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:MethodInfo>
                    <dip:Argument>urn:foo:type1</dip:Argument>
                    <dip:Argument>urn:foo:type2</dip:Argument>
                </dip:MethodInfo>
            </Statement>
        </Descriptor>
        <Resource mimeType="application/mp21-method"><![CDATA[
            function dim_02( arg1, arg2 )
            {
                /* rest of DIM definition goes here */
            }
        ]]></Resource>
    </Component>
    <Component id="dim_03">
        <Condition require="selection2"/>
        <Descriptor>
            <Statement mimeType="text/plain">Description of DIM 3</Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIM</dip:Label>
            </Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:MethodInfo xmlns:acme="urn:acme:demo" profileCompliance="acme:basic">
                    <dip:Argument>urn:foo:type3</dip:Argument>
                </dip:MethodInfo>
            </Statement>
        </Descriptor>
        <Resource mimeType="application/mp21-method"><![CDATA[
            function dim_03( arg1 )
            {
                /* rest of DIM definition goes here */
            }
        ]]></Resource>
    </Component>
    <Component>
        <xi:include href="otherDI.xml" xpointer="element(dim_04/1)"/>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIM</dip:Label>
            </Statement>
        </Descriptor>
        <xi:include href="otherDI.xml" xpointer="element(dim_04/2)"/>
```

```
        <xi:include href="otherDI.xml" xpointer="element(dim_04/3)"/>
    </Component>
</Item>
…
```

otherDI.xml:

```
…
<Component id="dim_04">
    <Descriptor>
        <Statement mimeType="text/plain">Description of DIM 4</Statement>
    </Descriptor>
    <Descriptor>
        <Statement mimeType="text/xml">
            <dip:MethodInfo/>
        </Statement>
    </Descriptor>
    <Resource mimeType="application/mp21-method"><![CDATA[
            function dim_04()
            {
                /* rest of DIM definition goes here */
            }
        ]]></Resource>
</Component>
…
```

5.3.4 Digital Item Method definition

5.3.4.1 Introduction

A DIM definition is embedded in or referenced from a DIDL RESOURCE element. Such a RESOURCE element shall be contained in a DIM declaration as specified in 5.3.3.

The DIM definition itself is the sequence of operations authored using the DIML as specified in 5.2. This includes calling DIBOs, as specified in 5.4, to access the functionality available for User interactions with the DI provided by the implementation of the DIBO semantics.

5.3.4.2 Embedded

A DIM definition may be embedded inline in a DID by base64 encoding the DIM definition prior to embedding or by placing the DIM definition in a CDATA section.

EXAMPLE 1 DIM definition embedded as base64 encoded data

```
…
<Component id="dim_04">
    <Descriptor>
        <Statement mimeType="text/xml">
            <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIM</dip:Label>
        </Statement>
    </Descriptor>
    <Resource mimeType="application/mp21-method" encoding="base64">
        cj5QUkVWMTk5MTQxMDkwNDQ5PC9BY2Nlc3Npb25OdW1iZXI+PERhdGVQcm9kdWNlZD4xOTkxPC9E
        YXRlUHJvZHVjZWQ+PE93bmVyPkNvcHlyaWdodCBCSU9TSVMgMjAwMzwvT3duZXI+PFByb2RJc3N1
        ZVllYXI+MTk5MTwvUHJvZElzc3VlWWVhcj48QWJzdHJhY3RQcmVzZW50WU4+TjwvQWJzdHJhY3RQ
    </Resource>
</Component>
```

```
…
```

EXAMPLE 2 DIM definition embedded in a CDATA section

```
…
<Component id="dim_04">
    <Descriptor>
        <Statement mimeType="text/xml">
            <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIM</dip:Label>
        </Statement>
    </Descriptor>
    <Resource mimeType="application/mp21-method"><![CDATA[
        function dim_04()
        {
            /* rest of DIM definition goes here */
        }
    ]]></Resource>
</Component>
…
```

5.3.4.3 Referenced

A DIM definition is referenced from a DIDL RESOURCE element by utilizing the `ref` attribute of the RESOURCE element. Referencing the DIM definition from the RESOURCE element allows the DIM definition to be located in a separate location from the DID (for example in a separate file).

EXAMPLE

```
…
<Component id="dim_05">
    <Descriptor>
        <Statement mimeType="text/xml">
            <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIM</dip:Label>
        </Statement>
    </Descriptor>
    <Descriptor>
        <Statement mimeType="text/xml">
            <dip:MethodInfo>
                <dip:Argument>urn:foo:bar</dip:Argument>
            </dip:MethodInfo>
        </Statement>
    </Descriptor>
    <Resource mimeType="application/mp21-method"
              ref="foop://some.where/dims/05"/>
</Component>
…
```

NOTE A file containing a DIM definition can contain more than one DIM definition provided that a URI can uniquely identify the required DIM definition.

5.3.5 Object Map

5.3.5.1 Introduction

An Object in a DID is associated with an Object Type using an Object Type descriptor. An Object Type descriptor is represented by a DIP `ObjectType` element contained in a DIDL DESCRIPTOR-STATEMENT. Any DIDL element that may contain a DESCRIPTOR may be associated with an Object Type. A single Object may be associated with zero or more Object Types. An Object Type descriptor is required for each Object Type associated with the Object.

An Object Type is associated with an argument of a DIM definition by a DIM arguments descriptor of the DIM declaration (see 5.3.3). An argument of a DIM shall be associated with only a single Object Type.

The Object Map is a map consisting of 2 parts (including their inter-relationship). The first part is a list of DID Objects (i.e., DIDL elements in a DIDL document that have a DESCRIPTOR-STATEMENT construct containing a DIP `ObjectType` element). The second part is a list of DIMs (that have zero or more DIM Arguments). When the value of an `ObjectType` element of a DID Object, and the value of the `Argument` element of a DIM are equal, then the DIM can be applied to that DID Object. Given a DIDL document, the Object Map describes these relationships between the DID Objects and the DIMs.

Thus the Object Map provides the link between the arguments of a DIM associated with a given Object Type, and the actual Objects in the DID that are associated with the same Object Type. An Object shall be allowed to be utilized as an argument to a DIM only if the Object is associated with the same Object Type as the argument.

Conceptually, an Object can be seen as a DIDL element that is mapped by the Object Map to an Object Type. The `MethodInfo` of the DIM declaration then describes what Object Type is associated with each argument of a DIM (see 5.3.3). This then determines which Objects (that is which DIDL elements that map to the required Object Type) may be used as actual arguments when the DIM is invoked.

EXAMPLE 1 In a DI representing a digital music album, one or more ITEM elements in the DID can be mapped to a "music track" Object Type. A "play track" DIM that plays a track of the music album can be authored such that it accepts an argument of Object Type "music track". Any ITEM element assigned the Object Type "music track" can be utilized as an argument to the "play track" DIM.

EXAMPLE 2 The figure below depicts the Object Map of a DID, called "sample", consisting of 3 Objects and 2 DIMs. The left hand side and right hand side of the figure depict two different views of this same DID. On the left hand side, each Object of the DID "sample" can be associated with an Object Type by using the DIP `ObjectType` element (see below) contained in a DIDL DESCRIPTOR-STATEMENT. The Object Type value of Object 1 and Object 2 is "`urn:foo:bar:001`"; Object 3 has Object Type "`urn:foo:bar:002`". On the right hand side, the arguments to the DIMs contained in the DID "sample" are declared by a DIP `Argument` element (see 5.3.3). The Argument values of DIM 1 and DIM 2 are "`urn:foo:bar:001`" and "`urn:foo:bar:002`", respectively. The centre of the figure depicts the Object Map which determines, via the Object Type, which Objects in the DID can be used as arguments to the DIMs in the DID. When the value of an `ObjectType` element of an Object, and the value of the `Argument` of a DIM are equal, then the DIM can be applied to that Object. From the left hand side view, Object 1 and Object 2 can be used as the Argument to DIM 1, while Object 3 can be used as the Argument to DIM 2. From the right hand side view, DIM 1 can be applied to Object 1 and Object 2; while DIM 2 can be applied to Object 3.

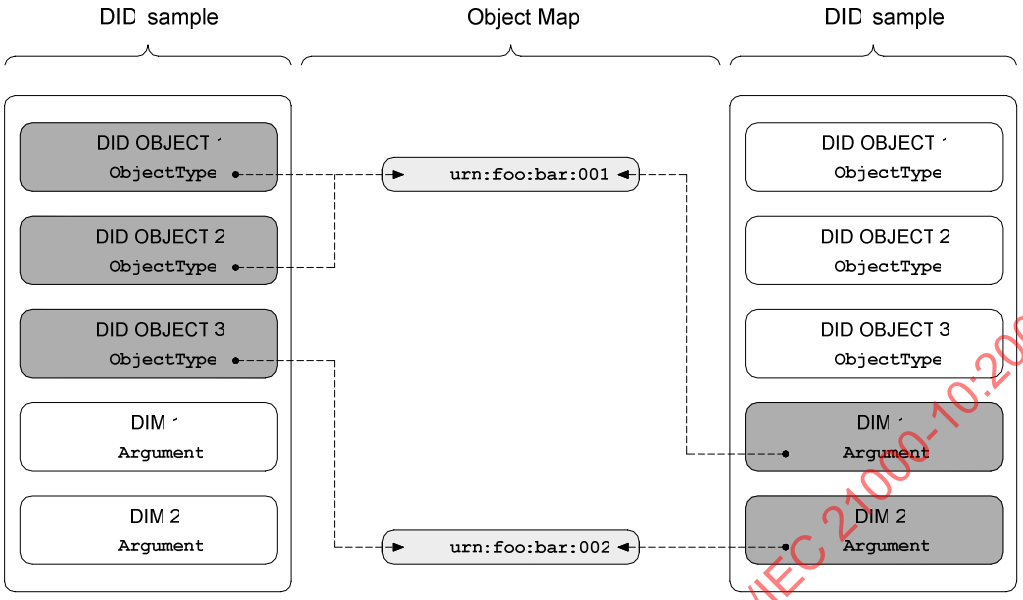


Figure 2 — Conceptual overview of Object Map

5.3.5.2 ObjectType syntax

```
<!--

################################################
# Definition of ObjectType                     #
################################################
-->
<element name="ObjectType" type="dip:ObjectTypeType"/>
<complexType name="ObjectTypeType">
    <simpleContent>
        <extension base="anyURI"/>
    </simpleContent>
</complexType>
```

5.3.5.3 ObjectType Semantics

Semantics of `ObjectType`:

Name	*Definition*
`ObjectType`	Content of this element is a URI (IETF RFC 3986) indicating the Object Type associated with the parent DIDL element of the DIDL DESCRIPTOR containing the DIP `ObjectType` element.

5.3.5.4 ObjectType examples

The following example shows the use of `ObjectType` to associate several different DIDL elements with one or more Object Types.

EXAMPLE

```
…
<Item id="track1">
    <Descriptor>
        <Statement mimeType="text/xml">
            <dip:ObjectType>urn:foo:MusicTrack</dip:ObjectType>
        </Statement>
    </Descriptor>
    <Choice>
        <Selection select_id="audiobitrate_192k"/>
        <Selection select_id="audiobitrate_64k"/>
    </Choice>
    <Component id="track1audio_192k">
        <Condition require="audiobitrate_192k"/>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:ObjectType>urn:foo:Audio</dip:ObjectType>
            </Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:ObjectType>urn:foo:Audio192k</dip:ObjectType>
            </Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:ObjectType>urn:foo:AudioMP3</dip:ObjectType>
            </Statement>
        </Descriptor>
        <Resource mimeType="audio/mpeg" ref="mysong192kbps.mp3"/>
    </Component>
    <Component id="track1audio_64k">
        <Condition require="audiobitrate_64k"/>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:ObjectType>urn:foo:Audio</dip:ObjectType>
            </Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:ObjectType>urn:foo:Audio64k</dip:ObjectType>
            </Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:ObjectType>urn:foo:AudioMP3</dip:ObjectType>
            </Statement>
        </Descriptor>
        <Resource mimeType="audio/mpeg" ref="mysong64kbps.mp3"/>
    </Component>
    <Component id="track1score">
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:ObjectType>urn:foo:MusicScore</dip:ObjectType>
            </Statement>
        </Descriptor>
        <Resource mimeType="application/pdf" ref="mysongScore.pdf"/>
    </Component>
</Item>
…
```

5.4 Digital Item Base Operations

5.4.1 Introduction

The Digital Item Base Operations (DIBOs) are the functional building blocks utilized by a Digital Item Method (DIM). They can be considered somewhat analogous to the standard library of functions of a programming language.

Subclause 5.4 specifies the set of normatively defined DIBOs. The DIBOs are the functions (of the global and local objects) from which a DIM is built and they provide the interface between the DIM and the DIBO functionality provided by a DIP engine.

A DIBO is described by

— a normatively defined interface; and

— normatively defined semantics.

The interface specifies the DIBO name, the parameters of the DIBO, and whether the DIBO returns a value. The semantics of the DIBO specify the functionality of the DIBO, in addition to the semantics of the parameters and the return value, if any.

DIBO parameters may include

— DID Objects representing DIDL elements mapped to Object Types, and

— parameters bound to a type provided by the Digital Item Method Language described in 5.2.

NOTE 1 Types provided by DIML include standard ECMAScript types such as primitive values (e.g., Boolean, Number, or String values) and native objects (e.g., String, Number, Boolean and Array objects).

While a DIBO has a normative interface and normative semantics, this does not mean that DIBOs shall be implemented in a normative way. The DIBO implementation is left to the implementer of the DIBO. The DIBO implementation can thus be viewed as being the decision of the DIBO implementer as to how they have decided to provide the functionality defined by the DIBO semantics (with the functionality accessed from a DIM via the normative DIBO interface).

EXAMPLE The `play` DIBO (see 5.4.2.7.8) requests the playing of a *component* or *descriptor*, but implementers of the DIBO are able to play the specified DID Model entity in a manner of their choosing.

The DIBO implementer shall ensure that rights, if present are evaluated and enforced. This rights related functionality is not intended to be implemented within Digital Item Methods (by the DIM author), but instead shall be implemented by the DIBO implementer. Hence it is normative behaviour of the DIBO implementations that they execute according to the rights that are associated with the Digital Item under consideration.

Exceptions listed in the interface for each DIBO are those related specifically to the DIBO. Additional exceptions, such as runtime exceptions, may also be generated.

Most of the DIBOs described in this clause fall into one of the following categories.

— Operations that access and manipulate the DID at the DIDL level;

EXAMPLE DIBOs that add child nodes, remove child nodes, modify element attributes, etc. The interface and semantics for these operations are provided by including the DOM Level 3 Core API in DIML.

— Operations that load and serialize DID instance documents. The interface and semantics for these operations are provided by including the DOM Level 3 Load and Save API in DIML; and

— Operations related to particular parts of ISO/IEC 21000.

EXAMPLE DID related operations dealing with the state of DIDL CHOICE, SELECTION, and CONDITION elements. The interface and semantics for these operations are defined in following subclauses of this part of ISO/IEC 21000.

In addition Digital Item eXtension Operations, described in 5.6, enable extended functionality that cannot easily be realized in a DIM by making use of DIBOs in the above categories.

NOTE 2 The DIBOs defined in subclauses 5.4.2.4 to 5.4.2.8 are DIBO properties of the global objects specified in 5.4.2.1.

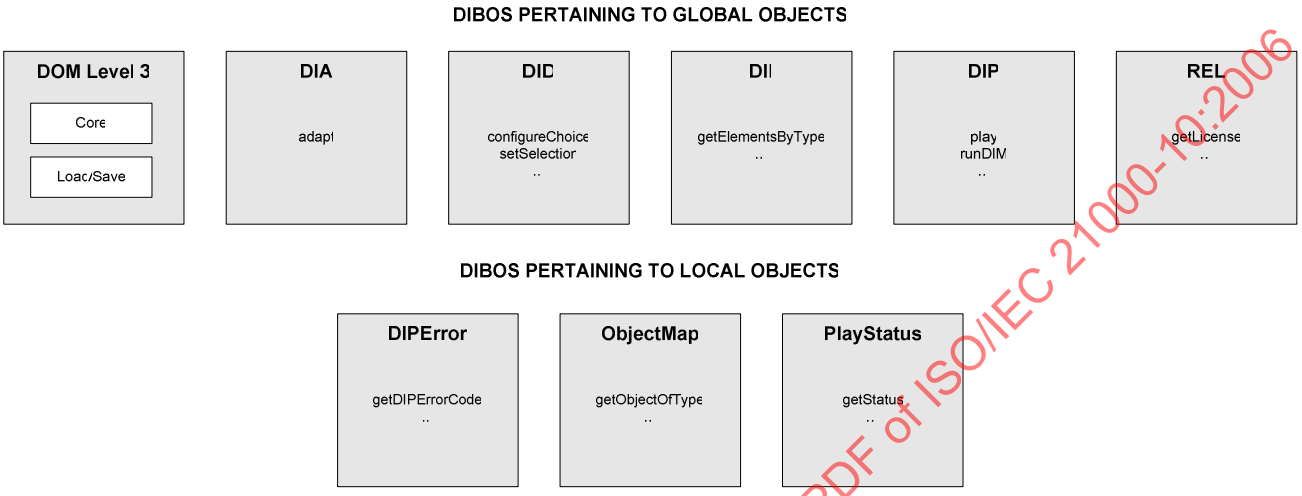


Figure 3 — Operations available to DIMs

5.4.2 Global objects, DIBOs and constants

5.4.2.1 Introduction

The DIML specification includes a normative set of DIP specific host-defined properties of the global object (see 15.1 of ISO/IEC 16262:2002). These DIP specific global properties include value properties and object properties, and are listed below. Unless otherwise stated these properties have the attributes { ReadOnly, DontDelete }.

Table 2 —DIML global values and objects

Value Properties:	`MSG_INFO` This global property is a number value (1) and may be used as a value for the `messageType` parameter of the `alert` DIBO to indicate the message is of a general informational nature. This property also has the attribute { DontEnum }.
	`MSG_WARNING` This global property is a number value (2) and may be used as a value for the `messageType` parameter of the `alert` DIBO to indicate the message is providing a warning. This property also has the attribute { DontEnum }.
	`MSG_ERROR` This global property is a number value (3) and may be used as a value for the `messageType` parameter of the `alert` DIBO to indicate the message is indicating some error condition has occurred. This property also has the attribute { DontEnum }.

	`MSG_PLAIN` This global property is a number value (4) and may be used as a value for the `messageType` parameter of the `alert` DIBO to indicate the message is generic in nature. This property also has the attribute { DontEnum }.
Object Properties:	`didDocument` This global object property is a DOM `Document` object representing the current DID instance document. This is the DID containing (directly in the DID or by inclusion) the DIM declaration that is currently executing. In the case where a series of DIMs are in the execution stack (via one or more calls to the RunDIM DIBO) it is the DID containing the DIM declaration of the first DIM on the execution stack. NOTE This is analogous to the global `document` object specified in client-side JavaScript in web browsers.
	`DIA` This global property is an object that contains DIBOs providing DIA related functionality (see 5.4.2.4).
	`DID` This global object property is an object that contains DIBOs providing DID related functionality (see 5.4.2.5).
	`DII` This global property is an object that contains DIBOs providing DII related functionality (see 5.4.2.6).
	`DIP` This global object property is an object that contains DIBOs providing DIP related functionality (see 5.4.2.7).
	`REL` This global object property is an object that contains DIBOs providing REL related functionality (see 5.4.2.8).

5.4.2.2 DIDL document access and manipulation

The hierarchical DID Model defined in part 2 of ISO/IEC 21000 reflects the structured nature of a Digital Item. This hierarchical DID Model is preserved by the hierarchical XML based representation (DIDL) of a DID also defined in part 2. The Document Object Model (DOM) defined by W3C is designed for the accessing and manipulation of such hierarchical documents.

Hence for Digital Item Methods the syntax and semantics of base operations for accessing and manipulating the DIDL document elements are those specified by the DOM Level 3 Core API as defined by W3C REC-DOM-Level-3-Core-20040407.

Since DIML is based on ECMAScript, the ECMAScript bindings of the DOM Level 3 Core API are included as part of the DIML specification.

5.4.2.3 DIDL document loading and saving

The syntax and semantics for operations for loading and saving a DIDL document are those specified by the DOM Level 3 Load and Save API as defined by W3C REC-DOM-Level-3-LS-20040407.

Since DIML is based on ECMAScript, the ECMAScript bindings of the DOM Level 3 Load and Save API are included as part of the DIML specification.

5.4.2.4 DIA related operations

5.4.2.4.1 Introduction

These DIBOs provide access to DIA related functionality. The DIBOs are provided as function properties of a `DIA` object. DIML includes such a `DIA` object as a property of the global object (see 5.4.2.1).

EXAMPLE Calling a DIA DIBO

```
function foo(arg1)
{
    …
    DIA.adapt( component, metadata );
    …
}
```

5.4.2.4.2 adapt

5.4.2.4.2.1 Interface

Syntax:	`adapt(element, metadata)`
Description:	Adapts a specified COMPONENT or DESCRIPTOR element.
Parameters:	`element` A DOM `Element` object representing the COMPONENT, or DESCRIPTOR element to be adapted.
	`metadata` Null or an array of DOM `Element` objects representing additional information that can be considered when adapting the element.
Return value:	A DOM `Element` object representing the adapted DIDL element or null if the element was not adapted.
Exceptions:	`DIPError` With DIP error code — `ADAPTATION_FAILED` if an error occurs during adaptation of the resource; or — `INVALID_PARAMETER` if — `element` is not a COMPONENT, or DESCRIPTOR; or — or if `metadata` is not null nor an array of DOM `Element` objects.

5.4.2.4.2.2 Semantics

This DIBO allows a DIM author to explicitly request an adaptation of the DIDL element represented by the `element` parameter.

The `metadata` parameter is an array of `Element` objects representing additional information the DIM author suggests can be considered when adapting the resource. The `metadata` parameter may be null if the DIM author does not wish to provide any such suggestions. If the metadata parameter is not null and is not an array of DOM `Element` objects then an invalid parameter exception is generated.

If an adaptation of the element does take place and is successful, then an `Element` object representing the DIDL element for the adapted element is returned. This can then be utilized as a parameter to other appropriate DIBOs. The original element remains unchanged.

If an adaptation of the element is attempted but an error occurs during the adaptation, then a `DIPError` is generated.

If no adaptation of the element is attempted, then null is returned.

NOTE 1 An adaptation might not be attempted if, for example, adaptation is not supported, or the available metadata for adaptation is not supported.

NOTE 2 For an implementer of this DIBO, the available metadata to guide the adaptation can be sourced from the `metadata` parameter provided by the DIM author in the call to `adapt`, or else directly accessed from the DID from within the DIBO implementation. For example, if the DIBO is implemented in a Java environment that has standard DOM access to the DID, the DIBO implementer can directly inspect the DESCRIPTORS, if any, associated with the element for hints on adaptation. Also, metadata to guide the adaptation is not limited to the `metadata` parameter or metadata within the DID. For example metadata from a preferences or configuration file stored separately from the DID could be utilized, if the DIBO implementer has access to such information.

NOTE 3 The adaptation need not be implemented directly within the DIBO. For example, an implementer of this DIBO might choose to send a request along with relevant information to a remote server to do the adaptation.

NOTE 4 This DIBO allows the DIM author to explicitly request an adaptation. An adaptation of an element might still occur when the element is, for example, passed to the `play` DIBO. This would be a decision of the `play` DIBO implementer. In addition, such adaptations might still occur when elements are accessed without any interactions with the DI using DIP.

NOTE 5 ISO/IEC 21000-7 provides the following tools related to resource adaptation which an implementer of this DIBO can consider supporting.

— Usage Environment Descriptions;

— Bitstream Syntax Description based adaptation;

— Terminal and network quality of service;

— Usage Constraints Descriptions; and

— DIA Configuration.

5.4.2.4.2.3 Example use

EXAMPLE This DIBO would be required if a DIM author wanted to author a DIM that created a derived Digital Item containing a single resource created by adapting a resource from the original DI such that the resource in the derived DI is “pre-adapted” to the current environment.

5.4.2.5 DID related operations

5.4.2.5.1 Introduction

These DIBOs provide access to DID related functionality. The DIBOs are provided as function properties of a `DID` object. DIML includes such a `DID` object as a property of the global object (see 5.4.2.1).

EXAMPLE Calling a DID DIBO

```
function foo(arg1)
{
    …
    choice = didDocument.getElementByID( "bitrate_choice" );
    DID. configureChoice( choice );
    …
}
```

5.4.2.5.2 areConditionsSatisfied

5.4.2.5.2.1 Interface

Syntax:	`areConditionsSatisfied(element)`
Description:	Tests whether conditions for a specified DIDL element are satisfied.
Parameters:	`element` A DOM `Element` object representing the DIDL element for which conditions will be tested.
Return value:	A boolean value. A value of `true` is returned if the conditions are satisfied. A value of `false` is returned if the conditions are not satisfied or the state of the conditions cannot be resolved.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAMETER` if `element` is not a DIDL element.

5.4.2.5.2.2 Semantics

This DIBO tests whether the conditions for a specified DIDL element are satisfied.

The `element` parameter specifies the DIDL element whose conditions are to be tested. It is an error to invoke this DIBO without specifying a DIDL element by the `element` parameter, in which case an invalid parameter exception is generated.

The return value is a `boolean` value indicating whether the conditions for the specified DIDL element are satisfied.

The conditions for the specified DIDL element are those described by any CONDITION child elements (as specified in ISO/IEC 21000-2) of the specified element. For the conditions of the specified element to be satisfied the rules for resolving CONDITION child elements as specified in ISO/IEC 21000-2 shall be evaluated.

EXAMPLE Multiple CONDITION child elements are evaluated in an “or” fashion based on the resolved state of each individual CONDITION.

If the specified DIDL element contains no child CONDITION elements, the conditions for the specified element are considered to be satisfied in context of this DIBO and a `true` value is returned.

In the context of this DIBO, the inability to resolve a CONDITION is interpreted as the condition being not satisfied. If all child CONDITION elements cannot be resolved or are false, a `false` value is returned.

If the specified DIDL element is a CONDITION element, then the predicates of the CONDITION are evaluated and the resulting value returned. If the CONDITION predicates cannot be resolved, `false` is returned.

NOTE The DIP engine usually will need to maintain its own internal state reflecting the state of the CHOICES and SELECTIONS of a DID. See also `configureChoice` (5.4.2.5.3) and `setSelection` (5.4.2.5.4).

5.4.2.5.2.3 Example usage

EXAMPLE An example where this DIBO can be used is at steps j), p), and u) in the scenario described in I.2.2.

5.4.2.5.3 configureChoice

5.4.2.5.3.1 Interface

Syntax:	`configureChoice(choice)`
Description:	Requests the User to configure a CHOICE element in the DID.
Parameters:	`choice` A DOM `Element` object representing the CHOICE element to be configured.
Return value:	Returns a boolean value. A value of `true` indicates the CHOICE configuration was modified, a value of `false` indicates the CHOICE configuration was not modified.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAMETER` if `choice` is not a CHOICE.

5.4.2.5.3.2 Semantics

This DIBO requests the User to configure a specified DIDL CHOICE element.

The `choice` parameter specifies the DIDL CHOICE element to be configured by the User. It is an error to invoke this DIBO if the `choice` parameter does not specify a CHOICE element, in which case an invalid parameter exception is generated.

The DIBO allows the User to configure the CHOICE in compliance with ISO/IEC 21000-2 by making decisions about the *predicates* embodied by the SELECTION child elements of the CHOICE. If a SELECTION is chosen, its *predicate* becomes true, if it is rejected, its *predicate* becomes false, if it is unresolved, its *predicate* is undecided.

The `boolean` return value indicates whether the CHOICE configuration was modified or not.

NOTE 1 The manner of configuring the CHOICE is a DIBO implementation decision. For example, one DIBO implementation might present the CHOICE in a GUI to a human allowing the human to select or deselect SELECTION elements. Another implementation might be intended to process a DI without human intervention and implement some form of automated CHOICE configuration.

NOTE 2 The DIP engine will usually need to maintain its own internal state reflecting the state of the CHOICES and SELECTIONS of a DID. See also `setSelection` (5.4.2.5.4) and `areConditionsSatisfied` (5.4.2.5.2).

5.4.2.5.3.3 Example use

EXAMPLE An example where this DIBO can be used is at step m) in the scenario described in I.2.2

5.4.2.5.4 setSelection

5.4.2.5.4.1 Interface

Syntax:	`setSelection(selection, state)`
Description:	Sets the state of a given SELECTION element.
Parameters:	`selection` A DOM `Element` object representing the SELECTION element whose state is to be set.
	`state` A string value specifying the state to set the SELECTION element. Valid values are `true`, `false`, or `undecided`.
Return value:	None.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAMETER` if — `selection` is not a SELECTION; or — `state` is not a string with value `true`, `false`, or `undecided`.

5.4.2.5.4.2 Semantics

This DIBO sets the state of a specified DIDL SELECTION element as maintained by the DIP engine.

The `selection` parameter specifies the DIDL SELECTION element whose state is to be set. It is an error to invoke this DIBO if the `selection` parameter does not specify a SELECTION element, in which case an invalid parameter exception is generated.

The `state` parameter specifies the value for the state of the SELECTION. This state is maintained by the DIP engine for subsequent reference in other DIBOs that rely on the configuration state of SELECTION elements (for example, `areConditionsSatisfied` 5.4.2.5.2). The valid values for the `state` parameter correspond to those SELECTION states described in ISO/IEC 21000-2, which are: `true`, `false`, `undecided`. It is an error to invoke this DIBO with any other value for the `state` parameter, in which case an invalid parameter exception is generated.

NOTE The DIP engine will usually need to maintain its own internal state reflecting the state of the CHOICES and SELECTIONS of a DID. See also `configureChoice` (5.4.2.5.3) and `areConditionsSatisfied` (5.4.2.5.2).

5.4.2.5.4.3 Example use

EXAMPLE An example where this DIBO can be used is when a DIM utilizes some input data to determine the state of a SELECTION, and then needs to call this DIBO to set the state of the SELECTION within the DIP engine execution environment.

5.4.2.6 DII related operations

5.4.2.6.1 Introduction

These DIBOs provide access to DII related functionality. The DIBOs are provided as function properties of a `DII` object. DIML includes such a `DII` object as a property of the global object (see 5.4.2.1).

EXAMPLE Calling a DII DIBO

```
function foo(arg1)
{
    …
    elements = DII.getElementsByIdentifier( null, "urn:acme:foo:bar123" );
    …
}
```

5.4.2.6.2 getElementsByIdentifier

5.4.2.6.2.1 Interface

Syntax:	`getElementsByIdentifier(sourceDID, value)`
Description:	Retrieves from the source DID existing DIDL elements that have a descriptor containing a DII Identifier with the specified value.
Parameters:	`sourceDID` The DOM `Document` object representing the source DID from which the DIDL elements are to be retrieved.
	`value` A string value specifying the value of the DII Identifier identifying the DIDL elements to be retrieved.
Return value:	An array of DOM `Element` objects representing the retrieved DIDL elements. May be empty if no DIDL elements matching the specified criteria are retrieved.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAMETER` — if `sourceDID` is not a DOM `Document` representing a DIDL instance document; or — `value` is not a string value.

5.4.2.6.2.2 Semantics

This DIBO retrieves from the specified DID existing DIDL elements based on a DII Identifier.

The `sourceDID` parameter is a DOM `Document` object representing the source DIDL document from which the elements are to be retrieved. It is an error to invoke this DIBO if the `sourceDID` parameter is not a DOM `Document` object, in which case an invalid parameter exception is generated.

The `value` parameter specifies the value of the DII Identifier required to identify the DIDL elements to be retrieved. The syntax and semantics of DII Identifier are specified in part 3 of ISO/IEC 21000. It is an error to invoke this DIBO if the `value` parameter is not a string value, in which case an invalid parameter exception is generated.

The retrieved elements are available to the invoking DIM via the returned array of `Element` objects. The elements can then be utilized as a parameter to other DIBOs. Only DIDL elements shall be returned. If no DIDL elements are identified by the specified criteria an empty array is returned.

5.4.2.6.2.3 Example use

EXAMPLE An example where this DIBO can be used is when a DIM author might not know where an element is located in the structure of a DID, but can determine a DII Identifier for the element. In this case the DIM author can utilize the `getElementsByIdentifier` DIBO to retrieve the element.

5.4.2.6.3 getElementsByRelatedIdentifier

5.4.2.6.3.1 Interface

Syntax:	`getElementsByRelatedIdentifier(sourceDID, value)`
Description:	Retrieves from the source DID existing DIDL elements that have a descriptor containing a DII RelatedIdentifier with the specified value.
Parameters:	`sourceDID` The DOM `Document` object representing the source DID from which the DIDL elements are to be retrieved.
	`value` A string value specifying the value of the DII RelatedIdentifier associated with the DIDL elements to be retrieved.
Return value:	An array of DOM `Element` objects representing the retrieved DIDL elements. May be empty if no DIDL elements matching the specified criteria are retrieved.
Exceptions:	DIPError With DIP error code `INVALID_PARAMETER` — if `sourceDID` is not a DOM `Document` representing a DIDL instance document; or — `value` is not a string value.

5.4.2.6.3.2 Semantics

This DIBO retrieves from the specified DID existing DIDL elements based on a DII RelatedIdentifier.

The `sourceDID` parameter is a DOM `Document` object representing the source DIDL document from which the elements are to be retrieved. It is an error to invoke this DIBO if the `sourceDID` parameter is not a DOM `Document` object, in which case an invalid parameter exception is generated.

The `value` parameter specifies the value of the DII RelatedIdentifier associated with the DIDL elements to be retrieved. The syntax and semantics of DII RelatedIdentifier are specified in part 3 of ISO/IEC 21000. It is an error to invoke this DIBO if the `value` parameter is not a string value, in which case an invalid parameter exception is generated.

The retrieved elements are available to the invoking DIM via the returned array of `Element` objects. The elements can then be utilized as a parameter to other DIBOs. Only DIDL elements shall be returned. If no DIDL elements are identified by the specified criteria an empty array is returned.

5.4.2.6.3.3 Example use

EXAMPLE This DIBO can be used when a DIM author has an identifier for some information and wants to retrieve any elements in the DID that identify that information as related information. For example, consider a Digital Item representing a collection of numerous musical pieces, with different sub-items for audio recordings, musical score, lyrics, and music videos, some of which might or might not be present for any individual musical piece. Given the identifier for a musical piece, the DIM author might want to retrieve in a single call all of the related sub-items. In this case the DIM author can utilize the `getElementsByRelatedIdentifier` DIBO to retrieve those elements (provided they utilize a DII RelatedIdentifier to identify the musical piece as being related).

5.4.2.6.4 getElementsByType

5.4.2.6.4.1 Interface

Syntax:	`getElementsByType(sourceDID, value)`
Description:	Retrieves from the source DID existing DIDL ITEM elements that have a descriptor containing a DII Type with the specified value.
Parameters:	`sourceDID` The DOM `Document` object representing the source DID from which the DIDL elements are to be retrieved.
	`value` A string value specifying the value of the DII Type of the DIDL ITEM elements to be retrieved.
Return value:	An array of DOM `Element` objects representing the retrieved DIDL ITEM elements. May be empty if no DIDL ITEM elements matching the specified criteria are retrieved.
Exceptions:	DIPError With DIP error code `INVALID_PARAMETER` — if `sourceDID` is not a DOM `Document` representing a DIDL instance document; or — `value` is not a string value.

5.4.2.6.4.2 Semantics

This DIBO retrieves from the specified DID existing DIDL elements based on a DII Type.

The `sourceDID` parameter is a DOM `Document` object representing the source DIDL document from which the elements are to be retrieved. It is an error to invoke this DIBO if the `sourceDID` parameter is not a DOM `Document` object, in which case an invalid parameter exception is generated.

The `value` parameter specifies the value indicating the DII Type of the DIDL ITEM elements to be retrieved. The syntax and semantics of DII Type are specified in part 3 of ISO/IEC 21000. It is an error to invoke this DIBO if the `value` parameter is not a string value, in which case an invalid parameter exception is generated.

The retrieved elements are available to the invoking DIM via the returned array of `Element` objects. The elements can then be utilized as a parameter to other DIBOs. Only DIDL ITEM elements shall be returned. If no DIDL ITEM elements are identified by the specified criteria an empty array is returned.

5.4.2.6.4.3 Example use

EXAMPLE An example where this DIBO can be used is when Digital Item is used as a container of several sub-items of different DII Types. In this case the DIM author can utilize the `getElementsByType` DIBO to retrieve all ITEM elements representing sub-items of a given DII Type.

5.4.2.7 DIP related operations

5.4.2.7.1 Introduction

These DIBOs provide access to DIP related functionality. The DIBOs are provided as function properties of a `DIP` object. DIML includes such a `DIP` object as a property of the global object (see 5.4.2.1).

EXAMPLE Calling a DIP DIBO

```
function foo(arg1)
{
    …
    DIP.Play( component, false );
    …
}
```

5.4.2.7.2 alert

5.4.2.7.2.1 Interface

Syntax:	`alert(message, messageType)`
Description:	Alerts the User with a message.
Parameters:	`message` A string value containing the message to be displayed.
	`messageType` A number value indicating the generic nature of the message. Valid values are `MSG_INFO`, `MSG_WARNING`, `MSG_ERROR` and `MSG_PLAIN` which are defined as value properties of the DIML global object (see 5.4.2.1).

Return value:	None
Exceptions:	DIPError With DIP error code `INVALID_PARAM` — If `message` is not a string value; or — if `messageType` does not specify a valid value.

5.4.2.7.2.2 Semantics

This DIBO provides simple textual feedback to the User.

EXAMPLE To notify the User of an error condition.

The textual message is specified by the `message` parameter. It is an error to invoke this DIBO if the message parameter is not a string value, in which case an invalid parameter exception is generated.

The `messageType` parameter indicates the nature of the message. It is an error to invoke this DIBO if the `messageType` parameter is not a valid number value, in which case an invalid parameter exception is generated (a valid number value is one of the values `MSG_INFO`, `MSG_WARNING`, `MSG_ERROR`, or `MSG_PLAIN` defined as number properties of the DIML global object).

NOTE The implementation of the alert is a DIBO implementation decision. For example, one DIBO implementation might present the message in a GUI to a human user. Another implementation might be intended to process a DI without human intervention and log the message to a log file.

5.4.2.7.2.3 Example use

EXAMPLE An example where this DIBO can be used is at step z) in the scenario described in I.2.2.

5.4.2.7.3 execute

5.4.2.7.3.1 Interface

Syntax:	`execute(element)`
Description:	Executes the resource associated with a specified COMPONENT, or DESCRIPTOR.
Parameters:	`element` A DOM `Element` object representing the COMPONENT, or DESCRIPTOR element containing the executable resource.
Return value:	A boolean value of `true` if the resource execution was successfully initiated, or `false` if resource execution was not initiated.

Exceptions:	`DIPError` With DIP error code — `INVALID_PARAMETER` if — `element` is not an COMPONENT, or DESCRIPTOR; or — more than one executable digital resource is found to be associated with the specified element; or — `EXECUTE_FAILED` if an error occurs during the execution.

5.4.2.7.3.2 Semantics

This DIBO causes execution to be initiated for the executable resource associated with the DIDL element represented by the `element` parameter.

The `element` parameter shall be a DOM `Element` object representing a COMPONENT or DESCRIPTOR containing the executable digital resource to be executed. It is an error to invoke this DIBO if the `element` parameter is not a DOM `Element` object representing a COMPONENT or DESCRIPTOR, in which case an invalid parameter exception is generated.

The manner of executing the associated resource, appropriate to its media type, is left as an implementation choice of the DIBO implementer.

NOTE 1 It is the responsibility of the DIBO implementation and DIP engine to ensure security related issues are addressed when executing executable resources.

NOTE 2 When determining the associated executable digital resource, the case where there is a COMPONENT with multiple RESOURCE child elements still represents a single *resource*.

5.4.2.7.3.3 Example use

EXAMPLE An example where this DIBO can be used is when a DI author might want to include as a resource in a Digital Item an executable program that is capable of presenting other resources in the Digital Item that are in a proprietary format understood only by the executable resource and that utilizes descriptors attached to the resources containing proprietary metadata to determine the presentation.

5.4.2.7.4 getExternalData

5.4.2.7.4.1 Interface

Syntax:	`getExternalData(mimeTypes, requestMessages)`
Description:	Requests the User to select resources located external to the DI.
Parameters:	`mimeTypes` An array of arrays of string values specifying the media formats required.
	`requestMessages` An array whose elements are either string values or `null`.

Return value:	Returns an array whose elements are string values specifying the URLs that describe the location of the resources.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAMETER` — if `mimeTypes` is not an array of arrays of string values, or — if `requestMessages` is not null and is not an array the same size as `mimeTypes` with each array member being a string value or null.

5.4.2.7.4.2 Semantics

This DIBO requests the User to select resources external to the current DID.

EXAMPLE An, audio or video resource.

The `mimeTypes` parameter specifies allowable MIME types (IETF RFC 2046) of the resources to be selected. It is an array with an element for each resource which is to be selected. Each element is another array of string values. The string values in one array specify the allowed MIME types for a single resource for which the User is to select.

The MIME type values are given in the form `type/subtype`. The values for the `type` and `subtype` of the MIME type should be valid values as specified by IETF RFC 2046.

EXAMPLE A value of `image/jpeg` specifies a JPEG image is allowed to be selected.

It is an error to invoke this DIBO if the `mimeTypes` parameter is not an array where each array entry is itself an array of string values, in which case an invalid parameter exception is generated.

The `requestMessages` parameter is an array of string values that contains a request message for each of the resource selections indicated by the members of the `mimeTypes` array. The `requestMessages` parameter may be null, in which case no explicit request messages are specified. If the `requestMessages` parameter is not null, it is an error if the number of members in the `requestMessages` parameter is not equal to the number of members in the `mimeTypes` parameter. However one or more of the members of the `requestMessages` parameter may be null. It is an error if a member of the `requestMessages` parameter is not null and is not a string value.

The selected resources are identified to the invoking DIM by the URLs returned by the DIBO. A null return value indicates that no resource was selected.

NOTE The presentation and selection of the resource is a DIBO implementation decision. For example, one DIBO implementation might present the resource selection in a GUI to a human allowing the human to select the resource. Another implementation might be intended to process a DI without human intervention and implement some form of automated resource selection.

5.4.2.7.4.3 Example use

EXAMPLE An example where this DIBO can be used is when a RESOURCE element is created and a DIM needs to set the ref attribute and requests the User to locate a resource to which the ref attribute will refer.

5.4.2.7.5 getObjectMap

5.4.2.7.5.1 Interface

Syntax:	`getObjectMap(document)`
Description:	Retrieves the Object Map from a DID instance document.
Parameters:	`document` A DOM `Document` representing the DID instance document containing the Object Type information.
Return value:	An `ObjectMap` object (see 5.4.3.3) representing the Object Map (see 5.3.5) of the DID instance document.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAMETER` if `document` is not a DOM `Document` object representing a DIDL instance document.

5.4.2.7.5.2 Semantics

This DIBO allows the retrieval of an Object Map (see 5.4.3.3 and 5.3.5) from a DID instance document.

The `document` parameter shall be a DOM `Document` object representing the DIDL instance document with the Object Type information needed to construct the Object Map. The Object Map is represented in DIML by an `ObjectMap` object (see 5.4.3.3). It is an error to invoke this DIBO if the `document` parameter is not a DOM `Document` object representing a DIDL document, in which case an invalid parameter exception is generated.

5.4.2.7.5.3 Example use

EXAMPLE An example where this DIBO can be used is if a DIM author wants to process all DID Objects of a particular Object Type. For example, in a music album Digital Item, to get all Objects associated with a music track Object Type, the DIM author can use the `getObjects` operation of the `ObjectMap` object. Prior to this, the DIM author would need to call the `getObjectMap` DIBO to first retrieve the `ObjectMap`.

5.4.2.7.6 getObjects

5.4.2.7.6.1 Interface

Syntax:	`getObjects(objectTypes, requestMessages)`
Description:	Provides the User with one or more selections of Objects of given Object Types. The Objects for each selection are those elements identified in the Object Map contained in the DI to be of the given Object Types (see also 5.4.3.3 and 5.3).
Parameters:	`objectTypes` An `Array` of string values specifying the Object Type names.

	`requestMessages` An array whose elements are either string values or `null`.
Return value:	Returns an array whose elements are either a DOM `Element` object or `null`. A DOM `Element` object represents the selected element of the corresponding Object Type as specified in the `objectTypes` array.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAMETER` — if .`objectTypes` is not an array of string values, or — if `requestMessages` is not null and is not an array the same size as `objectTypes` with each array member being a string value or null.

5.4.2.7.6.2 Semantics

This DIBO requests the User to select DIDL elements of specified Object Types from the current DID containing or referencing the DIM from which the DIBO is invoked.

DIM authors should provide DID Objects, representing DIDL elements, to be operated on in a DIM as DIM Arguments declared by the DIM declaration (see 5.3.3). DIM authors should only use the `getObjects` DIBO when it is not possible to provide a DID Object as an Argument.

NOTE 1 Use of the `getObjects` DIBO prevents DIP engines that, for example, present the User with a list of DID Objects, and then a list of DIMs that accept those Objects as Arguments, from identifying DIMs that operate on Objects that are not specified as the Arguments of a DIM in the DIM declaration.

The `objectTypes` parameter is an array of `string` values that specifies the Object Type for each element that the User will be requested to select. The Object Type is an Object Type that is found in the Object Map (see 5.4.3.3 and 5.3) for the current DID. It is an error to invoke this DIBO if the `objectTypes` parameter is not an array of string values, in which case an invalid parameter exception is generated.

For each Object Type specified in the `objectTypes` parameter, the DIBO allows the User to select one element that maps to that Object Type.

The `requestMessages` parameter is an array of `string` values that contains a request message for each of the element selections indicated by the members of the `objectTypes` array. The `requestMessages` parameter may be null, in which case no explicit request messages are specified. If the `requestMessages` parameter is not null, it is an error if the number of members of the `requestMessages` parameter is not equal to the number of messages in the `objectTypes` parameter. However one or more of the members of the `requestMessages` parameter may be null. It is an error if a member of the `requestMessages` parameter is not null and is not a string value.

The selected elements of the specified Object Types are available to the invoking DIM via the returned array of `Element` objects. These can then be used as parameters to other DIBOs.

NOTE 2 The presentation and selection of the elements is a DIBO implementation decision. For example, one DIBO implementation might present the element selection in a GUI to a human allowing the human to select the elements. Another implementation might be intended to process a DI without human intervention and implement some form of automated element selection.

5.4.2.7.6.3 Example use

EXAMPLE An example where this DIBO can be used is at steps h) and r) in the scenario described in I.2.2

5.4.2.7.7 getValues

5.4.2.7.7.1 Interface

Syntax:	`getValues(dataTypes, requestMessages)`
Description:	Requests the User for input data of one of the primitive data types.
Parameters:	`dataTypes` An `Array` of string values specifying the data type of each datum.
	`requestMessages` An array whose elements are either string values or `null`.
Return value:	Returns an array whose elements are null or values of type boolean, string or number, corresponding to the data type specified in the `dataTypes` array.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAMETER` if — `dataTypes` contains an invalid data type (not one of `Boolean`, `String`, or `Number`); or — `requestMessages` is not null and is not an array the same size as `objectTypes` with each array member being a string value or null.

5.4.2.7.7.2 Semantics

This DIBO requests the User to enter data values of one of the primitive DIML data types.

The `dataTypes` parameter is an array of `string` values that specifies primitive data type for each data value that the User will be requested to enter. The data type shall be one of: `Boolean`, `String`, or `Number`. It is an error to specify an invalid data type, in which case an invalid parameter exception is generated.

For each data type specified in the `dataTypes` parameter, the DIBO allows the User to enter a data value of that data type. The DIBO implementation should execute some basic validation on the value entered, at least to ensure the value is of the correct data type. It is an error to invoke this DIBO if the `dataTypes` parameter is not an array of string values, in which case an invalid parameter exception is generated.

The `requestMessages` parameter is an array of `string` values that contains a request message for each of the data values indicated by the members of the `dataTypes` array. The `requestMessages` parameter may be null, in which case no explicit request messages are specified. If the `requestMessages` parameter is not null, it is an error if the number of members of the `requestMessages` parameter is not equal to the number of messages in the `dataTypes` parameter. However one or more of the members of the `requestMessages` parameter may be null. It is an error if a member of the `requestMessages` parameter is not null and is not a string value.

The User entered data values of the specified data types are available to the invoking DIM via the returned array of values. If a value was not entered for a requested datum then the corresponding entry in the returned array of values will be null. If no data values at all were entered then a null is returned by the DIBO.

NOTE The data entry mechanism for the requested data is a DIBO implementation decision. For example, one DIBO implementation might present a data entry GUI to a human allowing the human to enter the values. Another implementation might be intended to process a DI without human intervention and implement some form of automated data entry.

5.4.2.7.7.3 Example use

EXAMPLE An example where this DIBO can be used is at step t) in the scenario described in I.2.2.

5.4.2.7.8 play

5.4.2.7.8.1 Interface

Syntax:	`play(element, async)`
Description:	Plays a specified COMPONENT, or DESCRIPTOR and optionally waits for completion before returning control to the calling DIM.
Parameters:	`element` A DOM `Element` object representing the COMPONENT, or DESCRIPTOR element to be played.
	`async` A boolean value indicating if the COMPONENT, or DESCRIPTOR should be played asynchronously or not. If `true` then the element is played asynchronously and the DIBO should return control immediately to the calling DIM after playing of the element is initiated. If `false` then the element is played synchronously and control is not returned to the calling DIM until the element media end time has been reached (if applicable).
Return value:	Returns a `PlayStatus` (see 5.4.3.4) object to identify the playing element. This is included so that the playing instance of the elements can be released at a later time.
Exceptions:	`DIPError` With DIP error code — `INVALID_PARAMETER` if — `element` is not a COMPONENT, or DESCRIPTOR; or — `async` is not a boolean value; or — `PLAYBACK_FAILED` if an error occurs during playback.

5.4.2.7.8.2 Semantics

This DIBO causes the DIDL element represented by the `element` parameter to be rendered into a transient and directly perceivable representation.

The `element` parameter shall be a DOM `Element` object representing a COMPONENT or DESCRIPTOR to be played. It is an error to invoke this DIBO if the `element` parameter is not a DOM `Element` object representing a COMPONENT or DESCRIPTOR, in which case an invalid parameter exception is generated.

The manner of playing the element, appropriate to its content, is left as an implementation choice of the DIBO implementer.

The `async` parameter shall be a boolean value indicating whether the element is to be played asynchronously or synchronously. It is an error to invoke this DIBO if the `async` parameter is not a boolean value, in which case an invalid parameter exception is generated.

For an asynchronous playing, once playing of the element is initiated, the DIBO should return execution control to the invoking DIM.

For synchronous playing, the DIBO should not return execution control to the invoking DIM until the play status transitions to `STATICPLAY` or `RELEASED`.

For time based media the play status on successful playing of the element will be `TIMEPLAY`. When the media end time is reached, the status will transition to `STATICPLAY`.

For non time based media the play status on successful playing of the element will be `STATICPLAY`.

If the element could not be successfully played the play status will be `RELEASED`.

The DIBO returns a `PlayStatus` (see 5.4.3.4) object which provides a handle to the playing instance of the element and its status. This handle is used as a parameter to other DIBOs to control the status of the playing instance of the element.

NOTE 1 Time based media is media that changes over time, such as animations, audio clips, and video clips. The MIME media type of resources associated with the element can be retrieved from the MIMETYPE attribute of any DIDL RESOURCE elements. In many cases this will allow the DIBO implementation to determine whether the resource is time based or not. For example, resources with a top-level media type of `audio` or `video` (such as `audio/mpeg` for MP3) are generally time based.

NOTE 2 For a synchronous call to the `play` DIBO for time based media, the play status transition through the `TIMEPLAY` state is effectively hidden from the invoking DIM, since the `play` DIBO will not return until the play status has transitioned to `STATICPLAY`.

NOTE 3 It is the responsibility of the DIBO implementation to locate the appropriate technology for playing of the element and to inform the User if this cannot be found. For example the resource could be played internally or a mechanism to locate and use some external “third party” software to play the resource could be provided. The MIME media type of resources associated with the element can be retrieved from the MIMETYPE attribute of any DIDL RESOURCE elements. This will enable the DIBO implementation to determine how to play the associated resources.

NOTE 4 While the manner of playing the element is a DIBO implementation choice, it could be guided by additional available information, for example information contained within *descriptors* associated with the element to be played.

5.4.2.7.8.3 Example use

EXAMPLE An example where this DIBO can be used is steps f), k), p), and bb) of the use case scenario described in I.2.2

5.4.2.7.9 print

5.4.2.7.9.1 Interface

Syntax:	`print(element)`
Description:	Prints a specified COMPONENT, or DESCRIPTOR.
Parameters:	`element` A DOM `Element` object representing the COMPONENT, or RESOURCE element to be printed.
Return value:	A boolean value of `true` if the element was successfully printed, or `false` if it was not printed.
Exceptions:	`DIPError` With DIP error code — `INVALID_PARAMETER` if `element` is not a COMPONENT, or DESCRIPTOR; or — `PRINT_FAILED` if an error occurs during the printing.

5.4.2.7.9.2 Semantics

This DIBO causes the DIDL element represented by the `resource` parameter to be rendered into a fixed and directly perceivable representation.

The `element` parameter shall be a DOM `Element` object representing a COMPONENT or DESCRIPTOR to be printed. It is an error to invoke this DIBO if the `element` parameter is not a DOM `Element` object representing a COMPONENT or DESCRIPTOR, in which case an invalid parameter exception is generated.

The manner of printing the element, appropriate to its content, is left as an implementation choice of the DIBO implementer.

5.4.2.7.9.3 Example use

EXAMPLE In some cases the DIM author might want to allow the User to print a resource contained in a *component*. For example, the lyrics of a song included as a text resource in a music digital item.

5.4.2.7.10 release

5.4.2.7.10.1 Interface

Syntax:	`release(playStatus)`
Description:	Stop playback of a COMPONENT or DESCRIPTOR and release related playback state information.

Parameters:	`playStatus` The `PlayStatus` object that was generated as a return value when playback of the DIDL element began. EXAMPLE As the result of an asynchronous call to the `play` (5.4.2.7.8) DIBO.
Return value:	None.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAMETER` if `playStatus` is not a `PlayStatus`.

5.4.2.7.10.2 Semantics

This DIBO causes playing of the DIDL element associated with the specified `playStatus` to be stopped and any state information to be released.

The `playStatus` parameter shall be an object of type `PlayStatus` (see 5.4.3.4) that was returned by a call to the `play` DIBO to play the associated element which is to be stopped.

After calling this DIBO the play status will transition to `RELEASED`.

If the current status is already `RELEASED`, then this DIBO does nothing.

5.4.2.7.10.3 Example use

EXAMPLE In some cases a DIM might play a *component* asynchronously, do some other operations, and then release the *component* before exiting. For example, if a Digital Item represents an electronic travel brochure, and an ITEM contains several *components* that contain information about a specific destination, a DIM could be authored such that it starts playing some background audio asynchronously, displays synchronously the information *components*, then prior to exiting, releases the asynchronously playing *component* containing the audio resource.

5.4.2.7.11 runDIM

5.4.2.7.11.1 Interface

Syntax:	`runDIM(itemIdType, itemId, componentIdType, componentId, arguments)`
Description:	Runs a DIM declared in an identified COMPONENT.
Parameters:	`itemIdType` A string value indicating the type of identifier (DII Identifier or URI) that is given by the `itemId` parameter. Valid values are `dii` to indicate a DII Identifier, or `uri` to inciate a URI.
	`itemId` A string value identifying the ITEM that contains the DIM declaration of the DIM to be run or `null`.

	componentIdType A string value indicating the type of identifier (DII Identifier or URI) that is given by the componentId parameter. Valid values are dii to indicate a DII Identifier, or uri to indicate a URI.
	componentId A string value identifying the COMPONENT that contains the DIM declaration of the DIM to be run or null.
	arguments An array of zero or more objects that are to be the arguments to be passed on to the invoked DIM.
Return value:	None.
Exceptions:	DIPError With DIP error code — INVALID_PARAM if — either the itemIdType or componentIdType parameters do not specify a valid value; — both the itemId and componentId parameters are null; — the itemId or componentId are not string values or null values; — either the itemId parameter, if not null, identifies an element that is not an ITEM, or the componentId parameter, if not null, identifies an element that is not a COMPONENT; — both an ITEM is identified and a COMPONENT is identified, but the COMPONENT is not a child of the ITEM; or — if the arguments array does not contain objects of the required Argument Types for this DIM; or — NOT_FOUND if, given an identified ITEM and/or COMPONENT, a DIM to run cannot be determined; or — GENERAL_EXCEPTION if the DIM cannot be run for any other reason.

5.4.2.7.11.2 Semantics

This DIBO allows a DIM to be invoked from within another DIM.

Without the runDIM DIBO it would not be possible to call one DIM from within another DIM. The runDIM DIBO provides the DIM author with this capability.

EXAMPLE An example where this capability could be used, is the calling of a PlayTrack DIM from a PlayTracks DIM. The runDIM DIBO can then also be used from other DIMs that require playing a track.

The `itemIdType` parameter shall contain a value of either `dii` or `uri` to indicate the type of identifier given by the `itemId` parameter. A value of `dii` indicates the `itemId` parameter contains the value of a DII Identifier (part 3 of ISO/IEC 21000). A value of `uri` indicates the `itemId` parameter contains the value of a URI (IETF RFC 3986).

The `itemId` parameter shall contain a value that identifies the ITEM in which the DIM to be invoked is declared. If the value of the `itemIdType` parameter is `dii`, then the value of the `itemId` parameter shall match the value contained in a DII Identifier identifying the ITEM. If the value of the `itemIdType` parameter is `uri`, then the value of the `itemId` parameter shall be a URI identifying the ITEM.

The DIM to be invoked shall be directly declared within the identified ITEM, not in a sub-ITEM. If the DIM to be invoked is declared in a sub-ITEM, then that sub-ITEM should be the identified ITEM.

If the `itemId` parameter is null, then only the `componentId` is used.

The `componentIdType` parameter shall contain a value of either `dii` or `uri` to indicate the type of identifier given by the `componentId` parameter. A value of `dii` indicates the `componentId` parameter contains the value of a DII Identifier (part 3 of ISO/IEC 21000). A value of `uri` indicates the `componentId` parameter contains the value of a URI (IETF RFC 3986).

The `componentId` parameter shall contain a value that identifies the COMPONENT in which the DIM to be invoked is declared. If the value of the `componentIdType` parameter is `dii`, then the value of the `componentId` parameter shall match the value contained in a DII Identifier identifying the COMPONENT. If the value of the `componentIdType` parameter is `uri`, then the value of the `componentId` parameter shall be a URI identifying the COMPONENT.

If both the `itemId` and the `componentId` parameters are not null, then the identified COMPONENT shall be a child of the identified ITEM, and the DIM declared in the identified COMPONENT shall be run.

If the `itemId` parameter is null and the `componentId` parameter is not null, then the DIM declared in the identified COMPONENT shall be run.

If the `itemId` parameter is not null and the `componentId` parameter is null, then the DIBO implementation may choose any DIM declared directly in the identified ITEM (i.e., in a COMPONENT that is a child of the ITEM).

If both the `itemId` and the `componentId` parameters are null, then a `DIPError` exception is generated.

If the DIM to be invoked requires arguments, then the `arguments` parameter is an array of objects containing the arguments required by the DIM.

If the DIM cannot be invoked or an unhandled exception occurs during execution of the DIM, then a `DIPError` exception is generated.

5.4.2.7.11.3 Example use

EXAMPLE An example where this DIBO can be used is when a DI author would like to have a message, included as a *resource* in the DI, displayed prior to the playing of any one of a number of other *resources*. The display of the message might be determined by certain conditions. A DIM to check the conditions and if required display the message can be authored. This DIM can then be invoked by the `runDIM` DIBO from other DIMs prior to playing any other *resources* for which the message might be required.

5.4.2.7.12 wait

5.4.2.7.12.1 Interface

Syntax:	`wait(timeInterval)`
Description:	Waits for a given length of time.
Parameters:	`timeInterval` A number value indicating the time in milliseconds for the `Wait` operation to pause execution of the DIM.
Return value:	None.
Exceptions:	`DIPError` With DIP error code `INVALID_PARAM` — if `timeInterval` is not a positive number value

5.4.2.7.12.2 Semantics

This DIBO causes a pause in execution of the invoking DIM. On invocation of the DIBO, execution control is not returned to the DIM until the specified time interval has elapsed. It is an error to invoke this DIBO if the `waitInterval` parameter is not a positive number value, in which case an invalid parameter exception is generated.

The pause applies only to the execution of the invoking DIM.

The DIBO implementer should ensure the actual elapsed interval is as close as possible to the specified time interval, but is not obliged to ensure the exact specified time interval.

5.4.2.7.12.3 Example usage

EXAMPLE An example where this DIBO can be used is when several resources are played asynchronously, and then the author wants to pause the execution of the DIM for an interval, before stopping one or more of the resources. For example, a Digital Item representing a travel brochure might contain an audio resource, a video resource, and an HTML resource representing a "brochure" of a travel destination. A DIM can be authored such that it plays all three resources asynchronously, and then waits for an interval of time, after which the audio and video resources are released.

5.4.2.7.13 Calling DIXOs

Calling DIXOs (see 5.6.5) requires a DIBO unique to the DIXO Language. Such DIBOs are also DIBO properties of the global `DIP` object (see 5.4.2.1).

A DIBO for calling a Java based DIXO is specified in C.2.

5.4.2.8 REL related operations

5.4.2.8.1 Introduction

These DIBOs provide access to REL related functionality. The DIBOs are provided as function properties of a `REL` object. DIML includes such a `REL` object as a property of the global object (see 5.4.2.1).

EXAMPLE Calling an REL DIBO

```
function foo(arg1)
{
    …
    license = REL.getLicense( resource );
    …
}
```

5.4.2.8.2 getLicense

5.4.2.8.2.1 Interface

Syntax:	`getLicense(resource)`
Description:	Gets licenses associated with the given resource.
Parameters:	`resource` The DOM `Element` object that represents the DIDL RESOURCE element.
Return value:	An array of DOM `Element` objects that represent any licenses or null if there is no license associated with the resource.
Exceptions:	`DIPError` with DIP error code `INVALID_PARAM` if the `resource` parameter is not an `Element` object that represents a DIDL RESOURCE element.

5.4.2.8.2.2 Semantics

This DIBO provides an interface to retrieve licenses associated with a resource that are expressed as specified in ISO/IEC 21000-5 (REL).

The `resource` parameter shall be a DOM `Element` object representing a RESOURCE. It is an error to invoke this DIBO if the `resource` parameter is not a DOM `Element` object representing a RESOURCE, in which case an invalid parameter exception is generated.

NOTE 1 A license can be

— located within the DI;

— referenced from the DI;

— accessed via a service referenced from the DI; or

— none of the above.

The DIBO may return more than one license since there is the possibility that a resource has more than one license attached to it for different rights/actions.

NOTE 2 This DIBO provides an interface for the DIM author to be able to retrieve license information. **However it is not intended that this DIBO is to be used to protect a resource.** Any entity supporting processing of Digital Items as specified by ISO/IEC 21000 is expected to always check and enforce rights regardless of whether a Digital Item is interacted with via a DIM or not. In the case a Digital Item is interacted with via a DIM, the protection is performed by the underlying library of DIBO implementations. Annex G describes in more detail one way this could be done.

NOTE 3 The underlying implementation of this DIBO could call on the same library of modules used to manage rights in general within a particular MPEG-21 environment.

5.4.2.8.2.3 Example use

EXAMPLE This DIBO could be used to get licenses to pass to the `QueryLicenseAuthorization` DIBO. It is also possible to get licenses from various documents using the DOM API, but the output of the `GetLicense` DIBO would likely be a better approximation to the set of licenses that will eventually be used by the security implementation (such as the one in Annex G) anyway.

This DIBO could also be used to retrieve licenses to pass to a DIXO.

5.4.2.8.3 queryLicenseAuthorization

5.4.2.8.3.1 Interface

Syntax:	`queryLicenseAuthorization(license, resource, rightNs, rightLocal, additionalInfo)`
Description:	Checks for the existence of an authorization proof for an authorization request formed according to the semantics of this DIBO given below.
Parameters:	`license` The DOM `Element` object that represents the license information.
	`resource` The DOM `Element` object that represents the DIDL RESOURCE element.
	`rightNs` The string value that represents the namespace of the right to be checked or `null`
	`rightLocal` The string value that represents the localname of the right to be checked or the value of the definition attribute of `sx:rightUri`, depending on whether `rightNs` is a `String` or `null`, respectively.
	`additionalInfo` An array containing DOM `Element` objects representing additional information that can be considered when validating the license. This parameter may be null, in which case no additional information is provided by this parameter.
Return value:	Boolean value with value true if a corresponding authorization proof is found and false if a corresponding authorization proof does not exist or could not be found.

Exceptions:	`DIPError` With DIP error code `INVALID_PARAM` if — the `license` parameter is not an `Element` or does not contain any license information; — the `resource` parameter is not an `Element` that represents a RESOURCE element; — the `rightNs` parameter is not a string value or null value; — the `rightLocal` parameter is not a string value or null value; or — the `additionalInfo` parameter is not an array of DOM Element objects or a null value.

5.4.2.8.3.2 Semantics

This DIBO checks for the existence of an authorization proof for an authorization request having the following members.

a) A principal representing the User running this DIBO;

b) A right as specified by the rightNs and rightLocal parameters to this DIBO;

c) The resource specified by the resource parameter to this DIBO;

d) A time interval of zero length that occurs during the execution of this DIBO;

e) An authorization context that is chosen non-normatively by the DIBO implementation, possibly with the guidance of the additionalInfo parameter to this DIBO;

f) A set of licenses with one member, that member being the license specified in the license parameter to this DIBO; and

g) A set of grants chosen non-normatively by the DIBO implementation to serve as root grants, possibly with the guidance of the additionalInfo parameter to this DIBO.

NOTE 1 The utility of this DIBO is limited with respect to the number of licenses supported. e.g., This DIBO does not intend to support the case where there are two licenses: one that allows a group to play and one that assigns a User named Alice to that group.

The return value of this DIBO depends on how the authorization context and trust root are decided and therefore this DIBO should not be used when the mechanism to determine the authorization context and trust root are not known by the DIM author.

NOTE 2 The authorization context information (such as a User's location, payment history, and usage history) can be obtained in ways such as:

— Automatically, e.g., by examining the terminal properties.

— Asking assistance from the user, e.g., by popping up an input window.

— Using additional information given in the `additionalInfo` parameter.

NOTE 3 In some instances, implementations of this DIBO can knowingly construct a false authorization context to try to meet this DIBO's goals of providing a preliminary screening of User intent before continuing on with the DIM. For example, if the license is conditioned upon a per-use payment of $3, the DIBO implementation might ask a human if he wants to pay $3 for the associated right on the associated resource. Then, without charging the user $3, the DIBO implementation

might still use an authorization context that says the user paid $3 so that the result value that gets returned reflects whether the user would be authorized if he were to pay $3.

NOTE 4 The status of the result is at the time and in the authorization context that the check is evaluated. For example, it is possible in some circumstances that an authorization proof exists at the time that the DIBO is invoked, but not at a time subsequent to the invocation of the DIBO due to conditions included in the license information. It is also possible in some circumstances that an authorization proof exists using the (likely false or outdated) authorization context and trust root chosen by the DIBO implementation but does not exist using the correct updated authorization context and trust root chosen by the security subsystem.

NOTE 5 This DIBO provides an interface for the DIM author to be able to retrieve license information. **However it is not intended that this DIBO is to be used to protect a resource.** Any entity supporting processing of Digital Items as specified by ISO/IEC 21000 is expected to always check and enforce rights regardless of whether a Digital Item is interacted with via a DIM or not. In the case a Digital Item is interacted with via a DIM, the protection is performed by the underlying library of DIBO implementations. Annex G describes in more detail one way this could be done.

5.4.2.8.3.3 Example use

EXAMPLE This DIBO could be used to, before allocating system resources for a decoding process, check that at least the User has some non-expired license to play a particular Digital Item and is willing to pay $3 to do so. The ultimate enforcement of the expiration and $3 for the playing of that Digital Item would take place by the security system using a technique such as that described in Annex G. The same thing could be done without this DIBO by pre-processing the entire DIM before allocating system resources for a decoding process.

This DIBO could also be used to filter licenses retrieved via the `GetLicense` DIBO before passing them on to a DIXO.

5.4.3 Local objects, DIBOs and constants

5.4.3.1 Introduction

The following additional supporting object types are also normatively included in the DIML specification.

The DIML object types are specified by the object properties (including primitive values, other objects, and functions), and attributes of those properties.

5.4.3.2 DIPError

This DIML object inherits from the ECMAScript `Error` (see 15.11 of ISO/IEC 16262:2002) object and is thrown as an exception whenever a runtime error specific to DIP occurs. In addition to the properties inherited from the `Error` object it has the properties specified below.

Other exceptions may also be thrown during execution of a DIM. These include ECMAScript native errors (see 15.11.6 of ISO/IEC 16262:2002) and exceptions derived from `DIPError` that are specified in this or other parts of ISO/IEC 21000.

EXAMPLE A DIML syntax error will generate an ECMAScript native `SyntaxError` exception.

Value Properties:	`GENERAL_EXCEPTION` A number value of 1. This indicates a general DIP error not covered by any other error code has occurred. This property has the attributes { ReadOnly, DontEnum, DontDelete }.

	`INVALID_PARAM` A number value of 2. This indicates a parameter passed to a DIBO is invalid. EXAMPLE A `DIPError` returning this value from the `getDIPErrorCode()` DIBO will be generated if the `element` parameter passed in a call to the `play` DIBO is not an `Element` object representing a DIDL,COMPONENT, or DESCRIPTOR element. This property has the attributes { ReadOnly, DontEnum, DontDelete }.
	`INVALID_PERMISSION` A number value of 3. This indicates that an attempt was made to execute an operation for which required permission in the host environment is unavailable. EXAMPLE If the host environment does not allow execution of resources in a Digital Item, a `DIPError` returning this value from the `getDIPErrorCode()` DIBO will be generated if the `Execute` DIBO is called. This property has the attributes { ReadOnly, DontEnum, DontDelete }.
	`NOT_FOUND` A number value of 4. This indicates something required to complete an operation was not found. The missing entity is dependent on the operation that was attempted. EXAMPLE A `DIPError` returning this value from the `getDIPErrorCode()` DIBO will be generated if a DIM with the required name is not found in the identified Item when the `runDIM` DIBO is called. This property has the attributes { ReadOnly, DontEnum, DontDelete }.
	`ADAPTION_FAILED` A number value of 5. This indicates an error occurred during an attempt to adapt a resource. This property has the attributes { ReadOnly, DontEnum, DontDelete }.
	`PLAYBACK_FAILED` A number value of 6. This indicates an error occurred during an attempt to play. This property has the attributes { ReadOnly, DontEnum, DontDelete }.

	`EXECUTE_FAILED` A number value of 7. This indicates an error occurred during an attempt to execute. This property has the attributes { ReadOnly, DontEnum, DontDelete }.
	`PRINT_FAILED` A number value of 8. This indicates an error occurred during an attempt to print. This property has the attributes { ReadOnly, DontEnum, DontDelete }.
Object Properties:	None
DIBO Properties:	`getDIPErrorCode()` This DIBO returns a number value indicating the specific error that caused the exception represented by this error object to be thrown. The value returned may be one of the specified values above, or some other value specified by other parts of ISO/IEC 21000.

5.4.3.3 ObjectMap

This DIML object represents the DIP Object Map as defined in 5.3. It has the properties specified below.

Value Properties:	None
Object Properties:	None.
DIBO Properties:	`getArgumentList(index)` This DIBO returns an array of string values representing the list of Argument Types. The index parameter is a number value that indicates the index of the Argument list in the Object Map. The order of the Argument lists in the Object Map corresponds to the document order of the lists of DIP `Argument` children of all `MethodInfo` descriptors in the DIDL document and with subsequent instances of duplicate lists removed. This DIBO generates a `DIPError` exception with error code `INVALID_PARAM` if the index parameter is — not a number value; or — not a valid index (a valid index is a number ranging from 0 to one less than the value returned from `getArgumentListCount()`).

	`getArgumentListCount()` This DIBO returns a number value and is the number of Argument lists with Arguments in a specific order that are defined in the Object Map (see 5.3.5). This corresponds to the number of unique permutations for all lists of DIP `Argument` children of all `MethodInfo` descriptors in the DIDL document.
	`getMethodCount(argumentList)` This DIBO returns a number value and is the number of DIMs that are defined to accept as parameters the Arguments listed in `argumentList`. This corresponds to the number of DIM declarations in the DIDL document that have a DIP `MethodInfo` descriptor with zero or more child `Argument` elements such that the number and values of those `Argument` elements match the number and names of the specified Argument Types. The `argumentList` parameter is an array of string values that indicates the Argument Types. A zero length array passed as the `argumentList` parameter indicates DIMs declared to accept no Arguments. This DIBO generates a `DIPError` exception with error code `INVALID_PARAM` if the `argumentList` parameter is not an array of string values.
	`getMethodsWithArgs(argumentList)` This DIBO returns an array of `Element` objects representing the DIDL COMPONENT elements representing the DIM declaration for DIMs that accept as parameters the Arguments listed in `argumentList`. This corresponds to the list of DIDL Component elements, in document order, in the DIDL document that have a DIP `MethodInfo` descriptor with zero or more child `Argument` elements such that the number and values of those `Argument` elements match the number and names of the specified Argument Types. The `argumentList` parameter is an array of string values that indicates the Argument Types. A zero length array passed as the `argumentList` parameter indicates DIMs declared to accept no arguments. This DIBO generates a `DIPError` exception with error code `INVALID_PARAM` if the `argumentList` parameter is not an array of string values.

	`getMethodWithArgs(argumentList, index)` This DIBO returns an `Element` object representing the DIDL COMPONENT element representing the DIM declaration for a DIM that accepts as parameters the Arguments listed in `argumentList`. The `argumentList` parameter is an array of string values that indicates the Argument Types. A zero length array passed as the `argumentList` parameter indicates DIMs that are declared to accept no Arguments. The `index` parameter is a number value that indicates the index of the DIM in the list of DIMs that accept as parameters the Arguments listed in `argumentNames`. The order of the DIMs in the list of DIMs corresponds to the document order of the DIM declarations in the DIDL document that have a DIP `MethodInfo` descriptor with zero or more child `Argument` elements such that the number and values of those `Argument` elements match the number and names of the specified Argument Types. This DIBO generates a `DIPError` exception with error code `INVALID_PARAM` if — the `argumentList` parameter is not an array of string values; — the `index` parameter is not a number value; or — the `index` parameter is not a valid index (a valid index is a number ranging from 0 to one less than the value returned from `getMethodCount(argumentList)`).
	`getObjectOfType(typeName, index)` This DIBO returns an `Element` object of a given Object Type. The `typeName` parameter is a string value that indicates the Object Type name. The `index` parameter is a number value that indicates the index of the Object in the list of Objects of the given Object Type. The order of the Objects in the list of Objects corresponds to the document order of the elements in the DIDL document that have a child descriptor containing a DIP `objectType` descriptor with a value matching the specified Object Type. This DIBO generates a `DIPError` exception with error code `INVALID_PARAM` if — the `typeName` parameter is not a string value; — the `index` parameter is not a number value; or — the `index` parameter is not a valid index (a valid index is a number ranging from 0 to one less than the value returned from `getObjectsOfTypeCount(typeName)`).

	`getObjectsOfType(typeName)` This DIBO returns an array of `Element` objects of a given Object Type. This corresponds to the list of elements, in document order, in the DIDL document that have a child DIP `ObjectType` descriptor with value matching the specified Object Type. If there are no objects of the given Object Type, a zero length array shall be returned. The `typeName` parameter is a string value that indicates the Object Type name. This DIBO generates a `DIPError` exception with error code `INVALID_PARAM` if the `typeName` parameter is not a string value.
	`getObjectsOfTypeCount(typeName)` This DIBO returns a number value and is the number of Objects that are defined in the Object Map (see 5.3.5) to be of a given Object Type. This corresponds to the number of elements in the DIDL document that have a child DIP `ObjectType` descriptor with value matching the specified Object Type. The `typeName` parameter is a String that indicates the Object Type name. This DIBO generates a `DIPError` exception with error code `INVALID_PARAM` if the `typeName` parameter is not a string value.
	`getObjectTypeCount()` — This DIBO returns a number value and is the number of Object Types that are defined in the Object Map (see 5.3.5). This corresponds to the number of unique values for all DIP `ObjectType` descriptors in the DIDL document.
	`getObjectTypeName(index)` This DIBO returns a string value representing the Object Type name. The index parameter is a number value that indicates the index of the Object Type in the Object Map. The order of the Object Types in the Object Map corresponds to the document order of the DIP `ObjectType` descriptors in the DIDL document and with subsequent instances of duplicate values removed from the list. This DIBO generates a `DIPError` exception with error code `INVALID_PARAM` if the index parameter is — not a number value; or not a valid index for an Object Type in the Object Map (a valid index is a number ranging from 0 to one less than the value returned from `getObjectTypeCount()`).

5.4.3.4 PlayStatus

This object is returned by the `play` DIBO to enable subsequent operations on the particular instance of the Act invoked. It has the properties specified below.

Value Properties:	`RELEASED` A number value of 0. This value indicates the associated resource is not currently playing. NOTE A resource can be in the `RELEASED` state because an error prevented a request to play the resource from successfully completing, or as a result of an explicit request to release the resource. A `RELEASED` resource has no other preserved state information. Playing a `RELEASED` resource will commence with relevant state information in an initial state. EXAMPLE For a resource with time based media, playing a `RELEASED` resource will commence from the media start time. This property has the attributes { ReadOnly, DontEnum, DontDelete }.
	`STATICPLAY` A number value of 1. This value indicates the associated resource is currently playing. Time based state information related to playing the resource, if relevant, is paused for a `STATICPLAY` resource. NOTE For time based media, this status is equivalent to a state typically considered as 'paused'. This property has the attributes { ReadOnly, DontEnum, DontDelete }.
	`TIMEPLAY` A number value of 2. This value indicates the associated resource is currently playing. Time based state information related to playing the resource, if relevant, is advancing for a `TIMEPLAY` resource. NOTE Only time based media can have a status of `TIMEPLAY`. This property has the attributes { ReadOnly, DontEnum, DontDelete }.
Object Properties:	None.
DIBO Properties:	`getStatus()` This DIBO returns a number value indicating the status of the resource associated with the `PlayStatus` object. The returned value is one of the value properties specified above for the `PlayStatus` prototype.

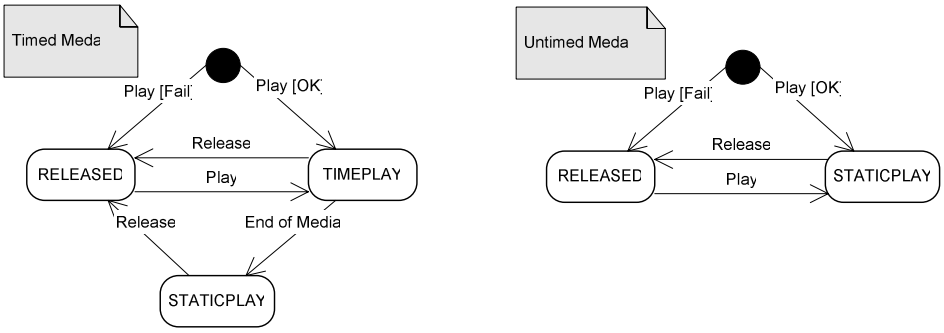


Figure 4 — State transition diagrams for `PlayStatus`

5.5 Relation of Digital Item Base Operations and RDD verbs (informative)

In some cases the functionality of the DIBOs specified in 5.4 can be characterized by association with a Rights Data Dictionary (RDD) verb (see ISO/IEC 21000-6).

The following table shows a mapping between DIBOs described in 5.4 and RDD verbs.

Table 3 — DIBO mapping to RDD verbs

Digital Item Base Operations	Rights Data Dictionary verb
`LSSerializer.write`	Adapt, Diminish, Enhance, Enlarge, Modify, Reduce
`LSSerializer.writeToURI`	Adapt, Diminish, Enhance, Enlarge, Modify, Reduce
`DIA.adapt`	Adapt, Diminish, Enhance
`DIP.execute`	Execute
`DIP.play`	Play
`DIP.print`	Print

Those DIBOs not listed in the above table typically would not require any rights checks.

For the DIBOs that are listed, it should be understood that

a) the list of rights that might need to be checked for that DIBO is not exhaustive;

b) not all the rights listed need to be checked every time that DIBO is called; and

c) it might be necessary to check rights associated with another DIBO when a DIBO is called if the latter DIBO operates on a result of the former DIBO.

For instance, since the result of a call to `DIA.adapt` is not perceivable until, for example, it is passed to `DIP.play`, it might not be necessary to check Adapt or Diminish or Enhance rights during the call to `DIA.adapt`. However in the call to `DIP.play` it might be necessary to check for both the rights associated with the adaptation and the Play right. If the adaptation was to, for example reduce the size of a resource, it might only be necessary to check either Adapt *or* Enhance.

Considering another case, if `LSSerializer.writeToURI` is called for a DIDL document with an item that has had a component removed, it might only be necessary to check either Adapt *or* Diminish. Checking Enhance in that case would not do any good. However, if the same DIBO were to be called with a component added by copying a component from a second item, it would be necessary to check both Enhance *and* Adapt. The Enhance check would be to see if the first item can be enhanced. The Adapt check would be to see if the second item can be copied from in order to enhance the first one.

Annex G gives implementation guidance on how to support DIP while maintaining a level of interaction with a Digital Item that is consistent with the available rights.

5.6 Digital Item eXtension Operations

5.6.1 Introduction

Digital Item eXtension Operations (DIXOs) are operations that may be used to extend the available DIBOs. DIXOs are provided by Users, packaged along with DIs, and acquired similar to DIMs. DIXOs, like DIBOs, can be used by DIMs but are not normatively defined. However, the mechanism to include DIXOs in Digital Items, and the mechanism to invoke DIXOs from DIMs are normatively defined.

NOTE The execution environment within an MPEG-21 environment for DIXOs is informative.

The following subclause explains the purpose of DIXOs and subsequent subclauses specify how DIXOs are defined, executed and how they interact with other components of DIP.

5.6.2 Purpose of DIXOs

DIBOs abstract out complicated operations by providing a high level interface for DIMs to access native functionality. On a number of occasions a User might need access to operation abstractions that are not normatively defined as DIBOs. Such operation abstractions can be highly specific to the application area of the DI or useful only for this DI and hence need not be normatively defined. In such situations, implementing the functionality using DIML is a possibility. However, this might neither be easy nor optimal in terms of execution efficiency and size. Thus, there is a need for a mechanism that enables the User to easily extend base operation functionality without compromising on efficiency. This is the purpose of DIXOs.

To summarize, DIXOs can be used when Users need to extend base operations when such operations are

— not normatively defined by this part of ISO/IEC 21000;

— unique to the application space;

— useful only for this DI; and

— more optimally implemented in a DIXO.

DIXOs can be implemented using the DIXO language, and packaged in DIs. The DIP engine, which is capable of handling DIXOs in that specific DIXO language, would then identify these DIXOs and make them available for use from DIMs.

5.6.3 Relationship between DIMs, DIBOs, and DIXOs

The DIMs call the DIBOs and the DIXOs to delegate processing to keep the DIM script simple. DIMs may choose to use DIBOs when they are available and DIXOs when the required functionality is not available as a DIBO.

The invocation mechanisms for DIBOs and the DIXOs from DIMs are different. The DIBOs have a mapping to DIML and this is used to call the DIBOs from within a DIM. The invocation mechanism for DIXOs is unique to the DIXO Language used to write the DIXOs. The same call for a given DIXO Language is used to invoke all the DIXOs in that DIXO Language from a DIM, where the name and the arguments of the DIXO are all in turn arguments to that invocation call.

DIXOs while implementing the extended processing may call normatively defined DIBOs and other DIXOs. The invocation mechanism of DIBOs and other DIXOs from any DIXO is direct using the bindings of DIBOs in that particular DIXO Language.

5.6.4 DIXO Language

The DIXO Language used to implement DIXOs may be chosen by the DIXO implementer. However, to execute a DIXO that is called from a DIM (and hence to execute the DIM itself), an execution environment for the DIXO Language in which the DIXO is implemented is required in the MPEG-21 environment in which the DIM is being executed.

A Java DIXO Language is specified in Annex B.

5.6.5 Calling DIXOs

DIXOs are called by an operation call unique to the DIXO Language. To call a DIXO from a DIM (and hence to execute the DIM itself), an implementation for the DIXO calling operation for the DIXO Language in which the DIXO is implemented is required in the MPEG-21 environment in which the DIM is being executed.

The syntax of the DIXO calling operation for a given DIXO Language shall conform to the following pattern.

```
runxxxDIXO( DIXOidentifier, arguments )
```

where `xxx` is a unique string of characters for the DIXO Language.

The `DIXOidentifier` parameter (or parameters) identifies the DIXO to be called. The actual values to be used for the identifier are dependent on the DIXO Language.

The `arguments` parameter is an array of zero or more arguments that shall be passed on as arguments to the called DIXO.

The DIXO calling operation, if supported by a DIP engine, shall be implemented as a DIBO property of the global `DIP` object (see 5.4.2.1).

EXAMPLE To call a DIXO taking three arguments, identified by the identifier `myDIXOid`, and implemented in the DIXO Language MYDIXL, the DIXO calling operation could be `DIP.runMYDIXLDIXO( "myDIXOid", dixoArgs ).`

The specification of a DIXO Language shall include the specification of the DIXO calling operation.

The calling procedure of a Java DIXO Language is specified in Annex C.

5.6.6 DIXO execution environment

The Digital Item Processing engine is responsible for the execution of the DIMs. Since the DIMs can invoke DIXOs which are downloaded, the requirements of the execution environment for DIXOs need to be well understood. As the DIXO Language is not uniquely identified and is left as a choice to the user, the execution environment for the chosen DIXO Language needs to exist in an MPEG-21 environment supporting the DIXO Language.

EXAMPLE An MPEG-J based execution environment for the Java DIXO Language is specified in Annex D.

5.6.7 DIXO examples

5.6.7.1 TypeText

EXAMPLE A `TypeText` DIXO could implement the functionality of displaying text *character by character* (i.e., one letter at a time). Other possible uses are: fading in and/or fading out; font effects; blinking; colour; etc. In other words, the `TypeText` DIXO can allow for text display with extended functionality, as opposed to the base (text) display mechanisms that are provided by the normatively specified DIBOs.

A possible Java-based interface for such a `TypeText` DIXO could look like:

```
TypeText(TextArea textBox, String text)
```

5.6.7.2 IMGTree

EXAMPLE An `IMGTree` DIXO could implement the functionality of displaying images hierarchically by using a tree component.

A possible Java-based interface for such an `IMGTree` DIXO could look like:

```
IMGTree(String[] ImgPath, int[] StructInfo, String[] ImgExplain)
```

5.6.7.3 SearchHighlightText

EXAMPLE A `SearchHighlightText` DIXO could implement the functionality of searching for specific words or strings in textual resources, and highlighting the words (or strings) that are found.

5.7 Auto run DIM

5.7.1 Introduction

Subclause 5.7 describes a mechanism for automatically running a DIM.

5.7.2 Identification

The DIM to be automatically run can be identified by the `autoRun` attribute of the `MethodInfo` element in the DIM declaration (see 5.3.3). If the `autoRun` attribute is present and has a value of true, then the DIM declared in that DIM declaration is an auto run DIM.

5.7.3 Time of execution

If the User wishes to automatically run a DIM without knowing any of them in particular, the User should request to run the DIM identified as the auto run DIM, as described above. In this case this DIM is executed before any other DIM is executed. If there is more than one auto run DIM available the User decides which one to run.

NOTE CHOICE/SELECTIONS and CONDITIONS can be used to configure different auto run DIMs for different cases. It is up to the author of the DID to make sure that only one auto run DIM is available if the author wants to have a single predefined entry point.

5.7.4 Examples

EXAMPLE The example below illustrates how an auto run DIM can be used to automatically start the playback of the music tracks that are declared in the DID. On receipt, the `listTracks` DIM is run. This DIM searches for music tracks and automatically plays them by calling the `playTrack` DIM.

```
<?xml version="1.0" encoding="UTF-8"?>
<DIDL xmlns="urn:mpeg:mpeg21:2002:02-DIDL-NS"
      xmlns:dii="urn:mpeg:mpeg21:2002:01-DII-NS"
      xmlns:dip="urn:mpeg:mpeg21:2005:01-DIP-NS">
  <Item>
    <Component>
      <Descriptor>
        <Statement mimeType="text/xml">
          <dip:ObjectType>urn:foo:MusicTrack</dip:ObjectType>
        </Statement>
      </Descriptor>
      <Resource ref="track1.mp3" mimeType="audio/mpeg"/>
```

```
    </Component>
    <Component>
      <Descriptor>
        <Statement mimeType="text/xml">
          <dip:ObjectType>urn:foo:MusicTrack</dip:ObjectType>
        </Statement>
      </Descriptor>
      <Resource ref="track2.mp3" mimeType="audio/mpeg"/>
    </Component>
    <Component>
      <Descriptor>
        <Statement mimeType="text/xml">
          <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIM</dip:Label>
        </Statement>
      </Descriptor>
      <Descriptor>
        <Statement mimeType="text/xml">
          <dip:MethodInfo autoRun="true"/>
        </Statement>
      </Descriptor>
      <Resource mimeType="application/mp21-method"><![CDATA[
            function listTracks()
            {
              var objTypes = ['urn:foo:MusicTrack'];
              var messages = ['Please select a track'];
              var tracks = DIP.getObjects(objTypes, messages);
              for (var i = 0; i < tracks.length; i++)
              {
                DIP.runDIM( "dii", null, "dii", "urn:foo:id:0146:4596X6745058", tracks[i]
);
              }
            }]]>
       </Resource>
    </Component>
    <Component>
      <Descriptor>
        <Statement mimeType="text/xml">
          <dii:Identifier>urn:foo:id:0146:4596X6745058</dii:Identifier>
        </Statement>
      </Descriptor>
      <Descriptor>
        <Statement mimeType="text/xml">
          <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIM</dip:Label>
        </Statement>
      </Descriptor>
      <Descriptor>
        <Statement mimeType="text/xml">
          <dip:MethodInfo>
            <dip:Argument>urn:foo:MusicTrack</dip:Argument>
          </dip:MethodInfo>
        </Statement>
      </Descriptor>
      <Resource mimeType="application/mp21-method"><![CDATA[
          function playTrack(arg)
          {
             var cmp = arg.getElementsByTagName("Component").item(0);
             DIP.Play(cmp, false);
          }]]>
      </Resource>
    </Component>
  </Item>
</DIDL>
```

5.7.5 Conformance points (informative)

There are two different conformance points relating to the auto run DIM.

a) Ability to recognize and run the auto run DIM (e.g., at time of choosing of User).

b) Always runs the auto run DIM.

Levels of conformance are

— None;

— a); or

— a) plus b)

Annex A
(normative)

ECMAScript binding for Digital Item Base Operations

ECMAScript is the basis of the DIML (see 5.2). Since the syntax for the DIBOs specified in 5.4 is provided conformant to DIML, the ECMAScript bindings for each DIBO is as specified by the syntax for each DIBO. A list of supporting global objects and their methods (DIBOs) is provided below (constant values can be found in the corresponding subclauses).

Table A.1 —List of ECMAScript bindings for DIBOs of global objects

object	DIBO
DIA	adapt(element,metadata)
DID	areConditionsSatisfied(element)
	configureChoice(choice)
	setSelection(selection,state)
DII	getElementsByIdentifier(sourceDID,value)
	getElementsByRelatedIdentifier(sourceDID,value)
	getElementsByType(sourceDID,value)
DIP	alert(message,messageType)
	execute(element)
	getData(dataTypes,requestMessages)
	getExternalData(mimeTypes,requestMessages)
	getObjectMap(document)
	play(element,async)
	print(element)
	release(playStatus)
	runDIM(itemIdType,itemId,componentIdType,componentId,arguments)
	runJDIXO(itemIdType,itemId,componentIdType,componentId,className,arguments)
	wait(timeInterval)
REL	getLicense(resource)
	queryLicenseAuthorization(license,resource,rightNs,rightLocal,addiontalInfo)

A list of supporting local objects and their DIBO properties is provided below (value and object properties can be found in the corresponding subclause).

Table A.2 —List of ECMAScript bindings for DIBOs of local objects

object	DIBO
DIPError	getDIPErrorCode()
ObjectMap	getArgumentList(index)
	getArgumentListCount()
	getMethodCount(argumentNames)
	getMethodWithArgs(argumentNames,index)
	getMethodsWithArgs(argumentNames)
	getObjectOfType(typeName, index)
	getObjectsOfType(typeName)
	getObjectsOfTypeCount(typeName)
	getObjectTypeCount()
	getObjectTypeName(index)
PlayStatus	getStatus()

Annex B
(normative)

Java bindings for Digital Item Base Operations

B.1 Introduction

Java bindings for DIBOs are specified so that Java based DIXOs (J-DIXOs) may invoke DIBOs directly from Java for optimal performance.

Each DIBO is bound to a Java interface in the package `org.iso.mpeg.mpeg21.mpegj.dibo`. The Java interface name is the same as the DIML object name that contains the DIBO as a member function as specified in 5.4.

Each interface defines a set of methods as the designated J-DIBO methods for each corresponding DIBO in the DIML object. The arguments to these designated methods are Java types corresponding to the DIML object types of the arguments of the bound DIBO. Refer to B.2 for the mapping from DIML object types to Java types. Refer to the J-DIBO interfaces in B.4 for the actual arguments to each J-DIBO method.

If the bound DIBO as specified in clause 5.4 throws the `DIPError` exception, the corresponding J-DIBO throws the Java exception `org.iso.mpeg.mpeg21.mpegj.dibo.DIPError`.

An MPEG-21 environment that supports J-DIXOs shall provide an implementation for each of the Java bindings for DIBOs.

A DIP engine shall also provide an implementation of a J-DIBO factory. The J-DIBO factory is used in a J-DIXO to obtain an instance of an object that implements the Java binding for a DIBO. Refer to B.3 for the interface for a J-DIBO factory.

NOTE The J-DIBO factory is specifically required to support J-DIXOs, hence there is no corresponding interface relating to invoking DIBOs from DIMs.

Refer to B.4 for the Java interface binding for each DIBO as specified in Clause 5.4.

EXAMPLE The following is the J-DIBO defined for the `configureChoice` DIBO.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

import org.w3c.dom.*;

/**
 * Java Binding interface for the DID related DIBOs
 */
public interface DID {

    /**
     * Designated method for the ConfigureChoice Java Binding interface
     * @param choice
     * @return true if Choice configuration was modified, false if Choice configuration
was not modified
     */
    public boolean configureChoice(Element choice) throws DIPError;
}
```

B.2 Java data type bindings for DIML object types

B.2.1 Introduction

B.2 specifies the Java data type bindings for the corresponding DIML object types specified in Clause 5.4.3 (and including ECMAScript object types).

Table B.1 — Mapping of DIML object types to Java data types

DIML object type	Java data type
`DIPError`	`org.iso.mpeg.mpeg21.mpegj.dibo.DIPError`
`ObjectMap`	`org.iso.mpeg.mpeg21.mpegj.dibo.ObjectMap`
`PlayStatus`	`org.iso.mpeg.mpeg21.mpegj.dibo.PlayStatus`
`String`	`java.lang.String`
`Number`	Appropriate Java primitive data type (`int`, `long`, `float`, `double`, etc.) as per context.
`Boolean`	`boolean` (Java basic datatype)
`Array`	An `array []` of the corresponding Java data type.

B.2.2 DIPError

See 5.4.3.2.

A `DIPError` implementation is presented below. Those data members and function members corresponding to the properties specified for the DIML `DIPError` object in 5.4.3.2 are normative. In addition, a Java binding of a `DIPError` object shall extend the Java `Exception` class and any constructors shall call the corresponding superclass constructor. Other specific implementation details are presented here for informative purposes only.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

/**
 * Java class for the DIPError. Can be used as base class
 * for DIXO based exceptions.
 *
 */
public class DIPError extends Exception {

  /* Normative */
  /**
   * General DIP error not covered by other defined error codes.
   */
  public static final int GENERAL_EXCEPTION = 1;

  /* Normative */
  /**
   * A parameter passed to a DIBO or other DIP function is invalid.
   */
  public static final int INVALID_PARAM = 2;

  /* Normative */
  /**
```

```
     * A required permission to execute an operation is unavailable.
     */
    public static final int INVALID_PERMISSION = 3;

    /* Normative */
    /**
     * Something required to complete an operation was not found.
     */
    public static final int NOT_FOUND = 4;

    /* Normative */
    /**
     * An error occurred during an attempt to adapt a resource.
     */
    public static final int ADAPTATION_FAILED = 5;

    /* Normative */
    /**
     * An error occurred during an attempt to play a resource.
     */
    public static final int PLAYBACK_FAILED = 6;

    /* Normative */
    /**
     * An error occurred during an attempt to execute a resource.
     */
    public static final int EXECUTE_FAILED = 7;

    /* Normative */
    /**
     * An error occurred during an attempt to print a resource.
     */
    public static final int PRINT_FAILED = 8;

    /* Informative */
    /**
     * Protected member variable to record error codes.
     */
    protected int errorCode;

    /* Informative */
    public DIPError(int code, String mesg) {
          super(mesg); // Normative
          errorCode = code;
    }

    /* Informative */
    public DIPError(int code, String mesg, Throwable cause) {
          super(mesg, cause); // Normative
          errorCode = code;
    }

    /* Normative */
    public int getDIPErrorCode() {
          return errorCode;
    }
}
```

B.2.3 ObjectMap

See 5.4.3.3.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

import org.w3c.dom.*;

/**
 * Java interface for the ObjectMap DIML Object Type
 */
public interface ObjectMap {

   /** Returns array of Argument Types of an argument list */
   public String[] getArgumentList(int index) throws DIPError;

   /** Returns number of unique argument lists with Arguments in a specific order */
   public int getArgumentListCount();

   /** Returns number of DIMs taking arguments of given Argument Types */
   public int getMethodCount(String[] argumentList) throws DIPError;

   /** Returns array of Elements represeting Components containing DIM declarations
    *  of DIMs taking arguments of given Argument Types.
    */
   public Element[] getMethodsWithArgs(String[] argumentList) throws DIPError;

   /** Returns Element representing Component containing DIM declaration of a DIM
    *  taking arguments of given Argument Types.
    */
   public Element getMethodWithArgs(String[] argumentList, int index) throws DIPError;

   /** Returns an Element representing an Object of given Object Type */
   public Element getObjectOfType(String typeName, int index) throws DIPError;

   /** Returns an array of Elements representing Objects of given Object Type */
   public Element[] getObjectsOfType(String typeName) throws DIPError;

   /** Returns number of Objects in Object Map of given Object Type */
   public int getObjectsOfTypeCount(String typeName) throws DIPError;

   /** Returns number of Object Types in the Object Map */
   public int getObjectTypeCount();

   /** Returns string representing Object Type name */
   public String getObjectTypeName(int index) throws DIPError;
}
```

B.2.4 PlayStatus

See 5.4.3.4.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

/**
 * Java interface for the PlayStatus DIML Object Type.
 */
public interface PlayStatus {

   /**
    * The resource has not been played, typically because a request to play
    * the resource has never been executed or some error prevented a request
    * to play the resource from successfully completing.
```

```
    */
   public static final int RELEASED = 0;

   /**
    * The resource is currently being played. For time based media, media time
   is not advancing, but playing state information (including current media time
   is preserved.
    */
   public static final int STATICPLAY = 1;

   /**
    * The resource is currently being played and for time based media, media time
   is advancing.
    */
   public static final int TIMEPLAY = 2;

   /**
    *
    * @return int indicating the current status of a played instance of a resource
   associated with this PlayStatus object.
    */
   public int getStatus();
}
```

B.3 J-DIBO factory

This clause specifies the Java interface for the J-DIBO factory. An MPEG-21 environment supporting J-DIBOs shall provide an implementation of `JDIBOFactory`.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

/**
 * JDIBOFactory is used to create new J-DIBO classes.
 *
 */
public interface JDIBOFactory {

   /**
    * This method is implemented by the J-DIBO implementation provider.
    * Essentially it returns the implementation for the DIML object interface
    * defining the set of Java-DIBO interface bound to the required DIBO.
    * @param objectName the name of one of the DIML object, e.g., "DID"
    * @return
    * @throws DIPError if any errors occur.
    */
   public Object getJDIBOObject(String objectName) throws DIPError;
}
```


B.4 Java interface bindings for DIBOs

B.4.1 Introduction

B.4 specifies the Java interface bindings for the corresponding DIBOs as specified in Clause 5.4.

NOTE A Java binding is specified for every DIBO except the `runJDIXO` DIBO which is defined in C.2.

B.4.2 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.

For authoring J-DIXOs, the Java bindings of the DOM Level 3 Core API provides the Java bindings for these base operations.

B.4.3 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.

For authoring J-DIXOs, the Java bindings of the DOM Level 3 Load and Save API provides the Java bindings for these base operations.

B.4.4 DIA related operations

See 5.4.2.4.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

import org.w3c.dom.*;

/**
 * Java Bindings for the DIA related DIBOs
 */
public interface DIA {

   /**
    * Adapts a specified Component or Descriptor
    * @param element DOM Element representing Component or Descriptor to be adapted
    * @param metadata array of DOM Elements representing additional information or null
    */
   public Element adapt(Element element, Element[] metadata)
         throws DIPError;

}
```

B.4.5 DID related operations

See 5.4.2.5.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

import org.w3c.dom.*;

/**
 * Java Bindings for the DID related DIBOs
 */
public interface DID {

   /**
    * Tests whether conditions for a specified DIDL element are satisfied
    * @param element DOM Element representing DIDL element for which conditions will be
tested
    * @return true if conditions of element are satisfied, false if not
    */
   public boolean areConditionsSatisfied(Element element) throws DIPError;
```

```
    /**
     * Requests the User to configure a Choice in the DID
     * @param choice DOM Element representing the Choice to be configured
     * @return true if Choice configuration was modified, false if Choice configuration
    was not modified.
     */
    public boolean configureChoice(Element choice) throws DIPError;

    /**
     * Sets the state of a given Selection
     * @param selection DOM Element representing the Selection
     * @param state state to set the Selection ("true", "false", or "undecided")
     */
    public void setSelection(Element selection, String state) throws DIPError;

}
```

B.4.6 DII related operations

See 5.4.2.6.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

import org.w3c.dom.*;

/**
 * Java Bindings for the DII related DIBOs
 */
public interface DII {

    /**
     * Retrieves from the source DID existing DIDL elements based on DII Identifier
     * @param sourceDID DOM Document from which to retrieve the elements
     * @param value value of DII Identifier identifying elemetns to retrieve
     */
    public Element[] getElementsByIdentifier(Document sourceDID, String value)
            throws DIPError;

    /**
     * Retrieves from the source DID existing DIDL elements based on DII RelatedIdentifier
     * @param sourceDID DOM Document from which to retrieve the elements
     * @param value value of DII RelatedIdentifier
     */
    public Element[] getElementsByRelatedIdentifier(Document sourceDID, String value)
            throws DIPError;

    /**
     * Retrieves from source DID existing DIDL elements based on DII Type
     * @param sourceDID DOM Document from which to retrieve the elements
     * @param value value of DII Type
     */
    public Element[] getElementsByType(Document sourceDID, String value)
            throws DIPError;

}
```

B.4.7 DIP related operations

See 5.4.2.7.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

import org.w3c.dom.*;

/**
 * Java Bindings for the DIP related DIBOs
 */
public interface DIP {

   /**
    * Alerts the User with a message
    * @param message
    * @param messageType
    */
   public void alert(String message, int messageType)
         throws DIPError;

   /**
    * Executes the resource associated with a Component or Descriptor
    * @param element The Element object that reflects the DIDL
    *         Component or Descriptor element.
    */
   public boolean execute(Element element)
         throws DIPError;

   /**
    * Requests the User to select resources located extenal to the DI
    * @param mimeTypes
    * @param requestMessages
    * @return array of string giving locations of the resources.
    */
   public String[] getExternalData(
         String[][] mimeTypes,
         String[] requestMessage)
         throws DIPError;

   /**
    * Retrieves the ObjectMap from a DID instance document
    * @param document
    * @return
    */
   public ObjectMap getObjectMap(Document document)
         throws DIPError;

   /**
    * Provides User with one or moe selections of Object os given Object Types.
    * @param objectTypes
    * @param requestMessages
    * @return
    */
   public Element[] getObjects(
         String[] objectTypes,
         String[] requestMessages)
         throws DIPError;

   /**
    * Requests the User for input data of one or the primitive data types
    * @param dataTypes
    * @param requestMessages
    * @return Array whose elements are of type Boolean, String or int,
    *         corresponding to the data type specified in the dataTypes array
    */
   public Object[] getValues(
```

```
        String[] dataTypes,
        String[] requestMessages)
        throws DIPError;

   /**
    * Plays a specified Component or Descriptor
    * @param element The Element object that reflects the DIDL
    *         Component, or Descriptor element.
    * @param async if true play the element asynchronously, else synchronously
    * @return PlayStatus object to identify the playing element
    */
   public PlayStatus play(Element element, boolean async)
        throws DIPError;

   /**
    * Prints a specified Component or Descriptor
    * @param element The Element object that reflects the DIDL
    *         Component, or Descriptor element.
    */
   public boolean print(Element element)
        throws DIPError;

   /**
    * Stop playback of a Component or Resource and release related playback state
information
    * @param playStatus PlayStatus object associated with the playing element
    */
   public void release(PlayStatus playStatus);

   /**
    * Runs a DIM declared in an identified COMPONENT.
    * @param itemIdType
    * @param itemId
    * @param componentIdType
    * @param componentId
    * @param args
    */
   public void runDIM(
        String itemIdType,
        String itemId,
        String componentIdType,
        String componentId,
        Object[] args)
        throws DIPError;

   /**
    * Waits for a length of time.
    * @param timeInterval - an integer
    */
   public void wait(int timeInterval)
        throws DIPError;

}
```

B.4.8 REL related operations

See 5.4.2.8.

```
package org.iso.mpeg.mpeg21.mpegj.dibo;

import org.w3c.dom.*;

/**
 * Java Bindings for the REL related DIBOs
 */
public interface REL {

   /**
    *
    * @param resource - Element representing the DIDL Resource
    * @return array of Element representing any licenses or null if none
    */
   public Element[] getLicense(Element resource)
         throws DIPError;

   /**
    *
    * @param license Element representing license information
    * @param resource Element representing DIDL Resource
    * @param rightNs namespace of the right to be checked or null
    * @param rightLocal localname of right to be checked or the value of the
   definition attribute of xs:rightUri, depending on whether rightNs is a
   String or null, respectively
    * @param additionalInfo array of Element representing additional information
   that can be considered
    * @return true if a corresponding authorization proof is found, false if a
   corresponding authorization proof does not exist or could not be found
    */
   public boolean queryLicenseAuthorization(
         Element license,
         Element resource,
         String rightNs,
         String rightLocal,
         Element[] additionalInfo)
         throws DIPError;

}
```

Annex C
(normative)

Calling MPEG-J based DIXOs from DIMs

C.1 Introduction

The MPEG-J based DIXOs (J-DIXOs) are written using the Java Programming Language. These J-DIXOs may in turn invoke the DIBOs through the Java Bindings defined in Annex B. Invoking other J-DIXOs is achieved by directly using the Java signature of that J-DIXO.

NOTE The J-DIXOs can be executed in a platform independent fashion using the execution environment defined in Annex D.

The J-DIXOs are invoked from DIMs using a single DIBO call. The generic invocation mechanism was described in subclause 5.6.5. This annex will adapt the generic invocation for MPEG-J based DIXOs.

C.2 Invoking J-DIXOs

A J-DIXO shall only be invoked from a DIM using the special DIBO `runJDIXO`. This DIBO is a DIBO property of the global `DIP` object (see 5.4.2.1). Other DIBO properties of the DIP object are specified in 5.4.2.7. The `runJDIXO` DIBO is defined as follows.

Syntax:	`runJDIXO(itemIdType, itemId, componentIdType, componentId, className, arguments)`
Description:	Executes a specified J-DIXO declared in an identified COMPONENT with the arguments supplied.
Parameters:	`itemIdType` A string value indicating the type of identifier (DII Identifier or URI) that is given by the `itemId` parameter. Valid values are `dii` to indicate a DII Identifier, or `uri` to inciate a URI.
	`itemId` A string value identifying the ITEM that contains the declaration of the J-DIXO to be run. If the value of the `itemIdType` parameter is `dii`, this value shall specify the value contained in a DII Identifier (part 3 of ISO/IEC 21000) that identifies the DIDL ITEM. If the value of the `itemIdType` parameter is `uri`, this value shall specify a URI (IETF RFC 3986) that identifies the DIDL ITEM.
	`componentIdType` A string value indicating the type of identifier (DII Identifier or URI) that is given by the `componentId` parameter. Valid values are `dii` to indicate a DII Identifier, or `uri` to indicate a URI.

	componentId A string value identifying the COMPONENT that contains declaration of the J-DIXO to be run. If the value of the componentIdType parameter is dii, this value shall specify the value contained in a DII Identifier (part 3 of ISO/IEC 21000) that identifies the DIDL COMPONENT. If the value of the componentIdType parameter is uri, this value shall specify a URI (IETF RFC 3986) that identifies the DIDL COMPONENT.
	className A string value giving the fully qualified class name of the J-DIXO to be run.
	arguments An array of objects in which each value in the array corresponds to an argument used by the J-DIXO. The argument type of each object in the array shall correspond to the type required by the J-DIXO. Rules for the types of objects that may be defined as arguments to a J-DIXO are given below. The types required by a J-DIXO are returned by the getArgumentTypes() method of the J-DIXO interface.
Return value:	Returns an object corresponding to the object type specified for the return value of the J-DIXO.
Exceptions:	DIPError with DIP error code — INVALID_PARAM if either the itemIdType or componentIdType parameters do not specify a valid value; — INVALID_PARAM if both the itemId and componentId parameters are null; — INVALID_PARAM if the itemId or componentId are not string values or null values; — INVALID_PARAM if either the itemId parameter, if not null, identifies an element that is not an ITEM, or the componentId parameter, if not null, identifies an element that is not a COMPONENT; — INVALID_PARAM if both an ITEM is identified and a COMPONENT is identified, but the COMPONENT is not a child of the ITEM; — INVALID_PARAM if the type of the objects passed in the arguments parameter do not match those returned by the getArgumentTypes() method of the J-DIXO interface; — NOT_FOUND if, given an identified ITEM and/or COMPONENT, a J-DIXO declaration containing the named class to execute cannot be determined; or — GENERAL_EXCEPTION if the J-DIXO cannot be run for any other reason.

If both the itemId and the componentId parameters are not null, then the identified COMPONENT shall be a child of the identified ITEM, and the named class in the J-DIXO declaration in the identified COMPONENT shall be run.

If the `itemId` parameter is null and the `componentId` parameter is not null, then the named class in the J-DIXO declaration in the identified COMPONENT shall be run.

If the `itemId` parameter is not null and the `componentId` parameter is null, then the DIBO implementation may choose any J-DIXO declaration directly contained in the identified ITEM (i.e., in a COMPONENT that is a child of the ITEM) that contains the named class and use that J-DIXO declaration to run the named class.

If both the `itemId` and the `componentId` parameters are null, then a `DIPError` exception is generated.

EXAMPLE

```
...
function AddTrackToPlaylist( track, playList ) {
    // Calling a J-DIXO.
    var dixoArguments = [track, playList];
    runJDIXO("uri", null, "uri", "urn:foo:bar", "com.companyx.demo.AddTrackToPlayList",
dixoArguments);
}
```

C.3 Including J-DIXOs in a DID

C.3.1 Introduction

This clause specifies how J-DIXOs are incorporated into a DID. Associating a J-DIXO with a Digital Item involves two steps:

c) The J-DIXO *declaration* refers to the declaring of the J-DIXO as being part of a particular Digital Item. C.3.2 elaborates on J-DIXO declaration.

d) The J-DIXO *definition* refers to the Java classes that define the J-DIXO. The J-DIXO definition may be listed in a separate J-DIXO location and referenced from the DID, or it may be embedded inline in the DID. In either case it is the J-DIXO definition itself that is the *resource* (in terms of the Digital Item Declaration Model). The rules associated with defining J-DIXO resources are outlined in C.3.3. Such resources can either be J-DIXO Java classes or helper classes (Java classes that the J-DIXO classes use). The rules governing the definition of J-DIXO classes are covered in C.4.

C.3.2 J-DIXO declaration

C.3.2.1 Introduction

A J-DIXO declaration is contained in a DIDL COMPONENT element which shall be constructed such that

— The COMPONENT should contain a list of J-DIXOs defined in the resource associated with the component. The list shall be represented by a `JDIXOClasses` element contained in a DIDL DESCRIPTOR-STATEMENT. If the resource associated with the component does not define any J-DIXOs but defines J-DIXO helper classes, the `JDIXOClasses` descriptor need not be present;

— The COMPONENT may contain a flag, represented by a DIP `Label` element contained in a DIDL DESCRIPTOR-STATEMENT, indicating the J-DIXO declaration is to be processed by the DIP engine; and

— The J-DIXO definitions and/or J-DIXO helper classes are referenced or embedded by a RESOURCE child of the COMPONENT.

A resource may define multiple J-DIXOs. See C.3.2.2 for details.

NOTE A J-DIXO declaration can be referenced from an external DID by utilizing provisions for document modularity as specified in ISO/IEC 21000-2.

NOTE An identifier can be associated with the J-DIXO declaration using a DII Identifier (as specified by part 3 of ISO/IEC 21000). This could be located at the level of the COMPONENT, or at the level of the ITEM containing the COMPONENT (perhaps depending on the particular application of the DI). The `RunJDIXO` DIBO (see C.2) supports both levels of identification.

C.3.2.2 JDIXOClasses syntax

```
<!--

################################################
# Definition of JDIXOClasses                   #
################################################
-->
<element name="JDIXOClasses" type="dip:JDIXOClassesType"/>
<complexType name="JDIXOClassesType">
    <sequence>
        <element name="Class" type="string" minOccurs="0"
                 maxOccurs="unbounded"/>
    </sequence>
</complexType>
```

C.3.2.3 JDIXOClasses Semantics

Semantics of `JDIXOClasses`:

Name	*Definition*
`JDIXOClasses`	Content of this element is a sequence of `Class` child elements. A DIDL COMPONENT representing a J-DIXO declaration should contain a DIP `JDIXOClasses` element contained in a DIDL DESCRIPTOR-STATEMENT. There shall be one `Class` child element for each J-DIXO class defined by this COMPONENT. Helper classes defined by this component are not listed in the `JDIXOClasses` element. If this component only defines helper classes and the `JDIXOClasses` element is still present, it shall contain zero `Class` child elements.
`Class`	Content of this element is a string indicating the fully qualified Java class name of a Java J-DIXO class that is intended to be invoked as a J-DIXO – adhering to rules laid out for J-DIXO classes.

The example in C.3.4 shows several different J-DIXO declarations specified as COMPONENT children within an ITEM.

C.3.2.4 J-DIXO Label

Subclauses 5.3.3.4 and 5.3.3.5 specify the syntax and semantics of the DIP `Label` element.

To indicate that a COMPONENT containing a J-DIXO declaration is to be processed by a DIP engine, the `Label` element shall be present (contained in a DIDL DESCRIPTOR-STATEMENT of the COMPONENT) and the value of the URI shall be `urn:mpeg:mpeg21:2005:01-DIP-NS:DIXO:Java`.

C.3.3 J-DIXO definition

A J-DIXO definition is embedded in (as base64-encoded bytes of binary data) or referenced from a DIDL RESOURCE element. Such a RESOURCE element shall be contained in a J-DIXO declaration as specified in C.3.2.

The J-DIXO definition itself shall be a Java class, a jar file containing Java classes, or a base directory that contains Java classes organized in a directory tree. The Java classes in a J-DIXO definition can be classes that are intended to be invoked as J-DIXOs and/or helper classes that are used by J-DIXO classes.

The `mimeType` for a RESOURCE element should be "`application/java`" when referring to Java class files or a base directory and should be "`application/java-archive`" when referring to a jar file.

The Java classes in a J-DIXO definition shall not be defined to belong to the package `org.iso.mpeg.mpeg21.mpegj` since this package name is reserved for J-DIBOs and other support classes specified in this standard.

The name scope of J-DIXOs or helper classes loaded is limited to the context of the parent ITEM that contains the J-DIXO declaration. Consequently, two classes having the same name but declared in different DIs, possibly with a common ITEM ancestor (but different parent ITEMs), will be loaded safely without name scope conflicts.

In order to invoke a J-DIXO from another DI, the desired J-DIXO declaration COMPONENT elements shall first be referenced in the current DI using the REFERENCE child element of the J-DIXO declaration COMPONENT. In case of a naming conflict, a J-DIXO or helper class that has been defined in a given DI will always take precedence over a J-DIXO or helper class that has been referenced from another DI.

For an example of an embedded J-DIXO definition, see the second COMPONENT in the example in C.3.4. For an example of a referenced J-DIXO definition, see the first COMPONENT in the example in C.3.4.

C.3.4 J-DIXO Example

An example of an ITEM which contains several J-DIXO declarations is given below.

In the item shown below, the first J-DIXO declaration provides an example of a `JDIXOClasses` element for a jar file. This J-DIXO declaration also shows an example of an external reference to the J-DIXO definition by referencing the location of the jar file using the `ref` attribute of the DIDL RESOURCE element.

The second J-DIXO declaration provides an example of a `JDIXOClasses` element for a J-DIXO class by itself. In this second J-DIXO declaration, the J-DIXO definition is embedded inline in the DIDL RESOURCE element using base64 encoding (complete J-DIXO definition not included for this example).

Both J-DIXO declarations also show an example of including a plain text DIDL DESCRIPTOR-STATEMENT for containing a short human readable description of each J-DIXO.

A third J-DIXO declaration shows an example of using a reference to a class path in place of a jar file as the external reference to the J-DIXO definition.

A fourth J-DIXO declaration shows an example of using XInclude [2] to reference a J-DIXO declaration contained in an external DI. The external DI is a library of J-DIXO declarations where each Component contains only two child elements. The first child element is a DESCRIPTOR containing a DIP `JDIXOClasses` element and the second child element is a RESOURCE. The J-DIXOs declared in this external DI are not intended to be processed within the context of the external DI, since they do not contain a DESCRIPTOR containing a DIP `Label` element. Instead, they can be referenced from other DIs, as in this fourth example, and those DIs can contain a DIP `Label` element to indicate the referenced J-DIXO is intended to be processed in the context of the referring DI.

EXAMPLE

```
…
<Item id="JDIXOs">
    <Component id="jdixo_01">
        <Descriptor>
            <Statement mimeType="text/plain">Description of JDIXO 1</Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIXO:Java</dip:Label>
            </Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:JDIXOClasses>
                    <dip:Class>com.companyx.demo.AddTrackToPlaylist</dip:Class>
                </dip:JDIXOClasses>
            </Statement>
        </Descriptor>
        <Resource mimeType="application/java-archive" ref="listmanip.jar"/>
    </Component>
    <Component id="jdixo_02">
        <Descriptor>
            <Statement mimeType="text/plain">Description of JDIXO 2</Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIXO:Java</dip:Label>
            </Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:JDIXOClasses>
                    <dip:Class>com.companyx.demo.RemoveTrackFromPlaylist</dip:Class>
                </dip:JDIXOClasses>
            </Statement>
        </Descriptor>
        <Resource mimeType="application/java" encoding="base64">...</Resource>
    </Component>
    <Component id="jdixo_03">
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIXO:Java</dip:Label>
            </Statement>
        </Descriptor>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:JDIXOClasses>
                    <dip:Class>com.companyx.demo.PlayPlaylist</dip:Class>
                </dip:JDIXOClasses>
            </Statement>
        </Descriptor>
        <Resource mimeType="application/java" ref="nfs://demo.companyx.com/classdir"/>
    </Component>
    <Component>
        <Descriptor>
            <Statement mimeType="text/xml">
                <dip:Label>urn:mpeg:mpeg21:2005:01-DIP-NS:DIXO:Java</dip:Label>
            </Statement>
        </Descriptor>
        <xi:include href="otherDI.xml" xpointer="element(jdixo_04/1)"/>
        <xi:include href="otherDI.xml" xpointer="element(jdixo_04/2)"/>
    </Component>
</Item>
…
```

otherDI.xml:

```
…
<Component id="jdixo_04">
    <Descriptor>
        <Statement mimeType="text/xml">
            <dip:JDIXOClasses>
                <dip:Class>com.companyx.demo. PlayTrack</dip:Class>
            </dip:JDIXOClasses>
        </Statement>
    </Descriptor>
    <Resource mimeType="application/java-archive" ref="listmanip.jar"/>
</Component>
…
```

C.4 J-DIXO Classes

A J-DIXO class can be realized by implementing the `org.iso.mpeg.mpeg21.mpegj.JDIXO` interface.

```
package org.iso.mpeg.mpeg21.mpegj;

import org.iso.mpeg.mpeg21.mpegj.dibo.DIPError;


/**
 * All J-DIXO classes must implement this interface
 */
public interface JDIXO {


    /**
    * Allows the execution engine to set the global env object. It will be
    * set by the engine soon after the JDIXO class is instantiated.
    * @param env Environment object supplied by the execution engine.
    *
    */
    public void setGlobalEnv( GlobalEnv env );

    /**
    * Starting point for the J-DIXO execution.
    * The execution engine invokes this method to execute the J-DIXO.
    * @param args array of arguments to the JDIXO (types specified by
    * getArgumentTypes())
    * @return object returned by the JDIXO (type specified by
    * getReturnType())
    * @throws DIPError if exception occurs.
    */
    public Object callJDIXO(Object [] args) throws DIPError;

   /**
    * Returns an array of the Class objects denoting the types and sequence
    * of arguments of the callJDIXO method. This method should return null
    * if the argument is void.
    * @return array of Class objects denoting the sequence and types of the
    * arguments to callJDIXO()
    */
   public Class[] getArgumentTypes();

   /**
    * Returns Class (type) of the return value of the callJDIXO method. This method
    * should return null if the return type is void.
```

```
     * @return Class object denoting return type for callJDIXO()
     */
    public Class getReturnType();

}
```

The `GlobalEnv` object is intended to provide a mechanism for J-DIXOs to receive (DIM) environment variables from the execution engine.

```
package org.iso.mpeg.mpeg21.mpegj;

import org.iso.mpeg.mpeg21.mpegj.dibo.*;
import org.w3c.dom.*;

/**
 * This class defines a mechanism for J-DIXOs to query the platform
 * for environment settings.
 */
public interface GlobalEnv {


    /**
     * Returns the instance of the JDIBOFactory which in turn is used to
     * instantiate J-DIBOs. This call must not fail.
     * @return an instance of the JDIBOFactory object.
     */
    public JDIBOFactory getJDIBOFactory();

    /**
     * Returns the instance of the Current DIDL document.
     * @return didDocument.
     */
    public Document getCurrentDIDDocument();

}
```

The platform implementation for a particular J-DIBO will be obtained from the `JDIBOFactory` object (see Clause B.3 for specification of `JDIBOFactory` interface) which can be queried from the `GlobalEnv` object.

EXAMPLE

```
import org.iso.mpeg.mpeg21.mpegj.*;
import org.iso.mpeg.mpeg21.mpegj.dibo.*;
import org.w3c.dom.*;


public class TestJDIXO implements JDIXO {
    GlobalEnv env;

    /**
     * Constructor
     */
    public TestJDIXO() {
    }

    // JDIXO methods
    /**
     * Sets globalenv
     */
    public void setGlobalEnv(GlobalEnv env) {
        this.env = env;
    }
```

```
    /**
     * callJDIXO method
     * @param args Object array
     * @return Boolean object
     */
    public Object callJDIXO(Object[] args) throws DIPError {

        Element element = (Element)args[0];
        DID didOps =
                (DID)(env.getJDIBOFactory().getJDIBOObject("DID"));

        boolean satisfied = didOps.areConditionsSatisfied(element);
        .
        .
        .
        return new Boolean(satisfied);
    }

    /**
     * callJDIXO takes one argument - an object of type org.w3c.dom.Element.
     * Therefore, return a Class array with a single element of type
     * org.w3c.dom.Element
     */
    public Class[] getArgumentTypes() {
        Class [] argTypes=null;
        try {
            argTypes = new Class[]{ Class.forName("org.w3c.dom.Element") };
        } catch (Exception e) {
            e.printStackTrace();
        }
        return argTypes;
    }

    /**
     * returns a Class object of type java.lang.Boolean
     */
    public Class getReturnType() {
        Class retType = null;
        try {
            retType = Class.forName("java.lang.Boolean");
        } catch (Exception e) {

        }

        return retType;
    }
}
```


The callJDIXO method may take arguments of the following types only.

— org.w3c.dom.Document;

— org.w3c.dom.Element;

— org.iso.mpeg.mpeg21.mpegj.dibo.ObjectMap;

— org.iso.mpeg.mpeg21.mpegj.dibo.PlayStatus;

— java.lang.String;

— java.lang.Object;

— `java.lang` classes corresponding to the primitive data types. For primitive datatypes `byte`, `int`, `long`, `char`, `float`, `double` and `boolean`, the classes `java.lang.Byte`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Character`, `java.lang.Float`, `java.lang.Double` and `java.lang.Boolean` shall be used respectively; and

— `array []` instances of any of the data types above.

The `callJDIXO` method may only throw exceptions of the following type.

— `org.iso.mpeg.mpeg21.mpegj.dibo.DIPError`

Annex D
(informative)

MPEG-J based model for execution of DIXOs

This annex describes the execution model for MPEG-J based Java bytecode DIXOs with Java as the DIXL. MPEG-J defines an application engine addressing the security, delivery, life cycle, and name scope aspects. MPEG-J can be used as a good framework to execute DIXOs sent as part of the Digital Items. Compiled DIXO code can be safely executed without risk of causing harm to the User or breaking the privacy of users or other downloaded code that is being run within the virtual machine.

The DIXL for the MPEG-J based model would be Java as mentioned above. Among other features Java provides platform independence, is object-oriented, has language support to express concurrency and provides localization and internationalization support.

The MPEG-J based application engine consists of the following.

— Java Virtual Machine;

— Supported platform (`java.*`) packages: `java.io`, `lang` and `util`;

— Required ISO/IEC JTC 1/SC 29/WG 11 defined (`org.iso.mpeg.mpegj.*`) APIs;

— Required Java mappings for normative DIBO APIs (`org.iso.mpeg.mpeg21.dibo.*`).

Invocation of J-DIXO methods has been discussed in Annex C.

Annex E
(informative)

XML Schema Definition for Digital Item Processing Elements

This annex provides the complete XML Schema Definition for Digital Item Processing Elements.


```
<?xml version="1.0" encoding="UTF-8"?>
<!--
################################################################################
# ISO/IEC 21000-10                                                             #
#   Information technology                                                     #
#   - Multimedia framework (MPEG-21)                                           #
#     - Part 10: Digital Item Processing                                       #
#                                                                              #
################################################################################

-->
<schema
    targetNamespace="urn:mpeg:mpeg21:2005:01-DIP-NS"
    xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:dip="urn:mpeg:mpeg21:2005:01-DIP-NS"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!--

    ##############################################
    #  Definition of MethodInfo                  #
    ##############################################
    -->
    <element name="MethodInfo" type="dip:MethodInfoType"/>
    <complexType name="MethodInfoType">
        <sequence>
            <element name="Argument" type="anyURI" minOccurs="0"
                     maxOccurs="unbounded"/>
        </sequence>
        <attribute name="autoRun" type="boolean" use="optional"
                   default="false"/>
        <attribute name="profileCompliance"
                   type="dip:ProfileComplianceType" use="optional"/>
    </complexType>
    <!--

    ##############################################
    # Definition of ProfileComplianceType        #
    ##############################################
    -->
    <simpleType name="ProfileComplianceType">
      <list itemType="QName"/>
    </simpleType>
    <!--
```

```
    ################################################
    # Definition of Label                          #
    ################################################
    -->
    <element name="Label" type="dip:LabelType"/>
    <complexType name="LabelType">
        <simpleContent>
            <extension base="anyURI"/>
        </simpleContent>
    </complexType>
    <!--

    ################################################
    # Definition of ObjectType                     #
    ################################################
    -->
    <element name="ObjectType" type="dip:ObjectTypeType"/>
    <complexType name="ObjectTypeType">
        <simpleContent>
            <extension base="anyURI"/>
        </simpleContent>
    </complexType>
    <!--

    ################################################
    # Definition of JDIXOClasses                   #
    ################################################
    -->
    <element name="JDIXOClasses" type="dip:JDIXOClassesType"/>
    <complexType name="JDIXOClassesType">
        <sequence>
            <element name="Class" type="string" minOccurs="0"
                     maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <!--
    -->
</schema>
```

Annex F
(informative)

A media handler implementation of play DIBO

F.1 Introduction

This Annex provides an informative description of an implementation of the `play` DIBO that utilizes the concept of a media handler. While the description particularly addresses the `play` DIBO it is clear the implementation technique can be applied to the implementation of any DIBO.

F.2 Key concepts

F.2.1 Media handler

A media handler is a module that lies below the level of the DIBO and is called by the DIBO implementation. Different media handlers can be implemented to handle media with different characteristics, such as media type.

F.2.2 Switching DIBO

A switching DIBO is a DIBO whose implementation primarily comprises switching between calls to lower level media handlers.

F.3 Perspectives on DIMs and DIBOs

Some different perspectives from which DIMs and DIBOs can be considered are listed below.

— A User that is an End User of the Digital Item – a User in this role can interact with the Digital Item using a DIM (i.e., by being a DIM End User);

EXAMPLE 1 To consume the resources contained in the Digital Item.

— A User that is the author of a DIM – a User in this role suggests interactions with the Digital Item by authoring a DIM. The DIM author has access to the functionality (defined by the normative semantics) provided by DIBOs via the ECMAScript bindings (determined by the normative syntax). Java bindings are also defined for authoring DIXOs;

— A User that processes a DIM – a User in this role provides the execution environment for processing a DIM. Such a User might be a software application running on a device; and

— An implementer of DIBOs used in a DIM – a DIBO implementer can implement the DIBO by any means provided that it provides the functionality defined by the normative DIBO semantics, and this functionality is made accessible to the DIM author via the normative DIBO syntax.

EXAMPLE 2 The manufacturer of the device mentioned above.

F.4 A Play Track DIM and the `play` DIBO

The syntax and semantics of the `play` DIBO is specified in 5.4.2.7.8.

The functionality of the `play` DIBO is that it provides a transient and directly perceivable representation of a DID entity (*component* or *descriptor*) represented by the `element` argument to the DIBO.

From the perspective of a DIM End User, the End User is not directly concerned in regards to the call to the `play` DIBO. The DIM End User's point of entry for interacting with the DI is selecting a DIM to be executed.

EXAMPLE 1 With a music album DI, the DIM End User might have access to a Play Track DIM. From the DIM End User perspective, they see only the Play Track DIM (plus any other accessible DIMs) and request the Play Track DIM to be run. The DIM End User would reasonably expect that when they run the DIM, an audio *resource* of the track will be played such that they could directly perceive it. The End User need not be concerned with the fact that the DIM author has used a call to the `play` DIBO to provide this functionality.

It is a DIM End User that invokes the DIM. This would be done in a manner determined by the provider of the environment in which the DI is being consumed.

EXAMPLE 2 In a desktop software application a menu of accessible DIMs could be presented, allowing the End User to choose a DIM to be run.

For the DIM author, the author might want to first ensure that the DIM End User has configured a certain user configurable *choice*, prior to playing the track.

EXAMPLE 3 In the Play Track DIM the DIM author could include a call to the `configureChoice` DIBO prior to the call to the `play` DIBO.

The DIM author could also cause other DID entities to be played if they consider that to be part of 'playing a track'.

EXAMPLE 4 In addition to the *component* containing the audio resource, they might play a *component* containing a text *resource* containing the lyrics of the track.

From the DIM End User perspective, all these interactions are part of the Play Track DIM.

It is a DIM author that implements (authors) a DIM, and in so doing includes calls to DIBOs. The DIMs are declared and defined within DIDs as specified in subclause 5.3. In authoring DIMs, a DIM author uses DIML (see subclause 5.2), including the DIBOs (see subclause 5.4). In addition the DIM author can invoke a DIXO (see subclause 5.6) from a DIM.

A DIP engine could be provided, for example, by the manufacturer of the device on which the DI is being consumed, or the developer of a software application being used to consume the DI on a desktop computer system. The provider of the DIP engine also provides the execution environment for the DIM, including handling the invocation and execution of the DIBOs called from the DIM. For the DIBO implementer, they provide an implementation of the DIBO consistent with the semantics of the DIBO and utilizing the information, if any, provided as arguments to the DIBO by the DIM author.

It is a DIBO implementer that implements the DIBOs. The DIBOs are invoked by the DIP engine as it processes a DIM authored by a DIM author and invoked by the DIM End User.

At the level above the DIP engine, an overall DI processor is also required. In many cases, the provider of the DI processor, DIP engine, and DIBO implementations could be the same entity.

EXAMPLE 5 An MPEG-21 desktop computer application that supports DIP developed and provided entirely by a single provider.

However, this might not be the case in all circumstances.

EXAMPLE 6 One provider might provide a library of DIBO implementations optimized for a particular mobile platform, and another provider might use that library in their own implementation of a DI processor for that same platform. In this case the DIBO implementations provided by the DI processor could be simply wrapper calls to the library of DIBO implementations provided by the other provider.

F.5 Implementing a DIBO

A key feature of DIBOs in general is that DIBO implementers can choose to provide the functionality specified by the semantics of the DIBO in a manner of their choosing.

EXAMPLE 1 When implementing the `play` DIBO, one implementer might choose to present the DIM End User with a graphical user interface allowing them to choose a third party application with which to render any *resources* associated with the DID entity being played. Another implementer might choose to directly render the *resources* itself if it supports the media type of the *resources*, and report an error for those media types for which it does not support.

In the case of the example music album and Play Track DIM discussed above, it would be natural to expect an audio *resource* of the track to be rendered as perceivable sound. However, it is also possible that a DIBO implementer, possibly in conjunction with information provided by the DIM End User (e.g., via user preferences), might also or alternatively provide some other form of perceptible medium.

EXAMPLE 2 If the DIBO implementation supports accessibility for physically impaired consumers, and it is known that the DIM End User is hearing impaired, then the DIBO implementer might provide alternative non-aural cues or adapted audio.

In addition, the DIBO implementer is able to provide a DIBO implementation as they see appropriate.

EXAMPLE 3 The choice of implementation might depend on the execution environment such as a handheld mobile device versus a desktop computer.

In implementing the DIBOs they have available any appropriate technology for the environment in which the DIBOs will be invoked.

EXAMPLE 4 A desktop software application developed in Java can develop their DIBO implementations in Java. Or they could develop their DIBO implementations in C, and access the platform specific compilations of these using the Java Native Interface (JNI).

EXAMPLE 5 A mobile device application developed for the Symbian OS could also utilize Java, but with Mobile Information Device Profile (MIDP) restrictions.

EXAMPLE 6 Another DIBO implementer supporting DIP in a desktop software product might implement the DIBOs using C++.

The choice of technologies to implement a DIBO are as wide as the choice of target platforms and architectures on which DIBOs will be executed. In some cases a provider of a particular platform might provide some library with a defined API for access to platform specific features that could be utilized by the DIBO implementer.

However the functionality defined by the normative semantics of the DIBO is still required to be provided by the DIBO implementer. In the case of `play`, the semantics require that a transient and directly perceivable representation of the DID entity be presented. In addition, this functionality is accessed (from the DIM by the DIM author) via the normative syntax of the DIBO.

To support DIMs and DIBOs in Digital Items, at least the following are required.

- Parsing of a DID;
- Parsing of DIP information contained in a DID;
- Execution environment for DIML, including
  - invocation and execution of available DIMs, and
  - implementation of DIBOs.

F.6 Implementing the `play` DIBO

F.6.1 Introduction

The semantics of the `play` DIBO is to provide a transient and directly perceivable representation of a DID entity. The DID entity passed to the `play` DIBO might contain *resources* that could be of any media type. This requires that the `play` DIBO is able to determine the media type of the *resources*, and then determine an appropriate representation of the *resources*.

F.6.2 Information available to DIBO implementer

The DIM author provides as a parameter to the `play` DIBO the *component*, *or descriptor* to be played and a boolean flag indicating whether to play the DID entity synchronously or asynchronously. The DID entity and its children are actually represented in DIML by DOM `Element` objects which in turn reflect the corresponding DIDL elements in the DID.

In addition to the parameters passed to the DIBO by the DIM author, the DIBO implementer has available any additional information that can be obtained from the DIM execution environment.

EXAMPLE The DIBO implementer might also have access to other elements of the DID by the same means used to access the DIDL elements in general processing of the DI. This would enable the DIBO implementer to access additional information in the DID that might be relevant for the DIBO implementation. In the case of the `play` DIBO, the DIBO implementer might also be able to access any *descriptors* associated with the DID entity. These *descriptors* could provide information such as licensing information (e.g., as specified in ISO/IEC 21000 part 5) or information enabling adaptation of the *resources* contained in the DID entity (e.g., as specified in ISO/IEC 21000 part 7).

F.6.3 Example implementation

The example below lists some high level pseudo-code for one possible implementation of the `play` DIBO.

EXAMPLE 1 Pseudo-code for one possible implementation of the `play` DIBO

```
for each resource contained in the DID entity
do
    mimeType = get value of 'mimeType' attribute of resource
    if mimeType is 'text/plain'
    then
        while license, if any, valid
        do
            display text data in text viewer
        done
    else if mimeType is 'audio/mpeg'
    then
        while license, if any, valid
        do
            if audio controller not displayed
            then
                display audio controller
            endif
            render audio, controlled by audio controller
        done
    else
        display message 'Sorry, unsupported media type'
    endif
endfor
```

It is important to remember that this is an example of a possible DIBO implementation as implemented by a DIBO implementer, and not a DIM authored by a DIM author.

The DIBO implementer can access whatever is provided by the DIM processing environment.

EXAMPLE 2 Since license checking is done regardless of how a *resource* is accessed, the license checks indicated in the above example could be done by calls to a library of license related functions which are used not only by the DIBO implementation but by the DI processor in general. This library of license related functions could be provided by the provider of the DI processor themselves, or it could be provided by a different provider.

Similarly for the actual rendering of a *resource*, rather than being implemented directly within the DIBO implementation, this could be done by calls to appropriate media rendering functions.

From the example above, it can be seen that a general feature of the `play` DIBO implementation is different handling of a *resource* based on its media type. This leads naturally to the concept of media handlers which are called from the DIBO implementation.

F.6.4 Media handlers

A media handler is a function available to the DIBO implementer that can be called from the DIBO implementation (see Figure F.1 — DIBO implementation using media handlers). A media handler is able to handle media with known characteristics, such as a particular MIME media type, and implement the DIBO semantics for that media.

The example DIBO implementation in the previous clause could be simplified by delegating all processing to media handlers. In this case the DIBO implementation primarily becomes a switch to dispatch the actual processing (in accordance to the DIBO semantics) to an appropriate media handler. This type of DIBO will be referred to as a switching DIBO in this Annex.

EXAMPLE 1 Pseudo-code for a possible switching DIBO implementation of the `play` DIBO

```
for each resource contained in the DID entity
do
    if MIME top-level media type is 'text'
    then
        call ACME corp. fast play text handler
    else if media type is MPEG-4 AAC
    then
        call Blogg's play AAC handler
    else
        display message 'Sorry, unsupported media type'
    endif
endfor
```

The media handlers need not be provided by the DIBO implementer. In addition there can be many different available media handlers.

EXAMPLE 2 One provider might specialize in providing media handlers for a few selected DIBOs for a particular type of media. Another provider might provide a library of media handlers targeted at a particular platform for the `play` DIBO for a number of common media types. Another provider might provide a library of media handlers for their own media type for all media handling DIBOs. The DIBO implementer would be free to choose among these or other media handlers, as well as providing their own media handlers, as appropriate for their needs.

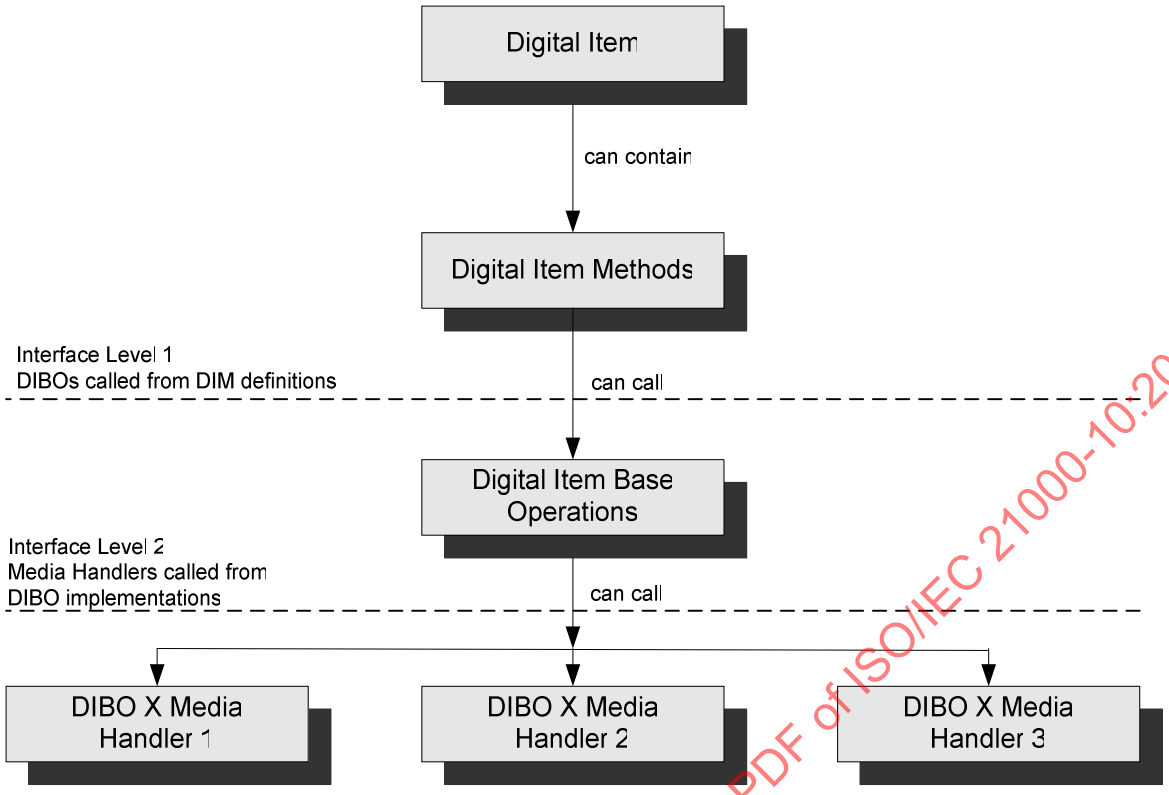


Figure F.1 — DIBO implementation using media handlers

F.6.5 Enhancements to media handlers

In the previous subclause a switching DIBO implementer is required to know the API to call the appropriate media handler, which could vary among providers of media handlers. Media handlers could be enhanced and made more interoperable by defining tools for

— identifying media handlers;

— identifying media handlers, either required or suggested, to consume a *resource* using a particular DIM;

— identifying media handlers available in a given DIM processing environment, and

— accessing and invoking identified media handlers from a switching DIBO implementation.

These tools could be utilized, for example,

— within a DI by a DI author (e.g., as a *descriptor* bound to a *resource*);

— within a description of a DIM processing environment (e.g., a digital item describing the capabilities of an MPEG-21 enabled mobile device), and

— by a DIBO implementer to determine and invoke an appropriate media handler (e.g., if a media handler *descriptor* for the *resource* to be played is present and the identified media handler is available, invoke the media handler).

EXAMPLE Pseudo-code for a possible switching DIBO implementation of the `play` DIBO with extensions

```
for each resource contained in the DID entity
do
    if media handler descriptor available for resource
    then
        if media handler available
        then
            call media handler
        else
            display message 'sorry, required media handler not available'
        endif
    else
        if MIME top-level media type is 'text'
        then
            call ACME corp. fast play text handler
        else if media type is MPEG-4 AAC
        then
            call Blogg's play AAC handler
        else
            display message 'Sorry, unsupported media type'
        endif
    endif
endfor
```

These basic enhancements could be extended further

EXAMPLE By including ability for utilizing trusted media handlers.

Annex G
(informative)

Tracking DIM execution for consistent rights checks

G.1 Introduction

This Annex gives implementation guidance on how to support DIP while maintaining a level of interaction with a Digital Item that is consistent with the available rights.

This Annex begins with a brief description of a) how to create a well-behaved implementation without worrying about supporting DIP and b) how to support DIP without worrying about being well-behaved. It then highlights the case where DIP is supported consistently with the available rights and outlines an approach to accomplishing this combined support.

G.2 Creating a well-behaved implementation without worrying about supporting DIP

An implementer who is not supporting DIP understands how their code interacts with Digital Items. Being well-behaved is simply a matter of finding the appropriate points in their implementation during its interaction with a Digital Item to check for the rights to do that interaction.

G.3 Supporting DIP without worrying about being well-behaved

An implementer who is not worried about creating a well-behaved application does not need an intimate understanding of how their implementation is interacting with Digital Items. Supporting DIP is simply a matter of leaving a certain amount of the decision of how to interact with a Digital Item up to the DIM author.

G.4 Supporting DIP while being well-behaved

An implementer who supports DIP and also desires to be well-behaved needs to address the challenge of being able to, at the appropriate times, check for the rights to interact with the Digital Item in the way suggested by the DIM author. Since the DIBOs provide much finer granularity than the base rights in the REL and RDD and since the implementer does not know ahead of time the sequences of DIBOs that the DIM author will use, the implementer needs to somehow, while processing the DIM, aggregate the effect of a sequence of DIBOs to determine the appropriate rights to check.

For example, the DIBO sequence of `getFirstChild`, `getFirstChild`, `adapt`, and `play` might correspond to a rights check for the “Play” right. Another DIBO sequence of `getFirstChild`, `getFirstChild`, `adapt`, `replaceChild`, and `writeToURI` might correspond to a rights check for the “Adapt” right. Even though both DIBO sequences begin with the same first three DIBOs, they result in different rights checks because they end with different “critical DIBOs” (in this case, `play` and `writeToURI`, respectively). A list of known “critical DIBOs” is given in subclause 5.5.

Another example to consider is the DIBO sequence of `getFirstChild`, `getFirstChild`, `play`. This sequence ends in the same “critical DIBO” as the first sequence (namely “Play”) and might also result in a rights check for the “Play” right. However, the context information for this rights check would differ from the context information for the rights check in the first example. The rights check in this example would have information indicating that the resource will be played in its original form while the rights check in the first example would have information indicating that the resource will be played in some adapted form (for instance, at reduced resolution).

One way to collect the additional context information necessary to perform the appropriate rights check is to implement some sort of information tracking into the implementation of each of the DIBOs such as `getFirstChild` and `adapt`. The information that needs to be collected includes what the operation operated on and how the operation affected the things it operated on.

Example 1 shows an example section of DIM code.

EXAMPLE 1 Section of DIM code.

```
// Get the item out of the current DIDL Document
itemA = document.getFirstChild().getFirstChild();
// Load up another DIDL Document and get the item out of it
document2 = lsParser.parseURI("did2.xml");
itemB = document2.getFirstChild().getFirstChild();
// Import the second item into the current DIDL Document
itemB = document.importNode(itemB, true);
// Get the metadata and resource from the second item
component = itemB.getFirstChild();
// Adapt the component according to the metadata in its descriptors
adaptedComponent = DIP.Adapt(component, null);
// Replace original component with adapted component
itemB.replaceChild(adaptedComponent, component);
// Create a new descriptor with a statement with the text "cool"
statement = document.createElement("Statement");
cool = document.createTextNode("cool");
statement.appendChild(cool);
descriptor = document.createElement("Descriptor");
descriptor.appendChild(statement);
// Add the descriptor
component.appendChild(descriptor);
// Add the second item as a child of the first item
itemA.appendChild(itemB);
// Write out the new first item
lsSerializer.writeToURI(document, "cool.xml");
```

The above section of DIM code gets a first item from the current DIDL document. It then loads up a second item, changes it a bit, and then adds it to the first item. In the end, when the updated first item is written out, the first item was “enhanced” and the second item was “adapted”. So before writing out the updates a well-behaved implementation should check for “Enhance” rights to the first item and “Adapt” rights to the second item. Example 2 shows an example of the tracking information that can be stored during the execution of the above section of DIM code to make it possible to figure out the correct rights to check.

EXAMPLE 2 Example tracking information stored during execution of DIM code.

```
#document_9.getFirstChild() xxxxxxxxxx   --->   DIDL_10
DIDL_10.getFirstChild() xxxxxxxxxx   --->   Item_11
TrackedDocument() xxxxxxxxxx   --->   #document_12
#document_12.getFirstChild() xxxxxxxxxx   --->   DIDL_13
DIDL_13.getFirstChild() xxxxxxxxxx   --->   Item_14
#document_9.importNode() Item_14   --->   Item_16
Item_16.getFirstChild() xxxxxxxxxx   --->   Component_17
Component_17.getFirstChild() xxxxxxxxxx   --->   Descriptor_18
Component_17.getLastChild() xxxxxxxxxx   --->   Resource_19
adaptResource(Resource_19, Descriptor_18) Resource_19   --->   Resource_20
Component_17.replaceChild(Resource_20, Resource_19) Component_17   --->   Component_21
#document_9.createElement() xxxxxxxxxx   --->   Statement_22
#document_9.createTextNode() xxxxxxxxxx   --->   #text_23
Statement_22.appendChild(#text_23) Statement_22   --->   Statement_24
#document_9.createElement() xxxxxxxxxx   --->   Descriptor_25
Descriptor_25.appendChild(Statement_24) Descriptor_25   --->   Descriptor_26
Component_21.appendChild(Descriptor_26) Component_21   --->   Component_27
```

```
Item_11.appendChild(Item_16) Item_11   --->   Item_28
```

“#document_9” represents the global document object. “Item_28” is the item that is stored out. Looking at the last line of the tracking information, we can see that “Item_28” came by adding “Item_16” into “Item_11”. So “Item_11” (ItemA) was enhanced. We can also see that parts of “Item_16” were heavily changed and that “Item_16” originally came from “Item_14”. So “Item_14” (ItemB) was adapted. These two rights (enhancing of ItemA and adaptation of ItemB) would be checked for.