INTERNATIONAL STANDARD

ISO/IEC 23007-1

First edition
2010-11-01

Information technology — Rich media user interfaces —

Part 1: Widgets

Technologies de l'information — Interfaces d'utilisateur au support riche —

Partie 1: Widgets

Reference number
ISO/IEC 23007-1:2010(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 23007-1 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 23007 consists of the following parts, under the general title *Information technology — Rich media user interfaces*:

— *Part 1: Widgets*

— *Part 3: Conformance and reference software*

Advanced user interaction interface will form the subject of a future Part 2.

Introduction

User interface represents a crucial feature for many consumer devices and services. User interfaces have recently evolved to support more media types including audio, video, 2D or 3D graphics and rich media functionalities. User interfaces are also evolving towards flexible and composite collections of small dedicated applications retrieved from different sources and aggregated into an effective and user friendly interface. Such applications are generally called widgets, a widget being a self-contained entity, with an interactive and dynamic visualization.

Additionally, more and more devices are capable of displaying rich media user interfaces, from desktop computers, to mobile devices, to home appliances, including TV sets. In this heterogeneous environment, users expect a homogeneous, unified experience when interacting with their devices.

The objective of this part of ISO/IEC 23007 is to provide normative interfaces between widgets and widget managers, to allow widgets from different service providers to run, communicate and be transferred within a unique framework.

In this part of ISO/IEC 23007, widgets can be processed by entities running on different devices, called widget managers, in charge of processing and managing the life cycle of the widgets supporting communications with other entities locally or remotely deployed and enabling widget mobility across devices.

This part of ISO/IEC 23007 is also known as “MPEG-U”. This part of ISO/IEC 23007 addresses the normative aspects of the MPEG-U widgets. In particular, it specifies widget packaging formats, aspects for widget communications with external entities and for widget mobility. It also contains a technical annex describing a list of use cases and examples to address such use cases. ISO/IEC 23007-2 will specify advanced user interaction interfaces to support various advanced user interaction devices. ISO/IEC 23007-3 addresses reference software and conformance aspects.

This part of ISO/IEC 23007 builds upon the W3C specification for widgets, packaging and configuration:

— to ensure that the widget packaging format and configuration documents are compatible with the MPEG media types which can be used to describe widgets (e.g. 2D or 3D content, MPEG-4 BIFS or MPEG-4 LASeR). For restricted profiles of these languages, this implies in particular the ability to create meaningful widgets which do not rely on scripting languages.

— to ensure that widgets can be transported on any existing transport mechanisms, in particular those defined by MPEG (e.g. ISO base media file format and the MPEG-2 Transport Stream).

— to ensure that it is targeted for domains in addition to Web-connected devices, e.g. broadcast, mobile or home networking domains.

— to enable interoperable communications between a widget and other entities (including widgets), these entities being remote (e.g. UPnP services [4]) or local services, or other widgets running in the same environment.

— to enable MPEG-specific requirements, such as the ability to dynamically update the widget presentation or to display a widget in a dynamic and interactive simplified representation.

— to enable widgets, mobility across devices while maintaining the state of the widget.

A general description of the architectures of this part of ISO/IEC 23007 is provided first, to clearly identify normative and non-normative entities.

This is followed by descriptions of behaviour and syntax of the normative elements, and syntax and examples of the normative elements for which a definition of new XML syntax is needed, namely widget manifest and widget API.

Annex A and Annex B provide registration forms for two media types defined within this part of ISO/IEC 23007. A complete example of the standardized technology can be found in Annex C. Examples of use cases which can be realized using MPEG-U are provided in Annex D. Finally, a description of the relationship of this specification with the W3C widgets family of specifications is provided in Annex E.

Information technology — Rich media user interfaces —

Part 1:
Widgets

1 Scope

This part of ISO/IEC 23007 defines a specification for the exchange, the control and the communication of widgets with other entities, a widget being a self-contained living entity, with an interactive and dynamic visualization.

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 14496-12, *Information technology — Coding of audio-visual objects — Part 12: ISO base media file format (technically identical with ISO/IEC 15444-12)*

W3C WPC "Widgets 1.0: Packaging and Configuration", W3C Working Draft 24 February 2009, available at http://dev.w3.org/2006/waf/widgets

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
communication entity
device or widget, local or remote to a widget manager, providing services

3.2
context information
set of data needed to reproduce a state and preferences of a widget, should it be deactivated and reactivated, possibly in a different widget manager

3.3
device
combination of hardware and software or just an instance of software that allows a user to perform actions

3.3
full representation
description of the widget appearance and behaviour given in a scene description language which represents the complete version of the widget with its maximum complexity and behaviour

3.4
icon
interactive, possibly animated and/or scripted, raster image or vector graphics that can be used to graphically represent the widget before the full representation is loaded

3.5
iconic representation

NOTE See simplified representation.

3.6
locale
set of information which defines local variants of widgets, elements for a particular country or region or language

3.7
manifest
XML description of the widget containing all the information necessary for the widget manager to process the widget

3.8
presentation engine
entity processing the scene description of the widget to provide its animated and interactive behaviour through composition and rendering

3.9
resource
part of a widget, in the form of either a file or a stream, which is needed by the widget manager or presentation engine to process and present the widget

3.10
scene description
description defining an audiovisual presentation for the widget in terms of spatiotemporal layout, and interactions by using text, graphics, animations, images, videos, sounds, etc.

3.11
service
system supporting interaction, local or over a network, by means of message exchanges (e.g. UPnP service or Web service)

3.12
simplified representation
description of the widget appearance and behaviour given in a scene description language which represents a version of the widget with a reduced complexity and/or behaviour compared to the full representation

3.13
widget
self-contained entity, with extensive communication capabilities, within a rich media user interface, composed of a manifest and associated resources, including scene descriptions for the full and simplified representations and context information

3.14
widget manager
user agent, processing widgets, in particular for communication between the widget and other entities

3.15
widget package
collection of the widget manifest and associated resources in a particular format used for delivery and storage

4 Abbreviations and symbols

ISOFF	ISO Base Media File Format
BIFS	BInary Format for Scene
LASeR	Lightweight application scene representation
SVG	Scalable Vector Graphics
UPnP	Universal Plug and Play
W3C WPC	Widgets 1.0: Packaging and Configuration
W3C WAE	The widget Interface
WLC	Widget life cycle
WM	Widget manager
XML	eXtensible Markup Language

5 Conventions

XML element and attribute names are written using this `style` in the text.

XML examples are written as follows:

```
<mw:messageOut name="Search">
 <mw:output name="searchString" scriptParamType="string"/>
 <mw:input name="URL" scriptParamType="string"/>
</mw:messageOut>
```

Script code examples are described as follows:

```
function myFunction { … }
```

6 Architectures

6.1 Widget manager architecture

The architecture of the different elements specified in this part of ISO/IEC 23007 is depicted in Figure 1.

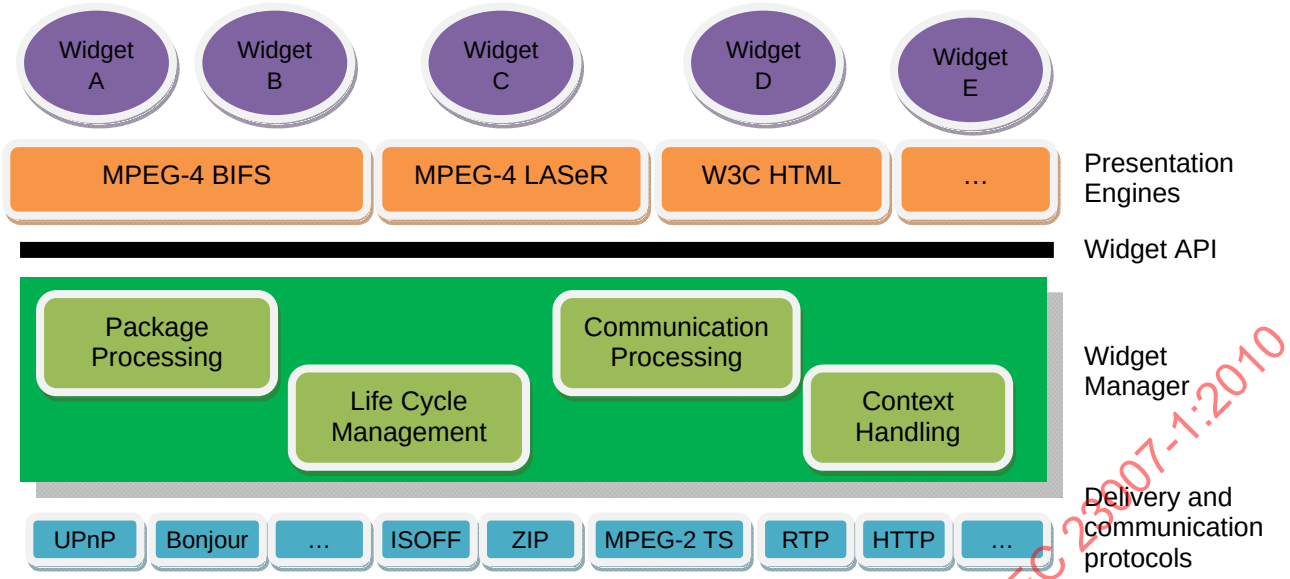

Figure 1 — Architecture of the rich media user interface standard

On client devices, an entity, called widget manager, shall provide the following functionalities:

— Processing widget packages, as defined in W3C WPC,

— Managing the life cycle of widgets, as described in 7.2,

— Managing the communication between widgets and external entities, as described in 7.3,

— And managing context information for widget, as described in 7.4.

Additionally, widget managers may support one or more of the widget delivery mechanisms described in Clause 8.

The presentation of a scene description of a widget is demanded to a presentation engine, running on the client device, supporting the scene description language. The presentation engine is an external entity interfaced with the widget manager. The widget manager may provide support for the API defined in Clause 9 to the presentation engine.

6.2 Widget architecture

In this part of ISO/IEC 23007, a widget shall be made of:

a) the manifest, i.e. an XML description which serves as an entry point for the widget and provides:

   i) metadata about the widget (e.g. author name, short title, …),

   ii) the format and the URL of the scene descriptions for the full and simplified representation,

   iii) description of the communication capabilities of the widget,

   iv) a set of preferences that needs to be restored and an indication whether they need to be saved as part of the context information;

b) optional context information to be used for restoration on a same or different widget manager;

c) the full representation of widget, i.e. a set of resources (e.g. scene description data, images, text content, …), allowing a full featured presentation of the widget;

d) one or more simplified representations, also called icons, i.e. an optional set of resources (possibly the same as the first one), allowing the retrieval or presentation of the widget in a simple state or way.

The simplified representation may be a different scene from the full representation, or may be a fragment of the full representation. Using a different scene may allow a fast presentation of the widget when the full representation is not yet available. This may also allow a simplified presentation when the full processing is not required, or too demanding.

7 Widget management

7.1 Widget composition

The spatial, temporal and interactive composition of the simplified or full representation of a widget shall be as specified by this representation. If the full and simplified representation both point to the same resource, a single instance of that resource may be used and the simplified and full rendering may work on the same synchronized compositing.

The spatial and temporal composition of the widgets together with other widgets or applications is not specified.

EXAMPLE In case of widgets being displayed on a TV at the same time as a TV program, widgets may be displayed on top of the TV program, or the TV program may be resized to show the widgets on the side, or any other paradigm may be used.

7.2 Widget life cycle

The widget life cycle (WLC) represents the set of states and transitions that a widget can be in during its lifetime. The WLC is depicted in Figure 2.

The widget manager shall apply the steps specified in Section 9 of W3C WPC. If this process fails, the widget is placed in the invalid state and the widget manager shall not process this widget. If the process is successful the widget manager shall place the widget in the validated state and provide the information from the manifest to the presentation engine. The widget manager shall not offer a widget for activation until all interfaces having a required attribute set true are ready to be bound.

NOTE 1 The icon or a text string may be used to show to the user that the widget is validated. This exact behaviour is implementation specific. A validated widget can then be selected for presentation. This is done either automatically or by user interaction, and its full representation and/or its simplified representation is started. The selection mechanism is also implementation specific. When a scene representation of a validated widget is selected for presentation, this representation is loaded by the presentation engine which processes it as defined in the related scene representation specification.

EXAMPLE If the widget manager and the presentation engines are interfaced to propose a widget dock, the presentation engine may allow the user to choose which simplified representation is to be used in the dock.

If the loading of the scene representation fails, the widget manager shall place the widget into the invalid state and shall not process it. If the loading is successful, the widget state becomes active and the widget manager shall apply the communication and context management behaviours as defined in 7.3 and 7.4.

NOTE 2 The presentations of the different widget representations are not exclusive. The simplified and full representations may be presented together by the widget manager, e.g. in a dock for the simplified presentation and in the main area for the full representation of the same widget. The transitions (if any) between the presentation of the full and simplified representation are implementation specific. Additionally, each representation of a widget may be shown or hidden.

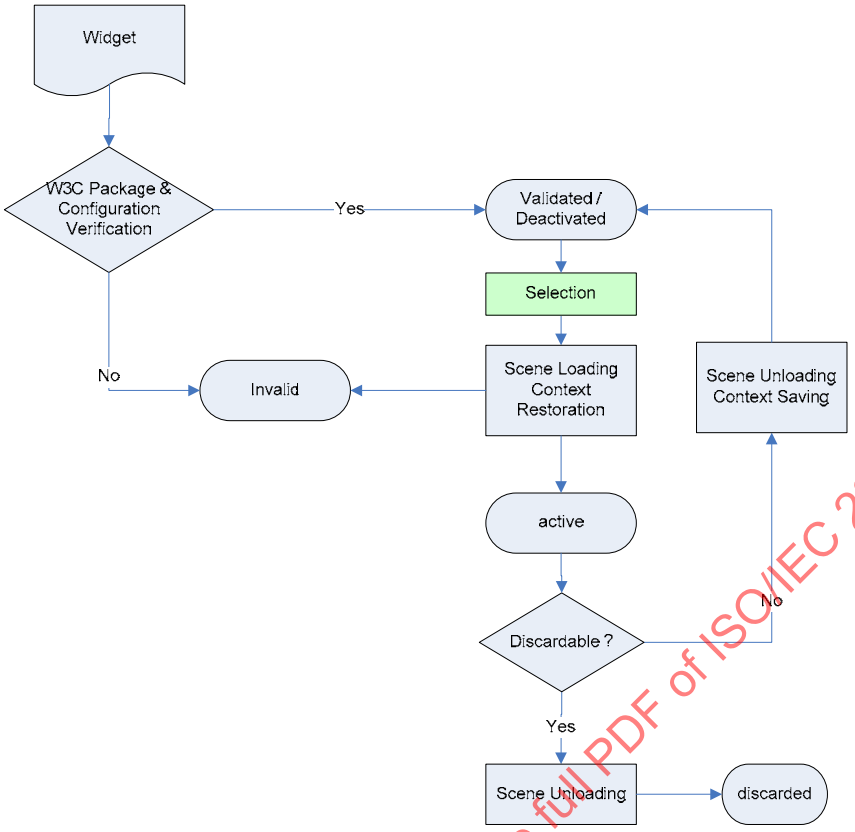

Figure 2 — MPEG-U widget life cycle

There are four events relevant to the life of each representation as specified in Table 1. These events may be used in particular to distinguish the substates of the active state of the widget life cycle when a single resource is used to describe both the full and simplified representations. These events are communicated to the widget using the mechanism described in 7.3.

Table 1 — Widget life cycle events

activateSimple	The simplified representation has been loaded and is now active and shown
activateFull	The full representation has been loaded and is now active and shown
showFull	The full representation was hidden and has been shown
showSimple	The simplified representation was hidden and has been shown
hideFull	The full representation was shown and has been hidden
hideSimple	The simplified representation was shown and has been hidden
deactivateFull	The full representation is going to be removed from the widget manager in the near future.
deactivateSimple	The simplified representation is going to be removed from the widget manager in the near future.

7.3 Widget communication

7.3.1 Overview

As regular scenes, widgets may communicate with any external entities using communications means supported by the presentation engine (e.g. Web servers and XMLHttpRequest object [2]; or streaming server and the RTSP protocol [3]). However, the widget manager offers means for widgets to communicate with entities for which URL are not known at widget authoring time nor at widget delivery time but when the widget enters the active state. This is the case of widgets communicating with devices in a home network (e.g. UPnP media server [4]) or the case of widgets communicating with local resources (e.g. battery status) or the case of widgets communicating with other widgets running in the same widget manager. For that purpose, widget authors may use the `MPEGwidget` script interface defined in Clause 9 to determine the address of the linked external communication entity.

In this Part of ISO/IEC 23007, we assume that the type of each communication entity (external or internal), or of its services (remote or local), can be identified by a unique string (e.g. using URN) and that a description of the communication capabilities of that entity is available. This description lists the messages that the entity can emit or receive. Each message has a name and a list of parameters. Each parameter carries a value and is identified by a name.

7.3.2 Matching services and interfaces

The widget manager shall match the available services with the interfaces of the activated widgets. This matching process is normative and may happen at any time, possibly at multiple times, when the widget is active. The matching process is as follows.

NOTE The widget manager is responsible for discovering available services which may come from local or remote communication entities, like devices or widgets. Services may become available or unavailable at any time.

First, the widget manager shall check the type of the widget interface and compare it with the string identifying the type of a service (e.g. the URN of the UPnP service). The exact comparison algorithm is service specific.

Then, if the types match, the widget manager shall compare the definition of the interface with the description of the service capabilities (e.g. UPnP actions and events) at the message level. For each message in the widget interface, there shall be a message in the service description with the same name. If this is not the case, the matching process fails.

Finally, for each message, the widget manager shall check:

— If, for each input in the widget message declaration, there is an output parameter in the service description, with the same name. If this fails, the widget manager shall check that there is a default value for this input. If this is not the case, the matching process fails. If after processing all input in the widget message declaration, some output parameters in the service description are not matched, the widget manager shall proceed with the assumption that these output parameters are not used in the widget.

— If for each output in the widget message declaration, there is an input parameter in the service description, with the same name. If this fails, the matching process fails.

In case of success of the matching process, the widget manager shall check that nodes or functions referenced in the widget interface declaration (see 10.23) are present in the widget representation when the matching process is performed. If this is not the case, the matching fails.

If the matching is successful, the interface is considered bound and the widget manager shall notify it to the widget either using the `mw:bindAction` attribute (see 10.15) or using the `onInterfaceBind` callback in the API (see 9.2). In case of failure of the matching, the widget manager shall not bind the interface.

In case the service previously bound to an interface becomes unavailable, the interface is considered unbound and the widget manager shall notify it to the widget either using the `mw:unbindAction` attribute (see 10.15) or using the `onInterfaceUnbind` callback in the API (see 9.2).

In the case of multiple services matching an interface not supporting multiple bindings (see 10.15), the decision of which service to bind is implementation specific. However, widget managers may interact with the user to take the decision.

Widgets may be seen as special case of services. Therefore, the above matching process shall be used to determine if widgets can communicate with each other.

Finally, a widget manager may decide to terminate communications between a widget and a service or between two widgets. This decision is implementation-specific. However, it shall notify the widget(s) of the unbinding of the interfaces.

7.4 Widget context

Widget managers shall maintain context information for each widget across activations. If several instances of a single widget are activated at the same time, widget managers shall maintain as many context information representations as there are activated instances. In particular, if a widget instance is deactivated, the widget manager shall save the context information as indicated in the manifest, and if later on reactivated, the widget manager shall restore this context information.

Widget managers shall maintain the context information of a widget according to the `<mw:contextConfiguration>` element.

NOTE 1 The way the context information is kept internally in the widget manager (e.g. using files or databases) and the way a context information is associated to a widget instance (e.g. using unique identifier) is implementation specific.

Widget managers shall control which preferences of the context information can be exchanged according to the `mw:migratable` attribute (see 10.10) of the <mw:preferenceConnect> element.

Widget managers shall support serialization of context information in the form of the XML format defined in Clause 11.

Widget managers shall also support parsing and processing of this XML format for context information restoration.

When a widget is activated for the first time by a widget manager, if context information is available in the widget package or at the URL defined by the syntax below, the widget manager shall retrieve this context information and process it to restore preference values that supersede the preference values indicated in the widget manifest. Widget manager may support retrieval of context information from other URL. Additionally, widget managers may support processing of context information in additional cases that at first activation.

EXAMPLE When a widget is exchanged back and forth between two widget managers, a second activation by the first widget manager may benefit from processing the context information of the second widget manager.

The URL syntax defined in this Part of ISO/IEC 23007 to retrieve context information associated to a widget is the following:

— `<base_url>?mpeg-u-context` if <base_url>, which denotes the URL used to retrieve the widget, does not contain '?'

— `<base_url>&mpeg-u-context` if <base_url> already contains a '?'.

NOTE 2 There are cases where the context information cannot be transmitted in the manifest, for example when the package is digitally signed and when the widget sender cannot modify the package signature. This URL syntax is designed in particular for these cases.

When a widget manager receives context information, it shall check if it has already retrieved the associated widget, identified by the ci:uuid and ci:version attributes. If this is the case, the widget manager shall apply the context information to that widget. If the associated widget is not available, it shall try to retrieve the widget from the ci:url attribute. If the retrieval is successful, the widget manager shall apply the context information. If the retrieval fails, the context information shall be ignored.

8 Widget packaging and delivery

8.1 Overview

This Clause specifies how to deliver widget content over the various existing transport protocols. There are two general options: delivery of unpackaged content, where the manifest and associated resources are delivered separately; and packaged delivery.

The packaged delivery may be used to avoid multiple connections to servers or for distribution convenience reasons. Widgets may be delivered using the packaged format defined by WPC or using the ISOFF-based packaged format specified in this Clause. Since the ISOFF allows both efficient processing of stream-based media and packaging of non timed data, the packaging of widgets based on the ISOFF avoids multiple levels of packaging and allows efficient playback of the widget-packaged audio/visual content in widget-unaware players.

If widgets are delivered using the format specified in WPC, the widget package must be conformant to that specification.

8.2 Unpackaged delivery

Widget content can be delivered unpackaged, e.g. over HTTP. The delivery process starts with the delivery of the manifest. Manifest files shall be identified with the media type `application/mw-manifest+xml`.

The delivery of subsequent resources follows as specified by their respective URL and protocols.

EXAMPLE The URL in the manifest which points to the full representation or the simplified representation may use any existing protocols, in particular streaming protocols as specified in RFC 3640 for ISO/IEC 14496-11 (BIFS) or ISO/IEC 14496-20 (LASeR) content.

For internationalization purposes, the use of a localization mechanism in the widget manager, such as the one defined in Section 8 of W3C WPC, is dependent on the delivery mechanism:

— For pull mode, widget managers shall provide localization information to the server (e.g. HTTP language accept header);

NOTE Indeed, the use of localized URL resource would lead to several requests (e.g a a request for ‘fr-fr’ localized resource, another request for ‘fr’ localized resource and a last request for the non-localized resource) and be inefficient.

— For push mode and for delivery mechanisms supporting localization of resources (e.g. using MPEG-4 ESD), widget managers shall not apply localization of URL before requesting a resource;

— For push mode and for delivery mechanisms not supporting localization of resources (e.g. FLUTE [6]), widget managers shall apply localization of URL before requesting a resource, and use fallback URL if needed, as specified Section 8 of W3C WPC.

8.3 Packaged delivery using the ISOFF

For widgets packaged using the ISOFF, the following applies:

— The file shall use the brand ‘mwgt’ either as major brand or as in the compatible-brands list of the file-type box.

— The manifest (compressed or not) shall be packaged as the primary item with the name config.xml, the media type `application/mw-manifest+xml` and stored in the XML box at the file level. The associated handler_type of handler box shall be ‘mwgt’.

— Other resources shall be stored as items in the file. If localization is needed, URLs used to address resources shall be first processed according to the localization behaviour specified in 8 of W3C WPC and then the localized URL shall be used in the shadowing mechanism as defined in 8.44.7 of ISO/IEC 14496-12 to determine their exact location in the ISO file. If this processing does not resolve to an item in the file, or if localization is not used, the URL shall be directly processed in the shadowing mechanism.

A player conforming to this Part of ISO/IEC 23007 shall compose the streams stored in the ISO base media file according to the rules given in 7.1.

NOTE File delivery protocols like FLUTE [6] or DSM-CC Object Carousel [7] may be used to deliver ISOFF packaged widgets in a broadcast scenario.

9 Widget API

9.1 Overview

This Clause defines APIs for use in widgets using scripting languages such as ECMAScript [5].

9.2 The MPEGwidget interface

If a widget manager implements a widget object as defined in W3C WAE, this object shall also implement the `MPEGwidget` interface to support the widget communications mechanism defined in 7.3. Interfaces are specified using IDL [9].

```
interface MPEGwidget extends widget {

    InterfaceHandler[] getInterfaceHandlersByType(in DOMString interfaceType);

    attribute Function onInterfaceBind;

    attribute Function onInterfaceUnbind;

    attribute Function onActivation;

    void activateComponentwidget(in DOMString componentID);

    void deactivateComponentwidget(in DOMString componentID);

}
```

The `getInterfaceHandlersByType` method allows a widget to retrieve an array of objects implementing the `InterfaceHandler` interface of the given type, and representing bound interfaces.

The function `onInterfaceBind` can be set to a function which will be called by the widget manager upon the binding of any interface. The function `onInterfaceUnbind` can be set to a function which will be called by the widget manager upon the unbinding of any interface.

The function `onActivation` can bet set to a function which will be called by the widget manager upon activation or upon failure to activate a component.The return parameter is a Boolean indicating if the activation was a success (true) or a failure (false).

The `activateComponentwidget` method is called by a widget to trigger the activation of a component widget. The only argument is the id of the component as indicated in the widget manifest.

The `deactivateComponentwidget` method is called by a widget to trigger the deactivation of a component widget. The only argument is the id of the component as indicated in the widget manifest.

9.3 The InterfaceHandler interface

The widget manager shall provide an `InterfaceHandler` object per defined interface of the widget.

```
interface InterfaceHandler {

    readonly attribute DOMString bound;

    readonly attribute DOMString type;

    void invoke(in MsgHandler msgHandler, …);

    MsgHandler msgHandlerFactory( in DOMString msgName, in function
callBackFunction);

    void invokeReply(in MsgHandler msgHandler, …);

    MsgHandler msgHandlerFactory();

}

interface MsgHandler {

    readonly attribute DOMString msgName;

    readonly attribute InterfaceHandler interfaceHandler;

}
```

The `bound` string indicates the address (e.g. IP Adress, hostname …) of the external communication entity corresponding to this `InterfaceHandler` object.

The `type` string indicates the type of the external communication entity corresponding to this `InterfaceHandler` object.

A script within a widget may call the `invoke` method from the `InterfaceHandler` object to send a message to the corresponding external communication entity. The message is built from the arguments of the method. Each argument is mapped to a message parameter as indicated in the external communication entity's interface declaration, in the declaration order. The widget manager shall trigger the `callBackFunction` upon reception of the entity's reply, giving the `MsgHandler` object as first argument, enabling the script to distinguish multiple calls to the callback.

When scripts functions are called upon reception of an incoming message (i.e. because the function is declared in the `mw:inputAction` attribute of a `<mw:messageIn>` element), the script may reply to the originating communication entity.

The `invokeReply` method is used to send a reply to a previous message. In that case, upon the call to this method, the widget manager shall send a message back to the communication entity. The first argument is a `MsgHandler` object. The message output parameters are mapped to the method's following arguments in order.

EXAMPLE Widget A defines a method doSearch. Widget B wishes to use the service in question. The widget manager identifies that widget A and B are compatible for communications. It connects the output of widget B to the input of widget A. Upon invocation by widget B of the “Search” method, the widget manager sends a message to widget A which triggers a call to “searchFunction” method. widget A then performs its search, creates and stores a MsgHandler object and return. When its processing is done, widget A calls the invokeReply method with the MsgHandler object and the result of its process. This sends a reply message which in turns calls back widget B and its myProcessSearchResults method with the result (the value of URL).

The manifest of widget A declares in an interface of type “example:search:interface” within which the following is declared:

```
<mw:messageIn name="doSearch" inputAction="searchFunction">
 <mw:input name="searchString" scriptParamType="string"/>
 <mw:output name="URL" scriptParamType="string"/>
</mw:messageIn>
```

The manifest of widget B declares in a interface of type “example:search:interface” within which the following is declared:

```
<mw:messageOut name="Search">
 <mw:output name="searchString" scriptParamType="string"/>
 <mw:input name="URL" scriptParamType="string"/>
</mw:messageOut>
```

In widget B, a script contains:

```
void myProcessSearchResults(msgHandler, URL) { … }

var ifce = widget.getInterfaceHandlersByType(“example:search:interface”)[0];

var msgHandler = ifce.msgHandlerFactory(“Search”,myProcessSearchResults);

ifce.invoke(msgHandler, “some message”);
```

In widget A, a script contains:

```
var mH;

void searchFunction(searchString) {

   … // some code to start the search

   mH =
widget.getInterfaceHandlersByType(“example:search:interface”)[0].msgHandlerFactor
y();

}

void searchFunction1() { // called when the search is finished

   var URL = …

   mH.interfaceHandler.invokeReply(mH, URL);

}
```

10 Widget manifest syntax

10.1 Overview

In this part of ISO/IEC 23007, a widget is described by a widget manifest using an XML description. For elements defined in W3C WPC no prefix is used. The extensions defined in this Part of ISO/IEC 23007 use the namespace '`urn:mpeg:mpegu:schema:widgets:manifest:2010`'. The prefix '`mw:`' is used below for this extension namespace. The XML syntax of the MPEG-U extensions to the W3C widgets configuration document is described below and is given in the attached schema file named mpegwidget.xsd.

NOTE The widget.xsd file is an informative helper attachment.

10.2 The <widget> element

The <widget> element is defined in W3C WPC.

W3C defined attributes: id, version, width, height, xml:lang, viewmodes

Extension Attributes:

`mw:type`

Optional. Identifies the type of widget with a string. This attribute may be used to present available widgets to the user in a structured way or to filter out widgets of type not wished by the user.

EXAMPLE A possible classification scheme for widgets could be the following:

Utilities / News / Games / Radio / Search / Clocks / Webcams / Countdown / Weather / Communication / Shopping / Sports / Programming / Audio / Transportation / Photos / Entertainment / Finance

`mw:multipleInstances`

Optional. A boolean which indicates if multiple instances of this widget will always present the same result (multipleInstances==“false”) or not. If not specified, the default value is “false”.

`mw:discardable`

Optional. Boolean attribute with a default value of false. It indicates whether the widget shall be discarded by the widget manager after its deactivation and shall no longer be used by the widget manager.

`mw:uuid`

Optional. If specified, this attribute defines a unique identifier for this widget, regardless of the version, as defined in IETF RFC 4122.

10.3 The <name> element

The <name> element is defined in W3C WPC.

W3C defined attributes: xml:lang, its:dir, short

10.4 The <description> element

The <description> element is defined in W3C WPC.

W3C defined attributes: xml:lang, its:dir

10.5 The <author> element

The <author> element is defined in W3C WPC.

W3C defined attributes: href, email, its:dir

10.6 The <license> element

The <license> element is defined in W3C WPC.

W3C defined attributes: href, xml:lang, its:dir

10.7 The <icon> element

The <icon> element is defined in W3C WPC.

W3C defined attributes: src, width, height

Widget managers shall support manifests in which the value of the src attribute of the icon element is the same as the value of the src attribute of the content element, possibly with an additional fragment (e.g. #svgView()).This URL can use any protocol and is in particular not limited to http.

10.8 The <feature> element

The <feature> element is defined in W3C WPC.

W3C defined attributes: name, required

For widgets conformant to this Part of ISO/IEC 23007, a <feature> element, whose "name" attribute value is "`urn:mpeg:mpegu:schema:widgets:manifest:2010`" shall be present in the widget manifest. The value of the "required" attribute is not restricted and can be either "optional" or "required".

The feature name "`urn:mpeg:mpegu:widgets:needsPlaceComponent`" shall be used in the widget manifest when the widget requires that the widget manager executes placeComponent messages. Such a widget shall not be activated by a widget manager that is unable to execute placeComponent messages.

Profiles and levels of MPEG data (e.g. audio, video, scenes …), used by the widget, should be declared in the widget manifest using the <feature> element and "name" values declared in the MPEG URI asset document [10].

10.9 The <param> element

The <param> element is defined in W3C WPC.

W3C defined attributes: name, value

10.10 The <preference> element

The <preference> element is defined in W3C WPC.

W3C defined attributes: name, value, readonly, xml:lang

10.11 The <content> element

The <content> element is defined in W3C WPC.

In extension to W3C WPC, widget manifest conformant to this Part of ISO/IEC 23007 may have multiple <content> elements but with different "type" attribute values. In such case, widget managers shall process the content elements, in document order, as described in WPC, but as if each content element was the first encountered element and stop processing when the widget start file, as defined in WPC, is set.

W3C defined attributes: src, type and encoding

Extended attributes:

`mw:minWidth / mw:minHeight`

Optional. Size in pixel under which the rendering of this representation will give poor result. If missing, the value 0 is assumed.

`mw:maxWidth / mw:maxHeight`

Optional. Size in pixel above which the rendering of this representation will give poor result. If missing, the infinite value is assumed.

10.12 The <mw:contextConfiguration> element

Description:

This element indicates to the widget manager that it needs to maintain context information for this widget and describes what this context information is by providing links to scene constructs. Such context shall be saved when the widget is deactivated and restored when the widget is activated. The preference elements in this widget shall be saved, restored or transmitted according to the `mw:migratable` attribute of the preferenceConnect elements of the same name.

Context:

This is a child of a <content> element.

Expected children (in any order):

It contains 0 or more <mw:preferenceConnect> elements.

Attributes:

mw:saveTrigger

Optional. Indicates the scene construct that is used to trigger the saving of the context by the widget manager. See 10.23.3 for the syntax.

mw:restoreTrigger

Optional. Indicates the scene construct that is used to trigger the restoring of the context by the widget manager. See 10.23.3 for the syntax.

mw:savedAction

Optional. Indicates the scene construct that is activated once the context has been saved by the widget manager. See 10.23.2 for the syntax.

mw:restoredAction

Optional. Indicates the scene construct that is activated once the context has been restored by the widget manager. See 10.23.2 for the syntax.

10.13 The <mw:preferenceConnect> element

Description:

This element makes a connection between a scene construct and a preference of the widget. It enables saving a scene construct, when its value changes, as a widget preference value.

Context:

This is a child of a <mw:contextConfiguration> element.

Expected children (in any order):

None.

Attributes:

mw:name

Mandatory. Indicates the name of a preference as indicated in the name attribute of a preference element (see 10.10). In the absence of a preference element with a matching name, this <mw:preferenceConnect> element is ignored.

mw:connectTo

Optional. Indicates the scene attribute connected to this preference. See 10.23 for the syntax.

mw:migratable

Optional. Indicates how this preference is saved and transferred when migrating the widget across devices or both. Possible values are:

— "saveOnly": the preference is saved but never migrated

— "migrateOnly": the preference is migrated but never saved to disk

— "saveAndMigrate": the preference is saved and migrated. This is the default value.

mw:value

Optional. If specified, it overrides the preference value given in the preference element with the same 'name' attribute value.

NOTE If multiple <content> elements are specified pointing to different scene types (e.g. SVG and BIFS), the preference value given in the <mw:preferenceConnect> element allows specifying a preference value adapted to the scene type (e.g. respectively according to SVG or to MPEG-4 BIFS).

10.14 The <mw:interfaces> element

Description:

Each <content> element may have an <mw:interfaces> element which describes the communication capabilities of the widget for the given representation. A widget manifest may not contain any <mw:interfaces> element, in which case the communication capabilities specified in Clause 10 are not available.

Context:

This element is a child of the <content> or <icon> element.

Expected children (in any order):

<mw:interface>: one or more.

Attributes:

This element does not have any attribute.

10.15 The <mw:interface> element

Description:

This element and its children describe the communication capabilities of the widget for a certain type of communication. An interface is bound dynamically with a communication entity when a match is found between the type attribute and the external communication entity description (see 7.3.2). Upon loading of the widget, the widget manager may propose to bind the interfaces with its own internal communication entities, and to the external communication entities already discovered. Upon the discovery of a new entity, unbound interfaces of validated widgets are examined for a possible match with the new entity and additional widgets may be processed.

Context:

This is a child of an <mw:interfaces> element

Expected children (in any order):

<mw:messageIn>: zero or more

<mw:messageOut>: zero or more

Attributes:

`mw:type`

Mandatory. A string identifying the type of interface. This attribute is used in the binding process as described in 7.3.2.

EXAMPLE mw:type="urn:schemas-upnp-org:service:AVTransport:1"

`mw:bindAction`

Optional. Indicates the scene construct to change or trigger, or the name of a script function to call upon the successful binding of this interface. See 10.23.2 for the syntax.

`mw:unbindAction`

Optional. Indicates the scene construct to change or trigger, or the name of a script function to call upon the unbinding of this interface. See 10.23.2 for the syntax.

`mw:serviceProvider`

Optional. Boolean, default false. Indicates whether this interface provides a service (true) or not. If it does, it may be published on the network as a discoverable service, so that for example other widgets in other widget managers on the network may communicate with it.

`mw:connectTo`

Optional. String, with no default value. Indicates the id of a component declared in this manifest. It indicates whether this interface shall be bound to a matching interface of the component rather than any other matching interface.

`mw:multipleBindings`

Optional. Boolean, default false. Indicates whether this interface can be bound only once (default behaviour) or multiple times.

`mw:required`

Optional. Boolean, default false. Indicates whether the activation of this widget requires that this interface is ready to be bound.

10.16 The <mw:messageIn> element

Description:

The <mw:messageIn> element is used to group multiple <mw:input> elements and, possibly, multiple <mw:output> elements together. It is used to define how messages received from external communication entities are dispatched to the widget and how the reply to this message may be created and sent.

The received message parameter's values are dispatched to the scene according to the <mw:input> element with the same name attribute value and the action specified in the mw:inputAction attribute is processed (attribute modification or script function call).

If the input action is a script function, the arguments of the function are the values indicated by the <mw:input> elements in declaration order.

If the input action is a declarative construct (attribute change or event firing), the action is performed after all <mw:input> elements are processed.

A received message may lead to a reply. This reply may be triggered either declaratively, as specified in the mw:outputTrigger attribute or via the invokeReply method.

If the output trigger is specified, it shall indicate a declarative construct (attribute change or event triggered), and the output message parameters values are taken from the values indicated by the <mw:output> elements.

If the output trigger is not specified, the message may only be triggered upon call to the invokeReply method. The values of the arguments of the invokeReply method are used, in the declaration order, to set the output message parameter values.

If multiple <mw:messageIn> elements with the same "name" attribute value are declared within an interface, upon reception of a message with that name, all these <mw:messageIn> elements are processed, in declaration order.

Context:

The <mw:messageIn> element is a child of an <mw:interface> element.

Expected children:

It contains first zero or more <mw:input> elements followed by zero or more <mw:output> elements.

Attributes:

`mw:name`

Mandatory. Indicates the name of a received message.

`mw:inputAction`

Optional. Indicates the scene construct to change or trigger, or the name of a script function to call upon receiving the message. See 10.23.2 for the syntax.

`mw:outputTrigger`

Optional. Indicates the event which triggers the sending of the reply. See 10.23.3 for the syntax.

EXAMPLE 1 Receiving a message with no reply (declarative case)

```
<mw:messageIn name="setSize" inputAction="conditional1.activate">
 <mw:input name="width" setAttribute="svgElement.width"/>
 <mw:input name="height" setAttribute="svgElement.height"/>
</mw:messageIn>
```

EXAMPLE 2 Receiving a message with no reply (scripting case)

```
<mw:messageIn name="setSize" inputAction="setwidgetSize">
 <mw:input name="width" scriptParamType="number"/>
 <mw:input name="height" scriptParamType="number"/>
</mw:messageIn>
```

In this example, the scene will have a setwidgetSize function which may look like:

```
function setwidgetSize(w,h) { … }
```

EXAMPLE 3 Receiving a message and sending a reply (declarative case)

```
<mw:messageIn name="setSize" inputAction="conditional1.activate"
              outputTrigger="conditional2.activate" >
 <mw:input name="width" setAttribute="svgElement.width"/>
 <mw:input name="height" setAttribute="svgElement.height"/>
 <mw:output name="actualWidth" attributeModified="svgElement.width"/>
 <mw:output name="actualHeight" attributeModified="svgElement.height"/>
</mw:messageIn>
```

EXAMPLE 4 Receiving a message and sending a reply (scripting case)

```
<mw:messageIn name="setSize" inputAction="setwidgetSize" >
 <mw:input name="width" scriptParamType="number"/>
 <mw:input name="height" scriptParamType="number"/>
 <mw:output name="actualWidth" scriptParamType="number"/>
 <mw:output name="actualHeight" scriptParamType="number"/>
</mw:messageIn>
```

In this example, the scene will have a setwidgetSize function which may look like:

```
function setwidgetSize(w,h) {

  aw = …; ah = … ;

  var interface = widget.getInterfaceHandlersByType(interfacetype)[0];

  var msgHandler = interface.msgHandlerFactory(“setSize”);

  interface.invokeReply(msgHandler, aw, ah);

}
```

10.17 The <mw:messageOut> element

Description:

The <mw:messageOut> element is used to group multiple <mw:output> elements and, possibly, multiple <mw:input> elements together. It is used to define how to create messages to be sent to external communication entities from multiple <mw:output> elements and how to be notified of a possible reply.

The output message may be triggered either declaratively, as specified in the mw:outputTrigger attribute or via the invoke method.

If the output trigger is a declarative construct (attribute change or event triggered), the output message parameter’s values are taken from the values indicated by the <mw:output> elements.

If the output trigger is not specified, the message may only be triggered upon a call to the invoke method. The values of the arguments of the invoke method are used, in the declaration order, to set the output message parameter values.

A sent message may lead to a reply. This reply may trigger scene modifications either declaratively or using scripting, as specified in the mw:inputAction attribute.

The reply message parameter’s values are dispatched to the scene according to the <mw:input> elements with the same name and the action specified in the mw:inputAction attribute is processed (attribute modification or script function call).

If the input action is a script function, the arguments of the function are the values of the <mw:input> elements in declaration order.

If the input action is a declarative construct (attribute change or event firing), the action is performed after all <mw:input> elements are processed.

If multiple <mw:messageOut> elements with the same "name" attribute value are declared within an interface, messages may be constructed and sent from any of the scene constructs identified in the <mw:output> elements or in the "outputTrigger" attribute of those messages.

NOTE For a given message name, results of reply processing are undefined if two messages are sent before the reply of the first message is received.

Context:

The <mw:messageOut> element is a child of an <mw:interface> element.

Expected children (in any order):

It contains first zero or more <mw:output> elements followed by zero or more <mw:input> elements.

Attributes:

`mw:name`

Optional. Indicates the name of the message to be sent.

`mw:outputTrigger`

Optional. Indicates the event which triggers the sending of the message. See 10.23.3 for the syntax.

`mw:inputAction`

Optional. Indicates the scene construct to change or trigger, or the name of a script function to call, upon receiving the reply. See 10.23.2 for the syntax.

EXAMPLE 1 Sending a message with no handling of the reply (declarative case)

```
<mw:messageOut name="messageA" outputTrigger="unloader.click">
 <mw:output name="name" attributeModified="eltAName.textContent"/>
 <mw:output name="value" attributeModified="eltBValue.textContent"/>
</mw:messageOut>
```

EXAMPLE 2 Sending a message with no handling of the reply (scripting case)

```
<mw:messageOut name="messageB" inputAction="processError">
  <mw:output name="name" scriptParamType="string"/>
  <mw:output name="value" scriptParamType="number"/>
</mw:messageOut>
```

In this example, the scene will contain some code like:

```
var interface = widget.getInterfaceHandlersByType(interfacename)[0];

var msgHandler = interface.msgHandlerFactory(“messageB”, null);

interface.invoke(msgHandler, “location”, “Paris”);
```

EXAMPLE 3 Sending a message and handling of the reply (declarative case)

```
<mw:messageOut name="messageC" outputTrigger="unloader.click"
               inputAction="conditional1.activate" >
 <mw:output name="name" attributeModified="eltAName.textContent"/>
 <mw:output name="value" attributeModified="eltBValue.textContent"/>
  <mw:input name="string" setAttribute="eltCResult.errorString"/>
 </mw:messageOut>
```

EXAMPLE 4 Sending a message and handling of the reply (scripting case)

```
<mw:messageOut name="messageD">
 <mw:output name="name" scriptParamType="string"/>
 <mw:output name="value" scriptParamType="number"/>
 <mw:input name="resultCode" scriptParamType="number"/>
 <mw:input name="resultString" scriptParamType="string"/>
</mw:messageOut>
```

In this example, the scene could contain some code like:

```
function processError(msgHandler, code, string) { … }

var interface = widget.getInterfaceHandlersByType(interfacetype)[0];

var msgHandler = interface.msgHandlerFactory(“messageD”, processError);

interface.invoke(msgHandler, “location”, “Paris”);
```

10.18 The <mw:input> element

Description:

This element identifies inputs into the associated representation of the widget. A widget manager may decide to communicate with a widget. Each input defines a name attribute whose value identifies a parameter in the message. The input direction is from the point of view of the widget, i.e. the data enters the widget.

If this element is used with a script, it shall not use the setAttribute attribute and may use the scriptParamType attribute.

If this element is used in a declarative construct, it shall not use the scriptParamType attribute and shall use the setAttribute attribute.

Context:

This is a child of a <mw:messageIn> or <mw:messageOut>elements.

Expected children (in any order):

None.

Attributes:

`mw:name`

Mandatory. Indicates the name of the associated message parameter.

`mw:default`

This attribute is optional with no default value. It indicates the default value that should be provided to the scene if the incoming message does not contain a value for the matching parameter (case of an optional parameter).

`mw:scriptParamType`

When used as a parameter within a mapping to a script function call, this attribute defines the type of the parameter on which this input is mapped.

`mw:setAttribute`

Optional. Indicates the attribute connected to this pin. See 10.23.2 for the syntax.

10.19 The <mw:output> element

Description:

This element identifies outputs out of the associated representation of the widget. A widget may communicate with external entities through the widget manager. Each output defines a name attribute, whose value identifies a parameter in the message. The name is intended to signify output from the widget point of view, i.e. the data exits the widget.

Context:

This is a child of a <mw:messageIn> or <mw:messageOut>elements.

Expected children (in any order):

None.

Attributes:

`mw:name`

Mandatory. Indicates the name of the associated message parameter.

`mw:default`

Optional. No default value. It indicates the value of this parameter in cases where there is no need for a connection to the scene tree of the widget and a constant value shall be provided upon activation of the message.

`mw:attributeModified`

Optional. No default value. Indicates the attribute in the widget representation whose value is sent out from this pin. See 10.23.2 for the syntax.

10.20 The <mw:component> element

Description:

This element identifies another widget that this widget depends on, called a component. Upon loading of this manifest, the widget manager shall get the manifest of the component widgets referenced by the mw:src attribute or try to find a widget that matches the interfaces declared by the <mw:requiredInterface> elements, without actually activating the component widget. If no available widget matches the required interfaces, the behaviour, and in particular the loading of the main widget, is implementation-specific. If a matching widget is found, events for the binding of this component shall be triggered after the events for the activation of that widget are triggered.

NOTE Depending on the actual situation, the activatedAction and deactivatedAction messages may not be received, for instance when the component was already activated before the parent requested the activation of the component.

EXAMPLE Components may be used to create template widgets as in the following. Widget A refers to a scene capable of visually grouping two objects. The manifest of A uses two components B and C, pointing to either specific widgets or to generic interfaces. Upon activation of B and C, A can be activated and its scene can indicate to the widget manager how to visually group B and C.

Context:

This is a child of a <content> element.

Expected children (in any order):

Zero or more <mw:requiredInterface> elements.

Attributes:

`mw:src`

Indicates the URI of a component widget. If the <mw:component> element has <mw:requiredInterface> children, this attribute is optional. If there are no children, then this attribute is mandatory. The value of this attribute may be “urn:uuid:” followed by the UUID of a widget (as specified in the attribute mw:uuid of the widget element).

`mw:id`

Mandatory. Indicates an ID for the component widget. The scope of this ID is limited to the widget.

`mw:activateTrigger`

Optional. Indicates the scene construct that is used to trigger the activation of the component widget. See 10.23.3 for the syntax.

`mw:deactivateTrigger`

Optional. Indicates the scene construct that is used to trigger the deactivation of the component widget. See 10.23.3 for the syntax.

`mw:activatedAction`

Optional. Indicates the name of a scene construct that is used to receive the notification of the successful activation of the component widget. See 10.23 for the syntax.

`mw:deactivatedAction`

Optional. Indicates the name of a scene construct that is used to receive the notification of the deactivation of the component widget. See10.23 for the syntax.

`mw:activateFailureAction`

Optional. Indicates the scene construct that is used to receive the notification of the failure of the activation of the component widget. See10.23.2 for the syntax.

10.21 The <mw:requiredInterface> element

Description:

This element identifies the interfaces that a component should implement.

Context:

This is a child of the <mw:component> element.

Children:

None.

Attributes:

`mw:type`

Mandatory. Identifies the type of the required interface.

10.22 Predefined interfaces and communications

This Subclause defines an interface, identified by the type "urn:mpeg:mpegu:schema:widgets:core:in:2010", which corresponds to a normative set of messages, that if declared in the widget manifest, the widget manager shall send. The messages in this interface are the followings.

```
<messageIn name="setSize">
 <input name="width" scriptParamType="number"/>
 <input name="height" scriptParamType="number"/>
 <input name="dpi" scriptParamType="number"/>
</messageIn>
```

This message is sent by the widget manager to a widget to inform it about the size in units of the local coordinate system and the dpi at which it is rendered.

NOTE This message can be used to inform about dynamic changes in the size of the widget.

```
<messageIn name="show"/>
```

This message is sent by the widget manager to a widget to inform it that it is visible.

NOTE If a single resource is used to represent both the iconic and full representation, it is possible to distinguish which representation is visible by declaring this message in the <mw:interface> element of the <icon> element.

```
<messageIn name="hide"/>
```

This message is sent by the widget manager to a widget to inform it that it is hidden.

```
<messageIn name="activate"/>
```

This message is sent by the widget manager to a widget to inform that it has been activated.

```
<messageIn name="deactivate"/>
```

This message is sent by the widget manager to a widget to inform it that it is being deactivated.

NOTE The amount of time that will elapse between the reception by the widget of this message and its actual deactivation is implementation-specific.

Similarly, the interface identified by the type "urn:mpeg:mpegu:schema:widgets:core:out:2010" defines a set of messages that a widget manager shall receive, process and, in some cases, answer. The messages in this interface are the followings.

```
<messageOut name="setSize">
 <output name="width" scriptParamType="number"/>
 <output name="height" scriptParamType="number"/>
</messageOut>
```

This message is sent by a widget to the widget manager to inform about the preferable display size in units of the local coordinate system.

```
<messageOut name="show"/>
```

This message is sent by a widget to the widget manager to request to be shown.

```
<messageOut name="hide"/>
```

This message is sent by a widget to the widget manager to request to be hidden.

```
<messageOut name="getAttention">
 <input name="returnCode" scriptParamType="number"/>
</messageOut>
```

This message is sent by a widget to request that the widget manager highlights or attracts the user attention to this widget. The widget manager shall reply using a returnCode parameter whose values are: 0 for failure of the request and 1 for success.

```
<messageOut name="requestActivate">
 <input name="returnCode" scriptParamType="number"/>
</messageOut>
```

This message is sent by the widget iconic representation to request that the widget manager activates this widget if not already activated. The widget manager shall reply using a returnCode parameter whose values are: 0 for failure of the request and 1 for success. This message shall only be declared in the <mw:interfaces> element of the <icon> element. If present in the interfaces element of the <content> element, it shall be ignored.

```
<messageOut name="requestDeactivate">
 <input name="returnCode" scriptParamType="number"/>
</messageOut>
```

This message is sent by a widget to request that the widget manager deactivates this widget. The widget manager shall reply using a returnCode parameter whose values are: 0 for failure of the request and 1 for success.

```
<messageOut name="showNotification">
 <output name="message" scriptParamType="string"/>
 <input name="returnCode" scriptParamType="number"/>
</messageOut>
```

This message is sent by a widget to request that the widget manager notifies the user with a message given as output parameter of the message. The widget manager shall reply using a returnCode parameter whose values are: 0 for failure of the request and 1 for success.

```
<messageOut name="placeComponent">
 <output name="componentID" scriptParamType="string"/>
 <output name="x" scriptParamType="number"/>
 <output name="y" scriptParamType="number"/>
 <output name="width" scriptParamType="number"/>
 <output name="height" scriptParamType="number"/>
 <output name="z-index" scriptParamType="number"/>
 <output name="transparency" scriptParamType="number"/>
 <output name="sticky" scriptParamType="boolean"/>
 <input name="returnCode" scriptParamType="number"/>
</messageOut>
```

This message is sent by a widget to request that the widget manager places a component whose ID is given in the componentID output parameter at the position given by the x and y output parameters, in an area of the given width and height, expressed in units of the local coordinate system of the root of the widget scene, and at the depth given by the z-index output parameter, a positive integer, relative to the depth of the requesting widget. The transparency output parameter indicates the request transparency, as decimal value between 0 (opaque) and 1 (fully-transparent). The sticky output parameter indicates whether the placement should be maintained permanently (or until the next placeComponent message is sent) or not.

The widget manager may have a placement policy that is incompatible with the execution of the placeComponent message, in which case it is ignored. If applied and sticky is true, the requested relative placement endures until the next placeComponent or until the parent widget is removed.

NOTE The widget manager and the widget may need to exchange information about the relationship between their respective coordinate systems prior to execution of placeComponent or setSize messages.

Predefined communication interface may also be used to provide access to the widget to user environment characteristics. This specification defines an interface, identified by the type "urn:mpeg:mpegu:schema:widgets:mpeg21ued:2010", that widget managers should support. Support for this interface should be indicated in the manifest using a <feature> element with the name "urn:mpeg:mpegu:schema:widgets:mpeg21ued:2010". The messages of this interface are described below:

```
<messageOut name="UserEnvironmentDescriptionRequest">
 <output name="name" scriptParamType="string"/>
 <input name="value" scriptParamType="string"/>
</messageOut>
```

This message is sent by a widget to request the widget manager to inform the widget about the value of a property of the environment as described in the MPEG-21 UED standard.

10.23 Connection between scene and interface constructs

10.23.1 Overview

Scene constructs (i.e. attributes, events, properties, text content or script functions) need to be connected with message parameters in the following cases:

i) An attribute or a property of an element may be set with the value coming from a communication entity, or its value may be sent to that entity.

ii) An element may receive an event from a communication entity, or notify that entity of an event.

10.23.2 Connection to or from an attribute or a property of an element

For element and attribute or property connection, the node ID concatenated with a “.” concatenated with an attribute or property name; the node ID may be a string or a number, depending on the scene description used; the attribute or property may be a string or an numeric attribute or property index, depending on the scene description used.

EXAMPLE myRect.width or node(10).attribute(5)

In the case of XML descriptions with namespaces, if the name of the attribute or property contains a namespace prefix (as in "xlink:href"), the prefix shall be interpreted within the scene description document whose URL is given in the src attribute of the content element in the configuration of the widget. The prefix shall not be interpreted within the configuration document.

For DOM-based presentation formats such as W3C HTML, W3C SVG or MPEG-4 LASeR, element attribute values refer to the DOM values. The special pseudo-attribute name “textContent” refers to the text content of a node as defined in SVG Tiny 1.2.

10.23.3 Connection from an event

For element and event connection, the node ID concatenated with a "." concatenated with an event ID; the node ID may be a string or a number, depending on the scene description used; the event ID may be a string or an numeric event index, depending on the scene description used. Additionally, the syntax defined in 10.23.2 is allowed and is equivalent to an event triggered when attribute or property value is changed.

EXAMPLE myRect.click or node(10).event(5) or myRect.width

11 Widget context information syntax

11.1 Overview

A widget context may be exchanged between widget managers using an XML description not embedded within the widget manifest. The widget context information language defined in this Part of ISO/IEC 23007 uses the namespace '`urn:mpeg:mpegu:schema:widgets:contextinfo:2010`'. The prefix '`ci:`' is used below. Manifest files shall be identified with the media type `application/mw-context+xml`. The XML syntax of this language is given in the attached schema file (context_information.xsd).

11.2 The <ci:contextInformation> element

Description:

This element enables the description of widget context information to be exchanged between widget managers.

Context:

This is the root element of a context information document.

Expected children (in any order):

One or more <preference> elements.

Attributes:

`ci:uuid`

Optional. If specified, this attribute indicates the unique identifier of the widget, regardless of the version, associated with this context information, as specified in the <widget> element.

`ci:version`

Optional. If specified, this attribute indicates the version of the widget associated with this context information, as specified in the <widget> element.

`ci:url`

Optional. If specified, this attribute indicates a URL where the widget associated with this context information may be retrieved.

12 Security considerations

12.1 Widget security

In this part of ISO/IEC 23007, it is assumed that the widget manager has means, outside of this part of ISO/IEC 23007, to know if the widget provider is trusted or not and to check the widget integrity. This can be done for example using secure communications between the widget provider and the widget manager and/or using technologies defined in [1].

12.2 Communication security

Widget manager may decide to give communication privileges to widgets depending on the trustiness of its source. For example, if a widget is received from a non trusted source, the widget manager may only accept communications back to the server of origin. If the widget is received from a trusted source, the widget manager may modulate the right to connect according to a “trusted” status of the widget. The exact privilege policy is implementation specific.

Annex A
(normative)

MIME type registration for application/mw-manifest+xml

Type name:

application

Subtype name:

mw-manifest+xml

Required parameters:

none.

Optional parameters:

none.

Encoding considerations:

XML general encoding considerations.

Security considerations:

The manifest identified by this media type is based on the XML configuration document specified by the W3C "Widgets 1.0: Packaging and Configuration", so the security considerations of XML applies, as detailed in RFC 3023.

Interoperability considerations:

Platform neutral, though some issues can arise with regards to character encodings in file names.

Published specification:

ISO/IEC 23007-1

Applications that use this media type:

Use within widget managers that claim conformance to this specification.

Magic number(s):

None.

File extension(s):

xml

Macintosh file type code(s):

none

Person & email address to contact for further information:

gen-sys@lists.uni-klu.ac.at

Intended usage:

Common.

Restrictions on usage:

None.

Author:

The Moving Pictures Expert Group (MPEG).

Annex B
(normative)

MIME type registration for application/mw-context+xml

Type name:

application

Subtype name:

mw-context+xml

Required parameters:

none.

Optional parameters:

none.

Encoding considerations:

XML general encoding considerations.

Security considerations:

The context information identified by this media type is based on XML so the security considerations of XML applies, as detailed in RFC 3023.

Interoperability considerations:

Platform neutral.

Published specification:

ISO/IEC 23007-1

Applications that use this media type:

Use within widget managers that claim conformance to this specification.

Magic number(s):

None.

File extension(s):

xml

Macintosh file type code(s):

none

Person & email address to contact for further information: