INTERNATIONAL STANDARD

ISO/IEC 14496-21

First edition
2006-11-15

Information technology — Coding of audio-visual objects —

Part 21: MPEG-J Graphics Framework eXtensions (GFX)

Technologies de l'information — Codage des objets audiovisuels —

Partie 21: Extensions du cadre graphique (GFX) pour MPEG-J

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

Reference number
ISO/IEC 14496-21:2006(E)

© ISO/IEC 2006

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

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

 © ISO/IEC 2006 – All rights reserved

Contents

Page

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

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 14496-21 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 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

— *Part 6: Delivery Multimedia Integration Framework (DMIF)*

— *Part 7: Optimized reference software for coding of audio-visual objects* [Technical Report]

— *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*

— *Part 9: Reference hardware description* [Technical Report]

— *Part 10: Advanced Video Coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

— *Part 13: Intellectual Property Management and Protection (IPMP) extensions*

— *Part 14: MP4 file format*

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

— *Part 15: Advanced Video Coding (AVC) file format*

— *Part 16: Animation Framework eXtension (AFX)*

— *Part 17: Streaming text format*

— *Part 18: Font compression and streaming*

— *Part 19: Synthesized texture stream*

— *Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)*

— *Part 21: MPEG-J Graphics Framework eXtension (GFX)*

— *Part 22: Open Font Format*

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

Information technology — Coding of audio-visual objects —

Part 21:
MPEG-J Graphics Framework eXtensions (GFX)

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

1 Scope

This International Standard specifies MPEG-J Graphics Framework eXtension (GFX). This extension enables Java-based applications to control the rendering and composition of synthetic and natural media in a programmatic manner.

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 14496-1:2004, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-11:2005, *Information technology — Coding of audio-visual objects — Part 11: Scene description and application engine*

JSR-135, *Mobile Media API (MMAPI)* — http://jcp.org/aboutJava/communityprocess/final/jsr135/index.html

3 Symbols and abbreviated terms

List of symbols and abbreviated terms.

API	Application Programming Interface
BIFS	BInary Format for Scenes
ES	Elementary Stream
IOD	Initial Object Descriptor
JCP	Java Community Process
JSR	Java Specification Request
M3G	Mobile 3D Graphics API for Java
MPEG-J	MPEG-4 Java Application Engine
OD	Object Descriptor

4 Notations

The UML (Unified Modelling Language) notation [18] is used extensively in this specification for class, sequence, collaboration, state and component diagrams.

© ISO/IEC 2006 — All rights reserved

5 MPEG-J Graphics Framework eXtension

5.1 Introduction

In an MPEG-4 terminal, multiple media are composed to create a final image displayed on its screen. These media may be synthetic (e.g. made by a computer such as vector graphics) or natural (e.g. audio and video captured from a sensor). Composition of visual media to produce a final image is achieved, for each frame, by rendering instructions.

In ISO/IEC 14496-11, the BIFS scene description describes rendering and composition operations in a structured manner using a tree or scene graph. The application engine enables programmatic access to terminal resources and interacts with the scene description to arrange rendering and composition operations based on the application's logic. However, the application has no direct access to rendering or composition operations; rather the terminal interprets the operations described in the scene graph and performs some rendering operations.

In this document, ISO/IEC 14496-21, the Java application engine is extended with direct access to rendering and composition operations. This enables applications to optimize organization of such operations based on their logic and to produce visual effects not possible with a descriptive language such as BIFS. Note that ISO/IEC 14496-21 application engine reuses interfaces defined in ISO/IEC 14496-11, however some of them are revised.

In this specification, two rendering APIs are selected as a recommended practice, a low-level graphics API (JSR-239 Java Bindings to OpenGL ES [1][3]), and an API with higher level constructs such as scene graphs and animation (JSR-184 Mobile 3D Graphics API for Java [4]). Alternatively, an implementer may choose a proprietary rendering API. The responsibility of ensuring the behaviour of the proprietary API calls is outside the scope of this specification. In other words, if one or both of the JSR-239 and JSR-184 APIs are chosen then this specification defines normatively how such implementations interact with the renderers.

Figure 1 depicts the block organization of systems and APIs in an MPEG-4 terminal using the specification in this document.

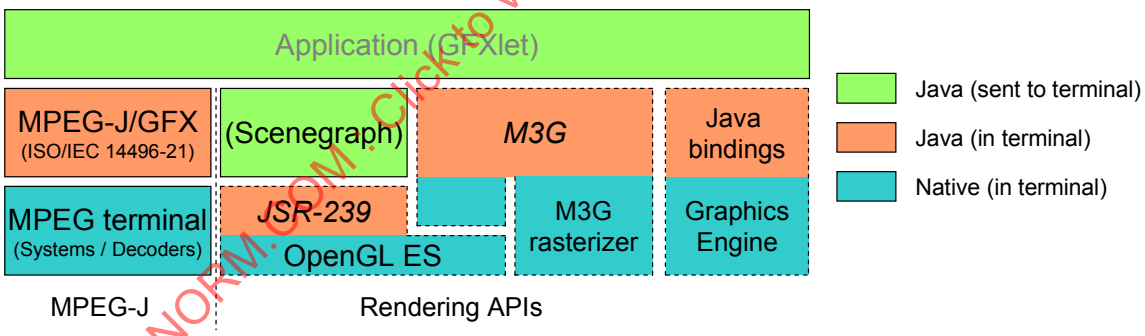


Figure 1 — Block diagram of an MPEG-4 Player with MPEG-J extensions for rendering.

NOTE 1 The two Java APIs defined by the JSR-239 and JSR-184 expert groups may be implemented on top of OpenGL ES. Some implementations may use a custom renderer tailored for M3G, instead of OpenGL ES.

NOTE 2 Typically, an MPEGlet - or application as called in this document - may define its own scene graph APIs built upon JSR-239, may use an API similar to ISO/IEC 14496-11 BIFS built upon JSR-239, use JSR-184 rich and lightweight scene graph API, or use any rendering engine available in the terminal. Through MPEG-J API and the extensions defined in this document, an application can interact with other resources in an MPEG terminal.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

5.2 Architecture

5.2.1 Overview

Figure 2 shows the typical workflow in an MPEG-4 terminal, following ISO/IEC 14496-1 and ISO/IEC 14496-11. From left to right, a multiplexed stream is received by the demultiplexer. The demultiplexer splits the stream in elementary streams that are decoded by decoders. MPEG-4 defines decoders for audio, video, and MPEG-J among others. The MPEG-J decoder receives Java classes or archives and launches those implementing MPEGlet interface in their own thread of execution and namespace. Once launched, the MPEGlet application can

- Issue rendering and compositing commands
- Control media retrieval and playback

While in ISO/IEC 14496-11 MPEGlets could only access rendering and compositing operations via the BIFS scene graph, in this specification, MPEGlets may issue graphic commands on the graphic context of the terminal output device. An application can query the rendering APIs available in the terminal and select the most appropriate one for its needs. Behaviour of the calls to rendering APIs is outside of the scope of this specification.

On the native side, software and hardware video decoders output pixel arrays per frame that refresh a texture object in the renderer's fast texture memory. These texture objects can be accessed and mapped onto 3D surfaces at any time as directed by the application. This enables any type of composition and effects using texture addressing, texture mapping, and blending operations among others.

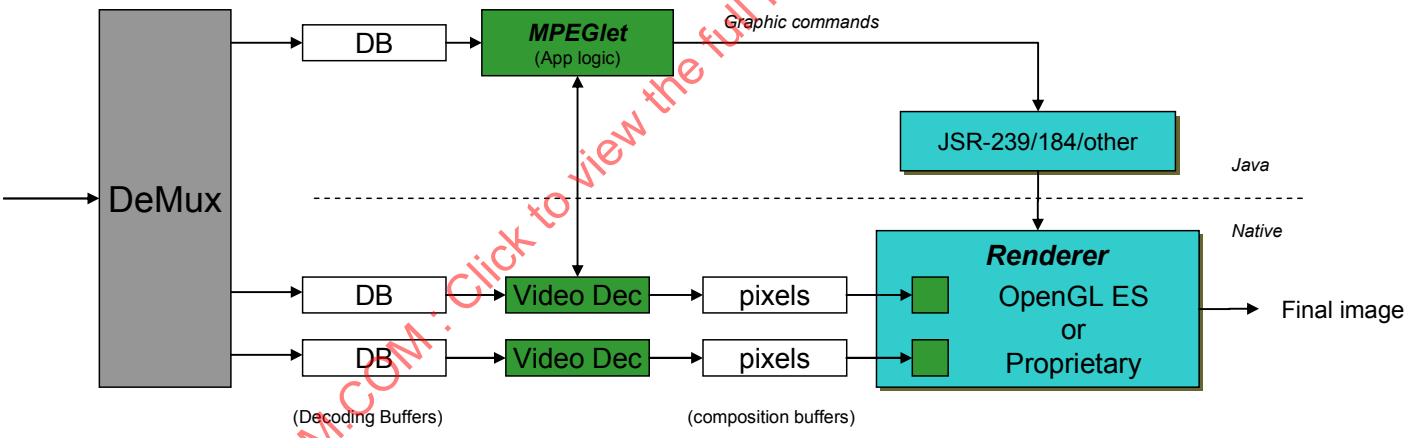


Figure 2 — Conceptual workflow.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

5.2.2 Systems interaction

The Java classes comprising the application define its logic and media must be retrieved from elementary streams. MPEG-4 Systems Object Descriptor framework defines many possibilities to interact with the streams flowing into a terminal. However, from an application point of view, higher-level functionalities are preferable:

- Connect to a media location using a protocol,
- Control of the playback of media (e.g. play, pause, stop a video stream and its associated audio),
- Retrieve the output of a stream for composition on the terminal's output,
- Possibly, control the post-processing of media

© ISO/IEC 2006 — All rights reserved

In this specification, such an abstraction is represented by the concepts of **DataSource**, **Player**, and **Control**s originally developed for Java Media Framework specification [7] and reused in Mobile Media API specification:

- **DataSource** abstracts protocol handling,
- **Player** abstracts content handling,
- **Control** provides a way to interact with the Player's processing.

Figure 3 provides a conceptual view of the interaction between **DataSource**s, **Player**s, **Renderer**s, and **MPEGlet**. **DataSource**s, **Player**s, and streams may expose controls for the MPEGlet.

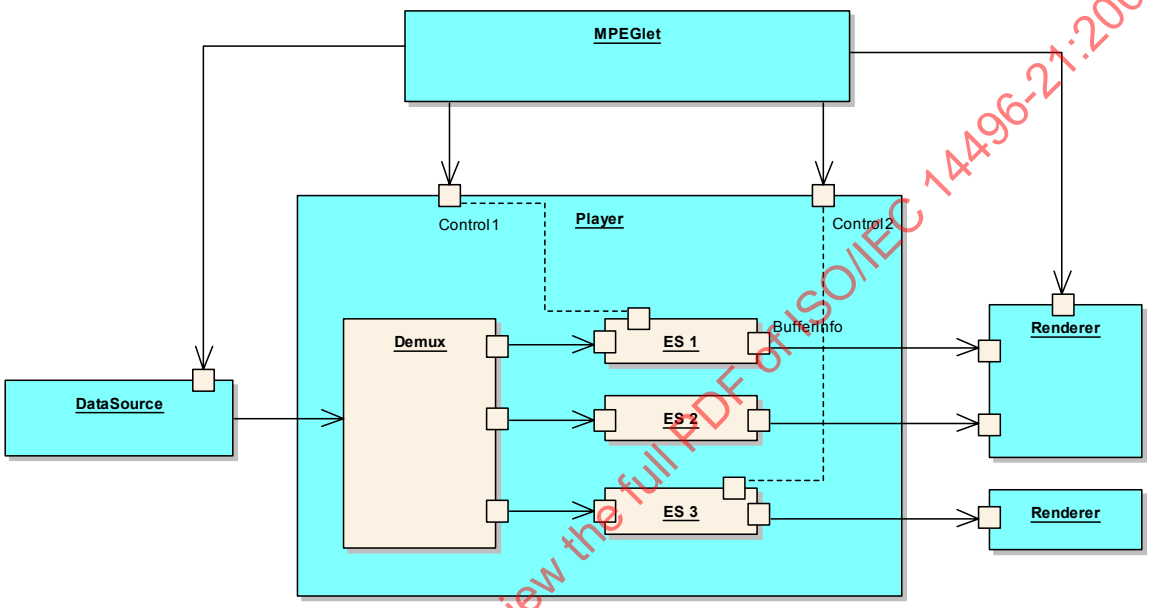


Figure 3 — Conceptual view of the terminal from an application.

Elementary streams (ES) output composition buffers abstracted by a **BufferInfo** interface that enables access to a GL**Buffer** interface that wraps native composition data:

— For a video decoder, GL**Buffer** wraps a byte array of pixels in an optimized format for the graphic card.

— For other decoders, specific **BufferInfo**s may be defined.

5.2.3 Contexts

An application communicates with the terminal resources via contexts. A context typically encapsulates the state management for a device. The application manager may run multiple applications at once, each with its own contexts but only one context can be active at a time for a device and, in general, a context is valid for one thread of execution.

In this specification, the following contexts are discussed (but not restricted to these contexts only):

- Application context or **MPEGletContext** – enables the application (MPEGlet) to communicate with the application manager within the terminal.
- Rendering contexts – to access graphic resources (e.g. OpenGL driver) and audio resources.
- System contexts – to access stream information

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

5.3 Static view

5.3.1 GFX MPEGlet architecture

Figure 4 depicts the GFX MPEGlet architecture. The application manager loads an MPEGlet and calls ***MPEGlet.init(MPEGletContext ctx)***. An MPEGlet is a **Runnable** because multiple MPEGlets may run in parallel and the application manager handles issues such as threading or switching between MPEGlets; from the terminal point of view, an MPEGlet is akin to a task.

The **MPEGlet** interface has the following methods:

- ***void init(MPEGletContext context)* -** called when the MPEGlet is loaded the first time. The context is provided by the application manager.

- ***void pause(), stop(), run(), destroy()* -** called by the application manager to notify the MPEGlet about state changes. The method ***run()*** is inherited from **Runnable** interface and is the "main loop" of the application. See subclause 5.4.1 for a description of MPEGlet states.

The **MPEGletContext** provides access to terminal resources and application state management and has the following methods:

- ***Object getDisplay()*** - returns **javax.microedition.lcdui.Display** for MIDP and **java.awt.Frame** for other Java profiles. This enables the application to add its own graphics components into the area provided by the terminals. These components can be Java components (e.g. Canvas, Graphics, Image) or Renderers using Java components as defined in this specification. **DisplayNotAvailableException** may be thrown if a display can not be granted at this time.

- ***String getProperty(String key)*** - returns the value of a property within the terminal or from the application descriptor of the application (see subclause 5.7). NULL is returned if the key doesn't exist. For renderers, if a named renderer exists, this method returns the version of the renderer.

- ***int checkPermission(String permission)*** - gets the status of the specified permission. If no API on the device defines the specific permission requested then it must be reported as denied. If the status of the permission is not known because it might require a user interaction then it should be reported as unknown. It returns 0 if the permission is denied; 1 if the permission is allowed; -1 if the status is unknown

- ***TerminalContext getTerminalContext()*** - returns a **TerminalContext** to access resource and network managers. MPEG-4 terminals return **MPEGContext**, an MPEG-4 specific sub-class of **TerminalContext** that provides access to the initial object descriptor, object descriptors and the scene manager for the content.

- ***void requestResume()* -** requests the terminal to resume the application (see subclause 5.4.2).

- ***void requestPause()* -** requests the terminal to pause the application (see subclause 5.4.2).

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

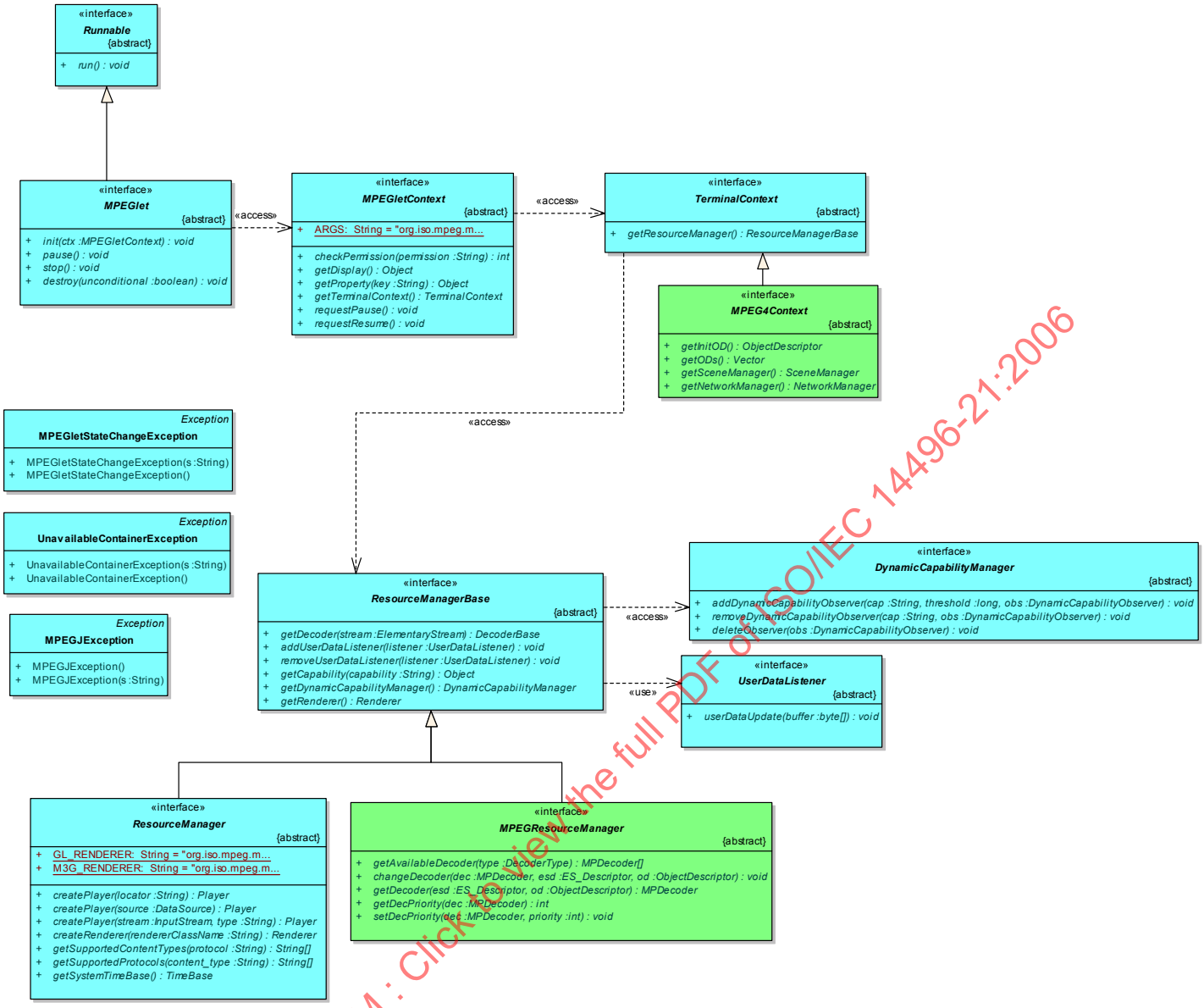


Figure 4 — GFX MPEGlets architecture.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

5.3.2 Eg.Terminal Contexts

5.3.2.1 TerminalContext

TerminalContext interface enables access to the **ResourceManagerBase**

— ***ResourceManagerBase getResourceManager()*** - returns the **ResourceManagerBase** associated with this **MPEGlet**. The object returned may support either or both of **ResourceManager** and **MPEGResourceManager** interfaces. An MPEG-4 terminal usually supports both interfaces.

5.3.2.2 MPEG4Context

MPEG4Context interface extends **TerminalContext** with access to MPEG-4 Systems specific stream information

© ISO/IEC 2006 — All rights reserved

— ***ObjectDescriptor getInitOD()*** - returns the initial object descriptor of this content.

— ***Vector getODs()*** **-** returns the list of object descriptors available in this content.

— ***SceneManager getSceneManager()*** - returns the (BIFS) scene manager.

— ***NetworkManager getNetworkManager()*** **-** returns the (DMIF) network manager.

5.3.3 Resource manager

ResourceManagerBase is a super interface for **ResourceManager** and **MPEGResourceManager**. **ResourceManager ResourceManager** is the central interface between an MPEGlet and the terminal's resources. **MPEGResourceManager** provides similar features for terminals that offer access to MPEG-4 Systems.

ResourceManager enables creation of Players from **DataSource** and provides convenient methods for creating players from locators and input streams. It enables creation of Renderers, access to the **RecordStore** and to the system's time base.

Due to extensibility restrictions in javax.microedtion.media.Manager, ResourceManager copies all methods and semantics of javax.microedition.media.Manager. ResourceManager provides MPEGlet- dependent resource contexts management and calls javax.microedition.media.Manager.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

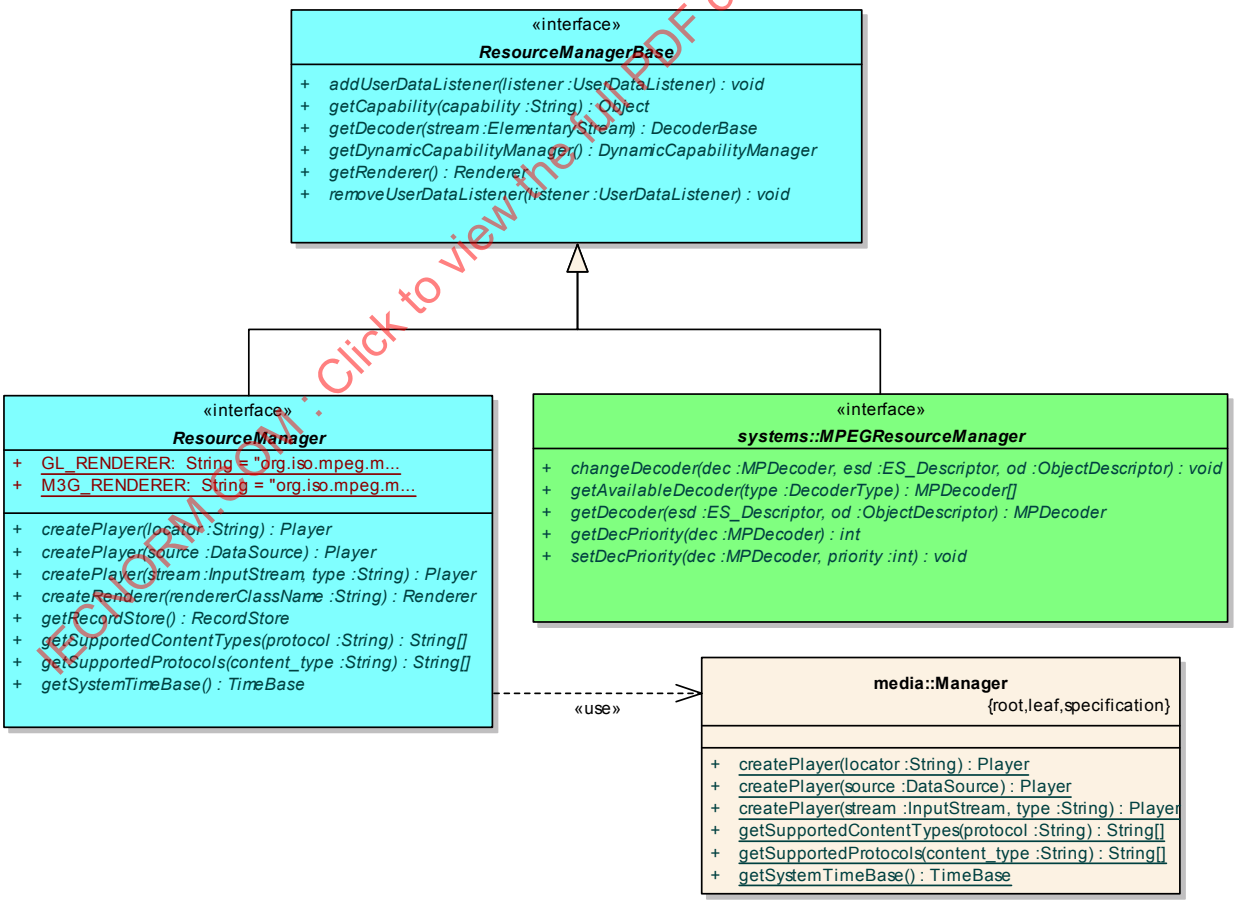


Figure 5 — ResourceManager and its factory methods.

© ISO/IEC 2006 — All rights reserved

5.3.3.1 ResourceManagerBase

ResourceManagerBase interface defines the following methods:

- ***DecoderBase getDecoderBase(ElementaryStream stream)*** - returns the decoder associated with an **ElementaryStream** stream.
- ***void addUserDataListener(UserDataListener listener)*** **-** registers a UserDataListener to listen to user data in Java stream header. See subclause 5.6.
- ***void removeUserDataListener(UserDataListener listener)*** - deregisters a UserDataListener. See subclause 5.6.
- ***Object getCapability(String capability)*** **-** retrieves a static capability. See subclause 0.
- ***DynamicCapabilityManager getDynamicCapabilityManager()*** - enables access to dynamic capabilities. See subclause 0.
- ***Renderer getRenderer()*** returns the current Renderer. See subclause 5.3.4 on using renderers.

5.3.3.2 ResourceManager

ResourceManager interface defines the following methods:

- ***Player createPlayer(String locator)*** - creates a Player from a locator, which takes the form of a Uniform Resource Identifier [14] i.e. a string of the form **<protocol>://<protocol-specific-part>** for example: http://server.com/movie.mp4. To access a media from an object descriptor, the syntax "od://<od_id>" is used with <od_id> replaced by the appropriate object descriptor identifier. RTP locators are defined by RFC 1889 [13].
- ***Player createPlayer(DataSource source)*** - creates a Player for a **DataSource**.
- ***Player createPlayer(InputStream stream, String content_type)*** **-** creates a Player to playback a media from an InputStream given its content-type.
- ***String[] getSupportedProtocols(String content_type)*** - returns the list of protocols supported for a content type.
- ***String[] getSupportedContentTypes(String protocol)*** - returns the list of supported content types for the given protocol such as file, HTTP, RTP. Content-types follow MIME types syntax (RFC 2045 [13], RFC 2046 [16]). Registered MIME media types are administered by the Internet Assigned Numbers Authority (IANA) at http://www.iana.org/assignments/media-types/.
- ***TimeBase getSystemTimeBase()*** **-** returns the system time-base, which is a continuously ticking source of time in the system.
- ***RecordStore getRecordStore()*** - returns **RecordStore** object.
- ***Renderer createRenderer(String name)*** - creates a Renderer given its fully qualified class name. See subclause 5.3.4.

And the following attributes to help refer to renderers mentioned in this specification:

- GLRENDERER = "org.iso.mpeg.mpegj.renderer.GLRenderer"
- M3GRENDERER = "org.iso.mpeg.mpegj.renderer.M3GRenderer"

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

5.3.3.3 MPEGResourceManager

MPEGResourceManager interface enables access to MPEG-4 systems specific information. See subclause 5.3.7 for more information about systems API.

NOTE this API is identical to ResourceManager class defined in ISO/IEC 14496-11.

- ***MPDecoder[] getAvailableDecoder(DecoderType type)*** – returns the list of available decoders in the terminal for a decoder type.
- ***void changeDecoder(MPDecoder dec, ES_Descriptor esd, ObjectDescriptor od)*** – change the decoder associated to an ES_Descriptor and an ObjectDecriptor. Note that the MPDecoder must be of the same type as the already attached decoder (e.g. an audio decoder cannot be replaced by a video decoder).
- ***MPDecoder getDecoder(ES_Descriptor esd, ObjectDescriptor od)*** – returns the MPDecoder associated with an ES_descriptor and ObjectDescriptor.
- ***int getDecPriority(MPDecoder dec)*** – returns the priority of a MPDecoder.
- ***void setDecPriority(MPDecoder dec, int priority)*** – sets the priority of a MPDecoder to the specified priority.

5.3.4 Renderer design

5.3.4.1 Creating renderers

The **Renderer** interface is available via ***ResourceManager.createRenderer(String rendererName)***. This specification defines two interfaces, for Java Bindings to OpenGL ES (**GLRenderer**) and M3G (**M3GRenderer**), which enable access to the graphic context of these specifications, as shown in Figure 6. The **rendererName** argument is the same name used in the application descriptor's **Renderer-Class** in subclause 5.7 and in the **Terminal-Renderers** property in subclause 5.8. The following renderer names are defined for OpenGL ES and M3G:

- **org.iso.mpeg.mpegj.gfx.M3GRenderer** for M3G renderer
- **org.iso.mpeg.mpegj.gfx.GLRenderer** for OpenGL ES renderer

If a named renderer is not available in the terminal, the ***ResourceManager.createRenderer()*** method throws a **RendererException**.

Any other renderer can extend **Renderer** interface to provide access to its graphic context. An application can indicate which renderer it uses using the application descriptor (see subclause 5.7).

NOTE Renderers recommended by this specification are typically heavyweight components for performance reasons. However, this specification doesn't preclude usage of lightweight renderers (e.g. pure Java ones).

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

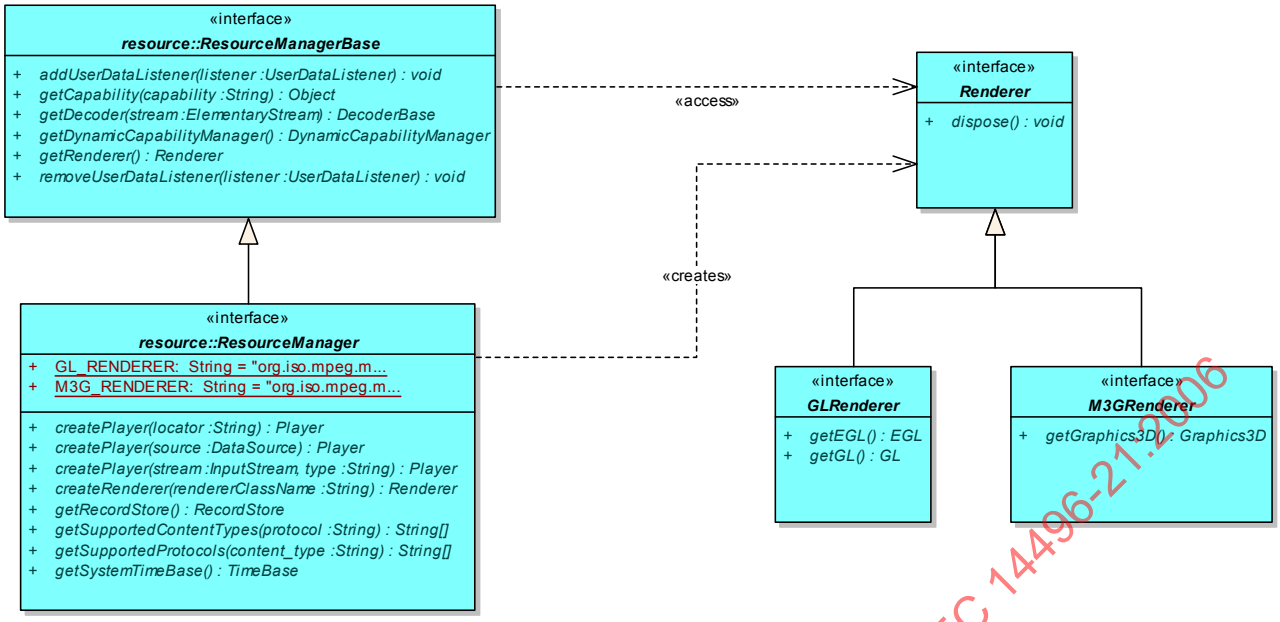


Figure 6 — Renderers class diagram.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

5.3.4.2 Using renderers

An MPEGlet application uses one renderer attached to a display. If the application calls ***ResourceManager.createRenderer()*** more than one time, the terminal does not create new renderers and it returns the renderer created the first time.

An MPEGlet application may use one renderer per display at a time. In order to switch to another renderer for the same display, the terminal must terminate the current renderer, which is only possible if there is no reference to it or if it is disposed of (see subclause 5.3.4.3).

In ISO/IEC 14496-11, a BIFS compositor is created when the terminal receives a BIFS stream. This compositor is optimized for a specific internal renderer. In ISO/IEC 14496-11, an MPEGlet application cannot access the renderer but it can send and receive events from the compositor.

In this specification, ISO/IEC 14496-11 model is extended for MPEGlet applications to provide their own BIFS compositor and/or renderers. The terminal receives a stream of BIFS commands and apply them onto a **Scene** object.

When an MPEGlet calls ***ResourceManager.createRenderer()*** *successfully on a terminal that is already receiving a BIFS stream as specified in* ISO/IEC 14496-11, the terminal hands over the control of rendering to the MPEGlet. No internal update to rendering is done from this point onwards by the terminal. The application is expected to handle rendering from that point. If the application releases its renderer and BIFS stream is still being received, then no rendering happens.

When the MPEGlet uses terminal's internal compositor and renderer the behaviour of the application is similar to ISO/IEC 14496-11 MPEGlets and the scene controller mechanism may be used for frame-by-frame scene modifications.

5.3.4.3 Disposing of renderers

A renderer is typically composed of a lightweight Java interface to native resources. These resources must be disposed of correctly before switching to another renderer or terminating an application. This cleanup can occur implicitly or explicitly:

© ISO/IEC 2006 — All rights reserved

— Implicit renderer disposal – the Java renderer is garbage collected by the Java Virtual Machine's garbage collector, which in turn releases the native resources.

— Explicit renderer disposal – ***Renderer.dispose()*** is called, thereby forcing disposal of all native resources. After this call, the Java renderer object may not be garbage collected yet but calls to this renderer will produce a **RuntimeException**.

5.3.5 Media API

The media API is modelled after JSR-135 Mobile Media API (MMAPI). By supporting MMAPI, JSR-234 Advanced Multimedia Supplements API (AMMS) and other APIs built upon MMAPI may be available to MPEGlets for even richer contents and experience.

Within the context of MPEG-4 terminals, the MMAPI encapsulates MPEG-4 Systems objects and provides high-level controls to interact with such objects. Since MMAPI has been designed with mobile phones specific features, this specification proposes the following features to be optional or not supported as follows:

- Audio-visual capture and encoding are optional
- MIDI and tone support are optional
- **GUIControl** and its derivative **VideoControl** are not supported because they may conflict with usage of Renderers
- MMAPI's **Manager** factory class is supported and its methods are copied to **ResourceManager** that provides MPEGlet-dependent resource management.

In addition, MMAPI has been enhanced with the following features:

- Elementary stream composition buffer access
- Rendering control

Figure 7 and Figure 8 show the API with the above features.

Controls defined in JSR-135 are reused except those for video rendering that have been replaced by GFX **VideoRendererControl** to attach a **Renderer** and graphical resources to a **Player**, **StreamControl** for accessing elementary streams output, and **DecoderControl** to control a decoder attached to an elementary stream.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

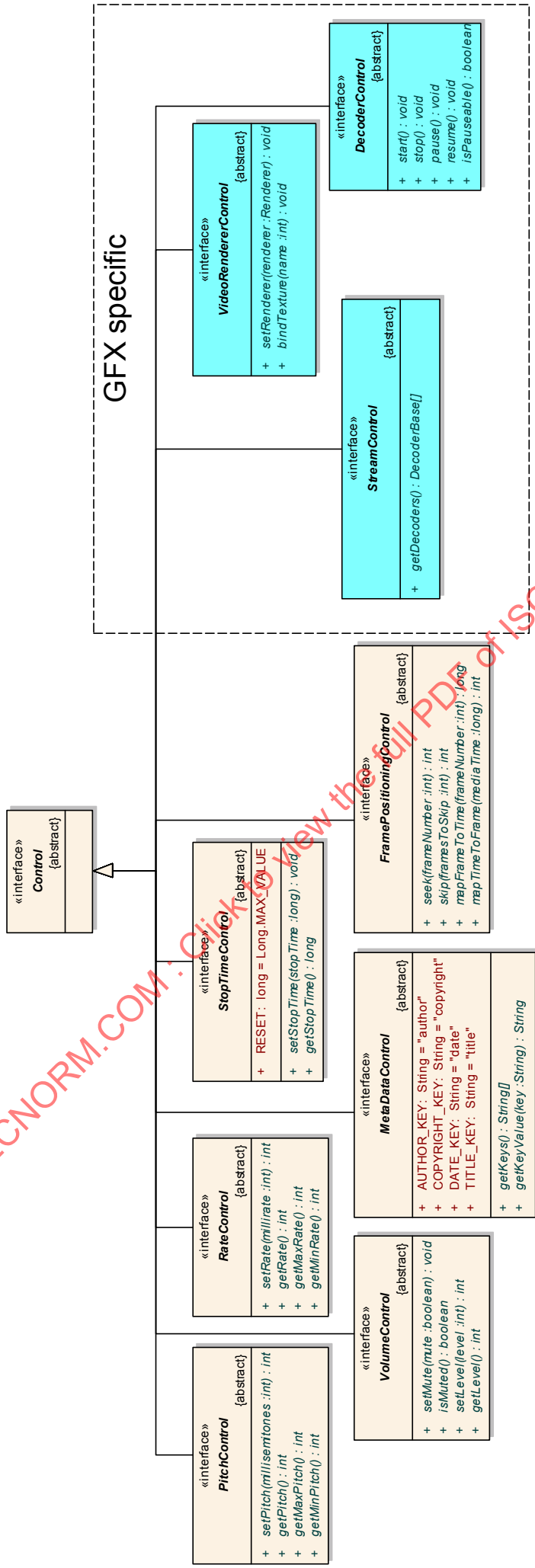


Figure 7 — Predefined controls. Blue interfaces are defined in this specification, other interfaces are defined in MMAPI specification.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

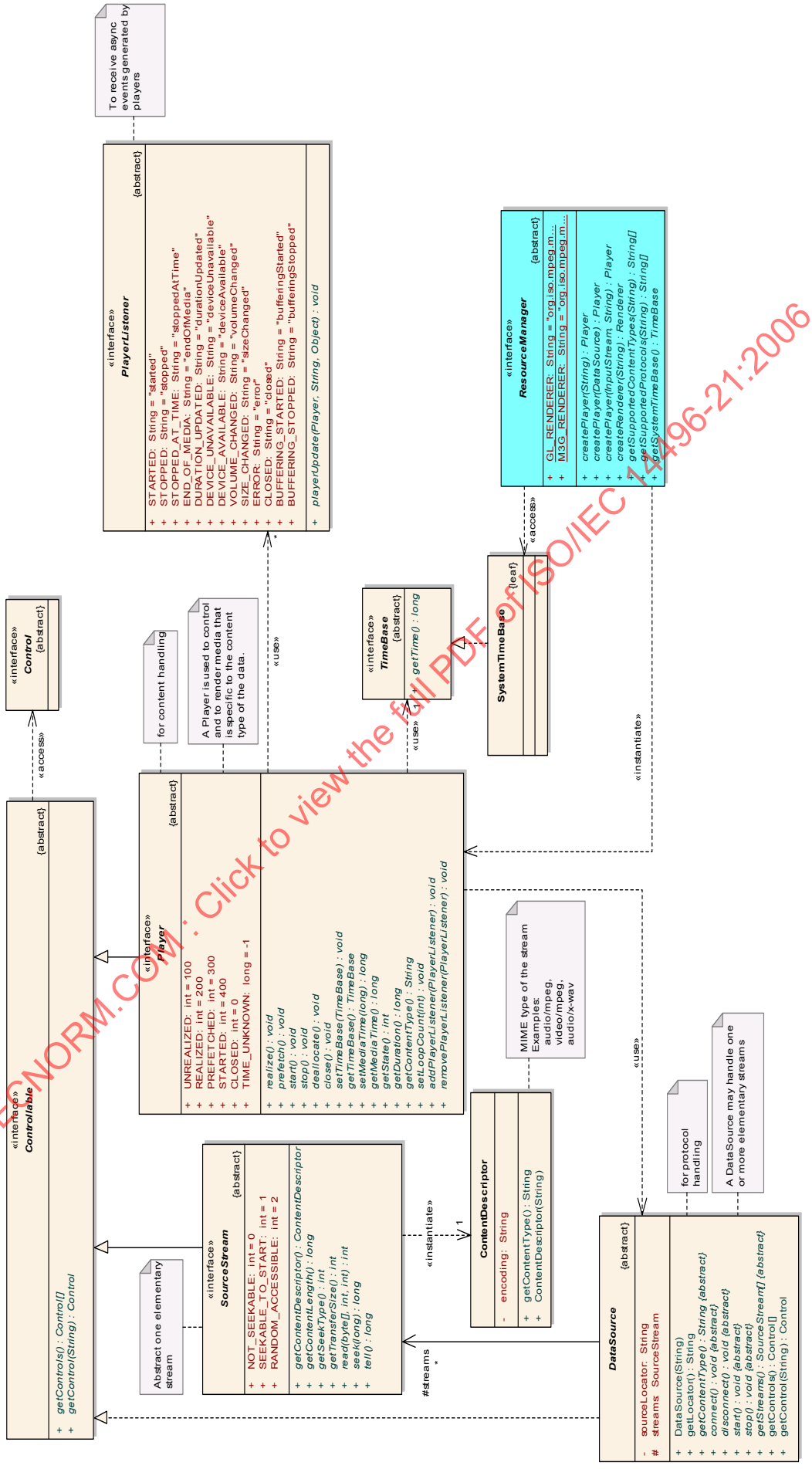
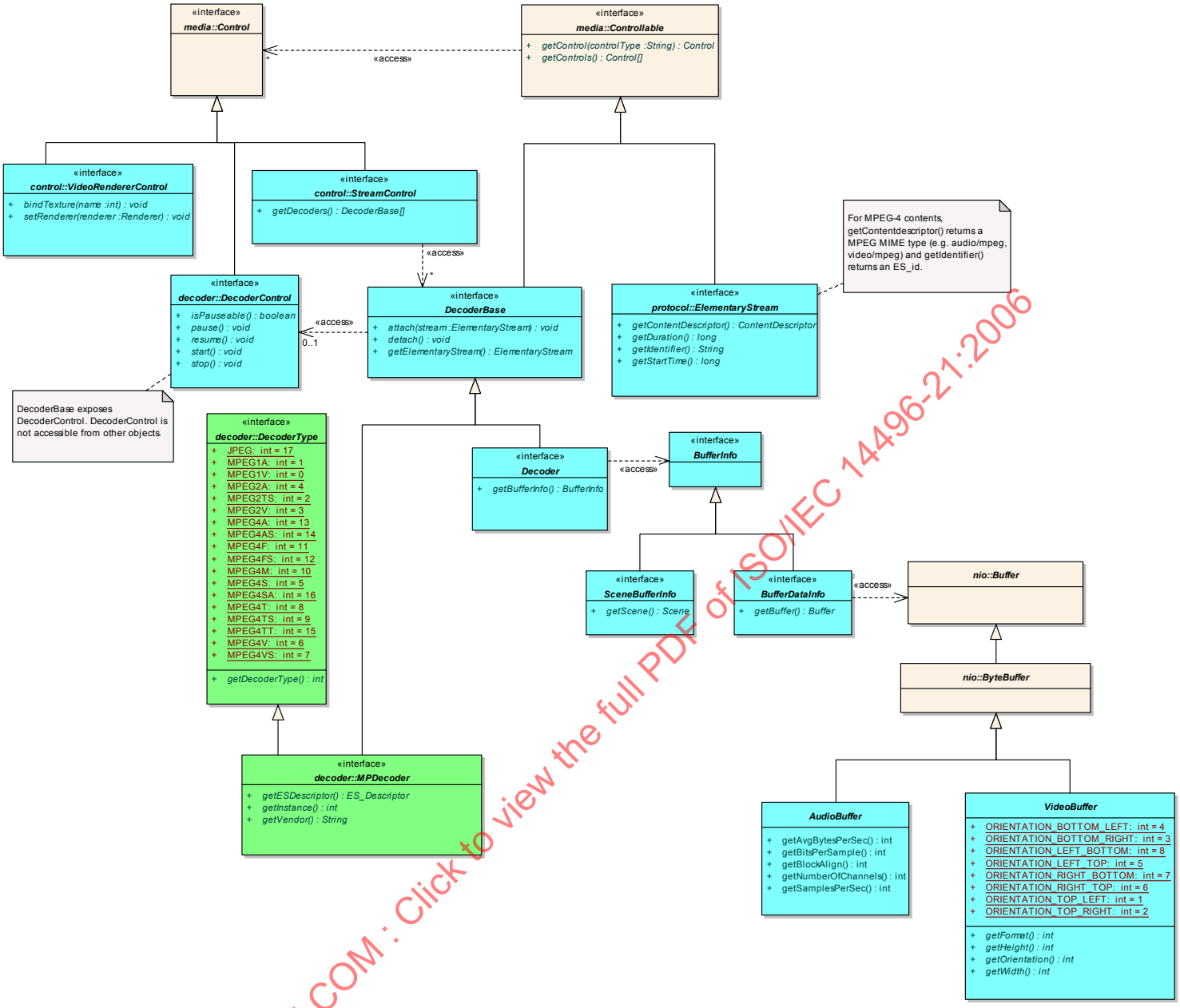


Figure 8 — GFX Media API. Except for ResourceManager, all other classes are from MMAPI.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved



IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

Figure 9 — Elementary stream information access. Classes in blue are defined in this specification, classes in orange are defined in MMAPI, classes in green are MPEG-4 specific.

5.3.5.1 VideoRendererControl

The **VideoRendererControl** interface enables a video **Renderer** to be attached to a **Player** and to assign a name to the video stream output of this Player:

— ***void setRenderer(Renderer renderer)*** – attaches a Renderer to this Player.

— ***void bindTexture(int name)*** – assigns a name to the video output of this Player. For OpenGL ES Renderer, it is the GL texture name obtained by, for example, ***glGetTextures()***. For M3G, it could be the userId of a Texture2D object (this requires the application to ensure desired texture has correct id).

© ISO/IEC 2006 — All rights reserved

5.3.5.2 StreamControl

The **StreamControl** interface enables access to the list of **DecoderBase** and there is one **DecoderBase** per stream.

— ***DecoderBase[] getDecoders()***– returns the list of decoders used by this Player or null if no decoder has been created.

5.3.5.3 Decoder

With the **Decoder** interface, a decoder object can expose the **BufferInfo** that provides information about the composition data of an elementary stream:

— ***BufferInfo getBufferInfo()*** – returns the composition buffer of this decoder

5.3.5.4 SceneBufferInfo

For BIFS streams, a **SceneBufferInfo** interface extends **BufferInfo** and enables access to the BIFS scene via ISO/IEC 14496-11 Scene API:

— ***Scene getScene()***– returns the BIFS scene

5.3.5.5 BufferDataInfo

The **BufferDataInfo** interface allows access to native composition data for elementary streams that enable this feature:

— ***getBuffer()*** returns a **GLBuffer** object that is a wrapper around a native memory area

NOTE **GLBuffer** is defined in JSR-239 *Java Bindings to OpenGL ES* package **javax.microedition.opengl_es.buffer**. GLBuffer and derived classes are generic native memory wrappers not specific to OpenGL ES.

5.3.5.6 VideoBuffer

For image or video data, ***getBuffer()*** returns a **VideoBuffer** that has the following methods:

— ***int getFormat()*** returns the format of the texture image data following GL conventions for glTexImage2D(). Table 1 summarizes the texture formats defined in OpenGL ES.

Table 1 – GL texture formats

Format	Number of components per pixel
GL_ALPHA	1
GL_RGB	3
GL_RGBA	4
GL_LUMINANCE	1
GL_LUMINANCE_ALPHA	2

NOTE Conversion from decoded video format (typically YUV) to the format of the renderers texture object is as specified by the renderer.

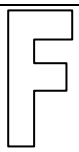
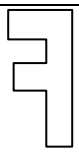
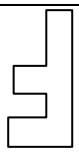
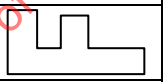
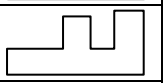
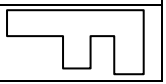
— ***int getWidth()*** and ***int getHeight()*** return the dimensions of the texture image

— ***int getOrientation()*** returns the orientation of the camera that took the image relative to the scene. The relation of the '0th row' and '0th column' to visual position is shown as below. Note that the numbering follows EXIF standard [17]:

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

Table 2 — Image orientation.

Value	*0^{th} row*	*0^{th} column*	*example*
1	Top	Left	
2	Top	Right	
3	Bottom	Right	
4	Bottom	Left	
5	Left	Top	
6	Right	Top	
7	Right	Bottom	
8	Left	Bottom	

NOTE M3G **Image2D** class is extended to support VideoBuffers as follows. The constructor ***Image2D(int format, Object image)*** is extended to support Object to be a **VideoBuffer**.

5.3.5.7 AudioBuffer

For audio data, ***getBuffer()*** returns an **AudioBuffer** that has the following methods:

— ***int getNumberOfChannels()*** – returns the number of audio channels (1 for mono, 2 for stereo, and so on).

— ***int getSamplesPerSec()*** – returns the number of samples per second.

— ***int getAvgBytesPerSec()*** – returns the average number of bytes per second.

— ***int getBitsPerSample()*** – returns the number of bits per sample.

— ***int getBlockAlign()*** – returns the block alignment in bytes. Software must process a multiple of block alignment bytes. For PCM data,

```
blockAlign = getNumberOfChannels() x getBitsPerSample() / 8.
```

AudioBuffer allows an MPEGlet application to perform some signal processing operations on the PCM data output from an audio decoder.

NOTE 1 Some terminals may allow compressed audio data to be sent to external audio processors. If **AudioBuffer** is exposed, the terminal must provide decompressed (PCM) data to the application.

NOTE 2 Data contained in **AudioBuffer**'s buffer is intended for visual processing by the application. The possibly modified buffer should not be sent to speakers.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

5.3.6 Terminal capability API

The terminal capability API enables an application to retrieve static properties of a terminal (e.g. CPU type, Operating System and so on) and dynamic properties (e.g. CPU load, network load, and so on). The terminal capability API provides the same features as ISO/IEC 14496-11 terminal capability API in a more concise and extensible manner. Figure 10 provides a static view of the interfaces and methods of this API:

— **ResourceManagerBase** interface provides the methods

  — ***Object getCapability(String capability)*** retrieves the value of a static capability.

  — ***DynamicCapabilityManager getDynamicCapabilityManager()*** provides access to the dynamic capability manager.

— **DynamicCapabilityManager**

  — ***void addDynamicCapabilityObserver(String capability, long threshold, DynamicCapabilityObserver obs)*** registers a DynamicCapabilityObserver *obs* to observe a dynamic *capability*. The DynamicCapabilityObserver will be notified if the value of the capability goes over the *threshold*.

  — ***void removeDynamicCapabilityObserver(String capability, DynamicCapabilityObserver obs)*** de-registers a DynamicCapabilityObserver *obs* from observing a *capability*.

  — ***void deleteObserver(DynamicCapabilityObserver obs)*** removes a DynamicCapabilityObserver *obs* from observing all capabilities it was registered to observe.

— **DynamicCapabilityObserver**

  — ***void update(String capability, long value)*** is called when a dynamic *capability* observed by this DynamicCapabilityObserver has reached a *value* above the threshold it was registered for.

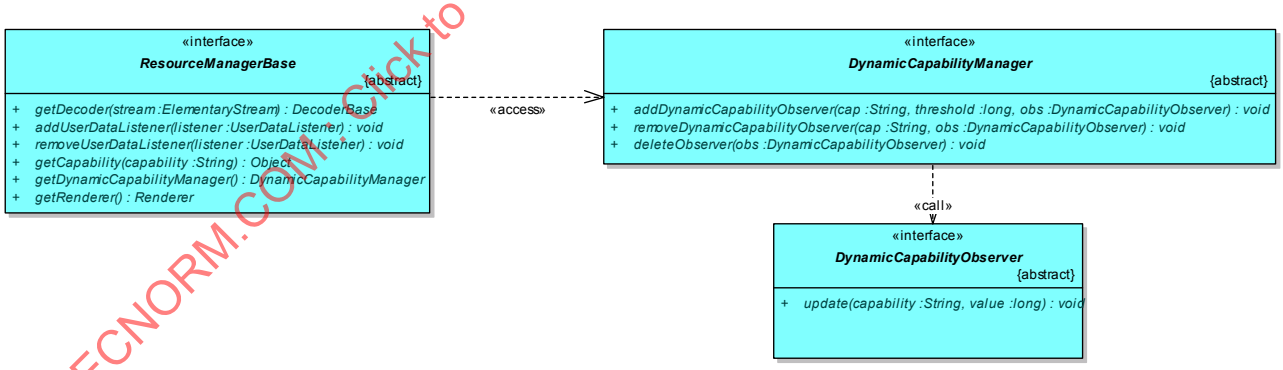


Figure 10 — Capabilities API

The following static capabilities and return values are defined in Table 3. Any other capability name returns a **null** object.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

Table 3 — Static capabilities

Static capability	Return value	Example of value
cpu.speed	Integer	3000
cpu.type	String	Pentium
cpu.architecture	String	x86
cpu.num	Integer	1
mpeg4.profile.audio	String	
mpeg4.profile.graphics	String	
mpeg4.profile.mpegj	String	
mpeg4.profile.od	String	
mpeg4.profile.scene	String	
mpeg4.profile.visual	String	
screen.dimension	int[2]	{800,600}

Dynamic properties evolve over time. An application can register itself as an observer of a dynamic property and it will be notified whenever the property reaches a threshold. Any other capability will not register an observer and hence no notification will happen.

Table 4 — Dynamic capabilities

Dynamic capability	Return value	Example of value
application.memory.free	long	
application.memory.total	long	
network.load	long	
terminal.memory.free	long	
terminal.memory.total	long	
terminal.load	long	

The semantic of the capabilities defined in this specification is as follows:

— ***cpu.speed*** – speed in Hertz of the CPU used in the terminal

— ***cpu.type*** – the type of CPU in the terminal

— ***cpu.architecture*** – the architecture of the CPU in this terminal

— ***cpu.num*** – the number of CPUs in this terminal

— ***screen.dimension*** – an array of 2 integers for the maximal width and height of the screen.

— ***mpeg4.profile.XXXX*** – the maximum profile and level supported by this terminal. The valid names are defined in ISO/IEC 14496-1.

— ***application.memory.free*** – the number of bytes of free memory available to an application

— ***application.memory.total*** – the total number of bytes of memory available to an application

— ***terminal.memory.free*** – the number of bytes of free memory available in the terminal

— ***terminal.memory.total*** – the total number of bytes of memory available in the terminal

— ***network.load*** – the percentage of bandwidth used

— ***terminal.load*** – the percentage of CPU used

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

5.3.7 Systems package

On terminals implementing ISO/IEC 14496 Systems, MPEG-J enables access to MPEG-4 specific objects with packages **org.iso.mpeg.mpegj.systems**, **org.iso.mpeg.mpegj.decoder, org.iso.mpeg.mpegj.scene,** and **org.iso.mpeg.mpegj.net.**

The **org.iso.mpeg.mpegj.systems** package provides access to MPEG-4 Systems features, namely **MPEG4Context** that is more a detailed **TerminalContext** (see subclause 5.3.2 for specification of terminal contexts), which provides access to object descriptors. Figure 11 depicts this API. The descriptor interfaces are unchanged from ISO/IEC 14496-11 and their semantics are unchanged.

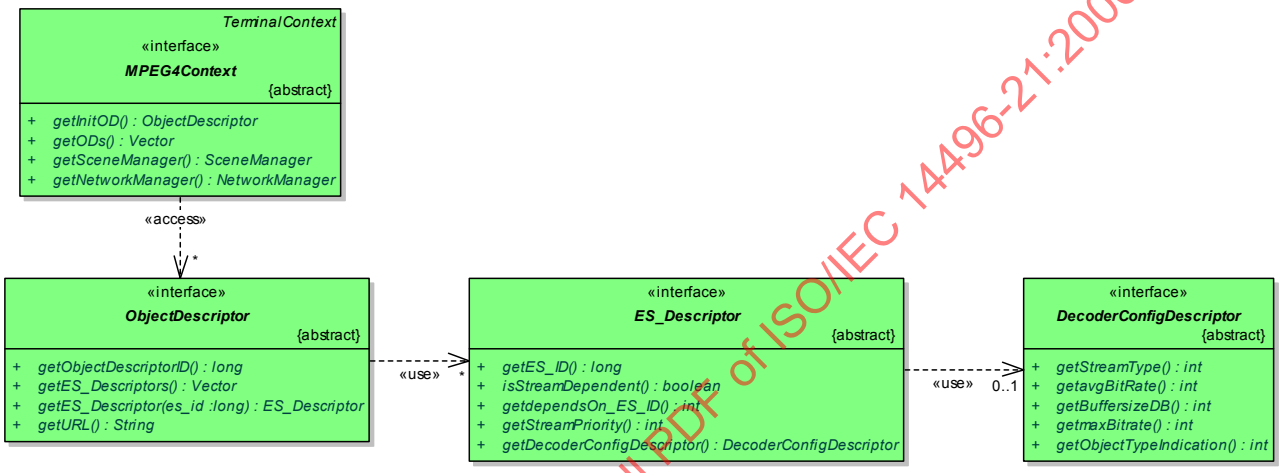


Figure 11 — org.iso.mpeg.mpegj.systems provides access to MPEG-4 Systems features.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

ISO/IEC 14496-11 defines **org.iso.mpeg.mpegj.scene, org.iso.mpeg.mpegj.decoder, org.iso.mpeg.mpegj.net** packages which are reused unchanged in this specification. The **decoder** package has been augmented in this specification with **DecoderControl** interface, and **MPDecoder** interface has been modified to fit the new architecture, as depicted in Figure 12.

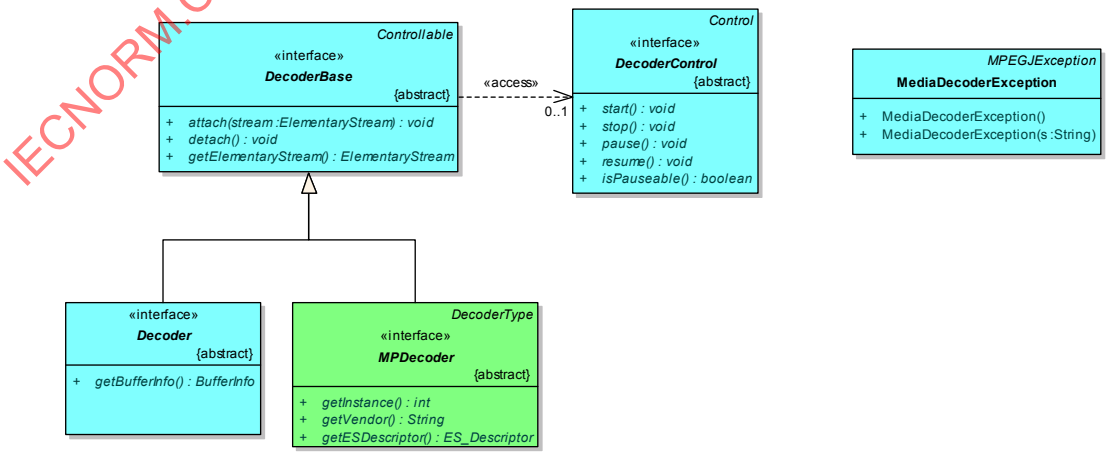


Figure 12 — org.iso.mpeg.mpegj.decoder package overview (decoder specific exceptions not shown)

© ISO/IEC 2006 — All rights reserved

5.3.8 Persistent storage (record store)

ResourceManager provides access to the **RecordStore** object (see subclause 5.3.2), which implements a generic persistent storage mechanism. A generic persistent storage mechanism is necessary for the applications targeted by this specification, for example to save applications' state, game score, and so on.

The Mobile Information Device Profile (MIDP) [6] provides the Record Management System (RMS) in the **javax.microedition.rms** package. A record store consists of a collection of records which will remain persistent across multiple invocations of the MPEGlet. RMS records consist of byte arrays.

An application can store any kind of information including elementary streams of a content. Stored contents can be played back from the store by creating a **Player** from an **InputStream** encapsulating a stored content.

5.4 Dynamic view

This subclause of the document defines the behaviour of applications: the MPEGlet states and their management by the terminal or the application.

5.4.1 MPEGlet states

An MPEGlet has five states:

- ***Loaded:*** The MPEGlet is loaded from local storage or network and its no argument constructor is called. It can enter the Initialized state if the ***MPEGlet.init()*** method is called.
- ***Initialized:*** The MPEGlet is initialized and ready to be active. It can enter the Running state after the ***MPEGlet.run()*** is called.
- ***Running:*** The MPEGlet is running normally. It can enter the destroyed state if ***MPEGlet.destroy()*** method is called. It may also return to the Paused state if ***MPEGlet.pause()*** method is called. It may enter the Initialized state if ***MPEGlet.stop()*** is called.
- ***Paused:*** The MPEGlet is paused. It can enter the Running state after the ***MPEGlet.run()*** is called. It can enter the Initialized state if ***MPEGlet.stop()*** is called. When entering Paused state, applications are expected to release all shared resources and to save the data necessary to resume later in a state identical to that when pause was entered.
- ***Destroyed:*** This is the terminal state. Once it's entered, it cannot return to other states. All its resources are subject to be claimed.

In addition, for example should an error occurs, the terminal may move the application into the Destroyed state from whatever state the application is already in.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

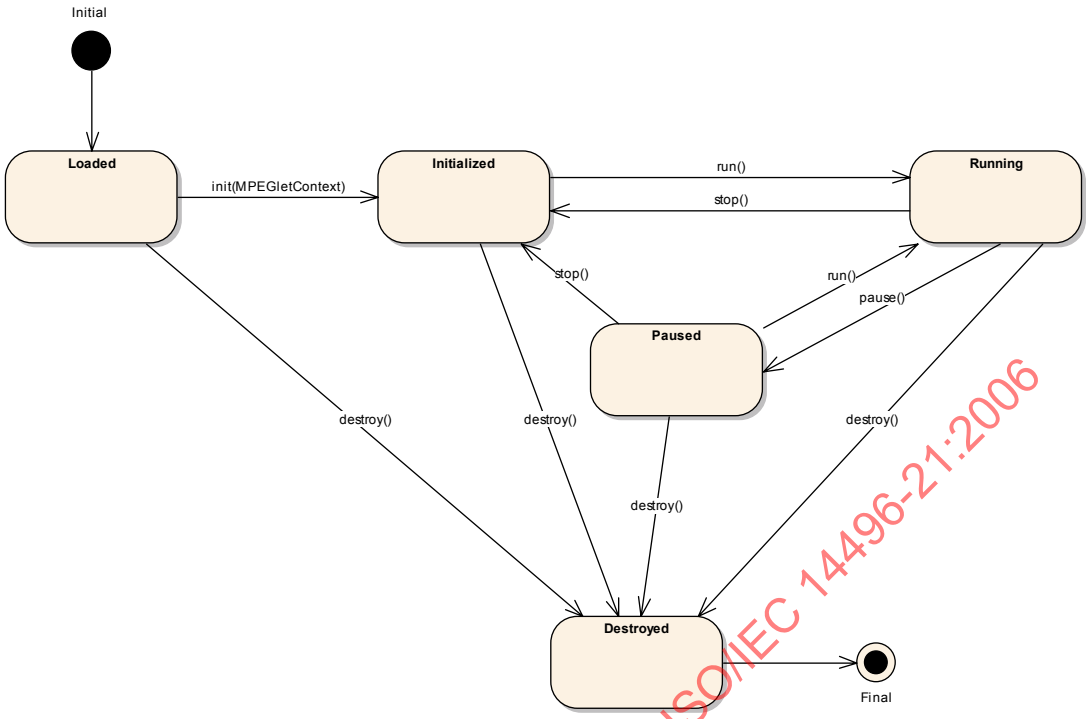


Figure 13 — GFX MPEGlet behaviour.

5.4.2 MPEGlet requests to the terminal

The previous subclause is used by the terminal to communicate to an MPEGlet application that it wants the MPEGlet to change state. If an MPEGlet wants to change its own state, it can use the **MPEGletContext**'s request methods. The sequence of these operations for requesting state change to Paused, or Running states is depicted in Figure 14.

The MPEGlet calls its ***MPEGletContext.requestPause()*** or ***MPEGletContext.requestResume()*** methods, which in turn notify the terminal. In return, the terminal calls ***MPEGlet.pause()*** or ***MPEGlet.run()*** respectively.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

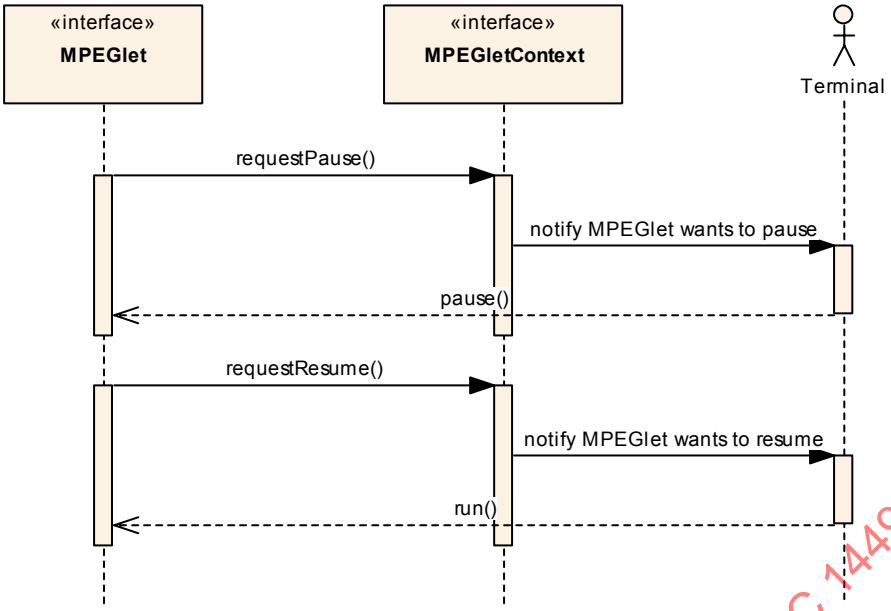


Figure 14 — Behaviour of MPEGletContext state change request methods.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

5.4.3 Player states

The player states and transitions are defined by MMAPI. This is summarized below and in Figure 15.

UNREALIZED State

A **Player** starts in the *UNREALIZED* state. An unrealized **Player** does not have enough information to acquire all the resources it needs to function.

The following methods must not be used when the **Player** is in the *UNREALIZED* state.

- ***`getContentType()`***
- ***`setTimeBase()`***
- ***`getTimeBase()`***
- ***`setMediaTime()`***
- ***`getControls()`***
- ***`getControl()`***

An `IllegalStateException` will be thrown.

The ***realize()*** method transitions the **Player** from the *UNREALIZED* state to the *REALIZED* state.

REALIZED State

A **Player** is in the *REALIZED* state when it has obtained the information required to acquire the media resources. Realizing a **Player** can be a resource and time consuming process. The **Player** may have to communicate with a server, read a file, or interact with a set of objects.

© ISO/IEC 2006 — All rights reserved

Although a realized **Player** does not have to acquire any resources, it is likely to have acquired all of the resources it needs except those that imply exclusive use of a scarce system resource, such as an audio device.

Normally, a **Player** moves from the *UNREALIZED* state to the *REALIZED* state. After ***realize()*** has been invoked on a **Player**, the only way it can return to the *UNREALIZED* state is if ***deallocate()*** is invoked before ***realize()*** is completed. Once a **Player** reaches the *REALIZED* state, it never returns to the *UNREALIZED* state. It remains in one of four states: *REALIZED*, *PREFETCHED*, *STARTED* or *CLOSED*.

PREFETCHED State

Once realized, a **Player** may still need to perform a number of time-consuming tasks before it is ready to be started. For example, it may need to acquire scarce or exclusive resources, fill buffers with media data, or perform other start-up processing. Calling ***prefetch()*** on the **Player** carries out these tasks.

Once a **Player** is in the *PREFETCHED* state, it may be started. Prefetching reduces the startup latency of a **Player** to the minimum possible value.

When a started **Player** stops, it returns to the *PREFETCHED* state.

STARTED State

Once prefetched, a **Player** can enter the *STARTED* state by calling the ***start*** method. A *STARTED* **Player** means the **Player** is running and processing data. A **Player** returns to the *PREFETCHED* state when it stops, because the ***stop()*** method was invoked, it has reached the end of the media, or its stop time.

When the **Player** moves from the *PREFETCHED* to the *STARTED* state, it posts a `STARTED` event. When it moves from the *STARTED* state to the *PREFETCHED* state, it posts a `STOPPED`, `END_OF_MEDIA` or `STOPPED_AT_TIME` event depending on the reason it stopped.

The following methods must not be used when the `Player` is in the *STARTED* state:

- ***`setTimeBase()`***
- ***`setLoopCount()`***

An `IllegalStateException` will be thrown.

CLOSED state

Calling `close` on the **Player** puts it in the *CLOSED* state. In the *CLOSED* state, the **Player** has released most of its resources and must not be used again.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

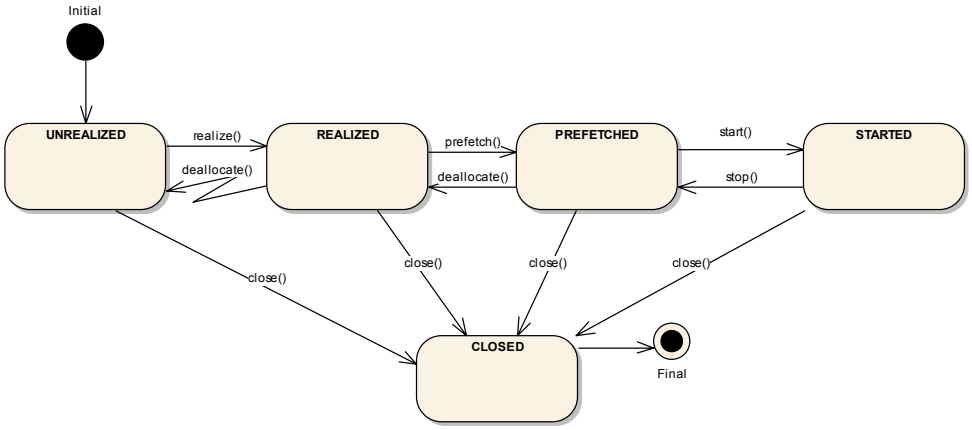


Figure 15 — Player states

5.5 Considerations

The design of this specification favoured non exposure of native handles to region in memory, terminal window, graphic context, and so on, for security reasons.

This specification does not restrict image dimensions and orientation as some rendering APIs do.

Per the OpenGL ES and M3G specifications, only one MPEGlet at a time can access the rendering context. Therefore, special care must be taken in a multithreaded environment such as multiple MPEGlets running in parallel, or an MPEGlet spawning multiple threads, or an MPEGlet listening to AWT events (e.g. mouse events, keyboard events and so on). In those cases, one must use proper synchronization mechanisms to protect shared information among threads including the Renderer.

Even if MMAPI's **Manager** class is available in a MMAPI compliant terminal, applications should not call **Manager** directly but instead use **ResourceManager** that provides MPEGlet-dependent resource management.

5.6 Application-specific data in MPEG-J stream

As Graphics applications can define their own scene representation and logic, a dedicated format may be used to carry application specific data. This is supported by enabling the MPEG-J elementary stream to carry private data similar to audio and video elementary streams. By making it part of the MPEG-J elementary stream the timing model in MPEG-4 Systems can be used for synchronization with other elementary streams.

As this specification allows usage of any rendering API, downloaded applications may not be able to run on a terminal that doesn't support the rendering API the application needs. A Java Application Descriptor can be sent to the terminal before it loads the application. If application's necessary resources are not available, the terminal will not load the application.

5.6.1 Java stream header extensions

Subclause 11.4.3.2 of ISO/IEC 14496-11 defines the Java stream semantic and the Java stream header. In this document, we update them with the fields in **bold** in Listing 1. The changes are backward compatible with ISO/IEC 14496-11.

- If **hasUserData** = 1, the **UserData** class contains an array of bytes and is passed to the application in the Buffer interface

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

- If **hasApplicationDescriptor** = 1, the application descriptor is provided as a null-terminated String. The application descriptor is similar to a Jar's manifest file. It contains specific key-value pairs that indicates the framework specific resources (in particular graphics) the application needs. If the terminal doesn't support such resources, it can skip loading further data from the stream.

- for this specification, **version** = 0x2 in hexadecimal

Listing 1 — Java stream header

```
aligned(32) class JavaStreamHeader {
    bit(2) version;
    bit(1) isClassFlag;
    bit(13) numReqClasses;
    bit(1) isPackaged ;
    bit(3) compressionScheme;
    bit(1) hasUserData;
    bit(1) hasApplicationDescriptor;
    bit(10) reserved;
    JavaClassID classID;

    JavaClassID reqClassID[numReqClasses];

    if(hasUserData)
        UserData    userData;
    if(hasApplicationDescriptor)
        String      appDescriptor;
}

aligned(32) class UserData
{
}
```

5.6.2 MPEGlet access to JavaStreamHeader user data

An **MPEGlet** may register to the **ResourceManager** as a **UserDataListener**. The **MPEGlet** will be notified about access units of **UserData** whenever received and the system notifies the **MPEGlet** with a byte array that contains user data; the **MPEGlet** is responsible for decoding such information.

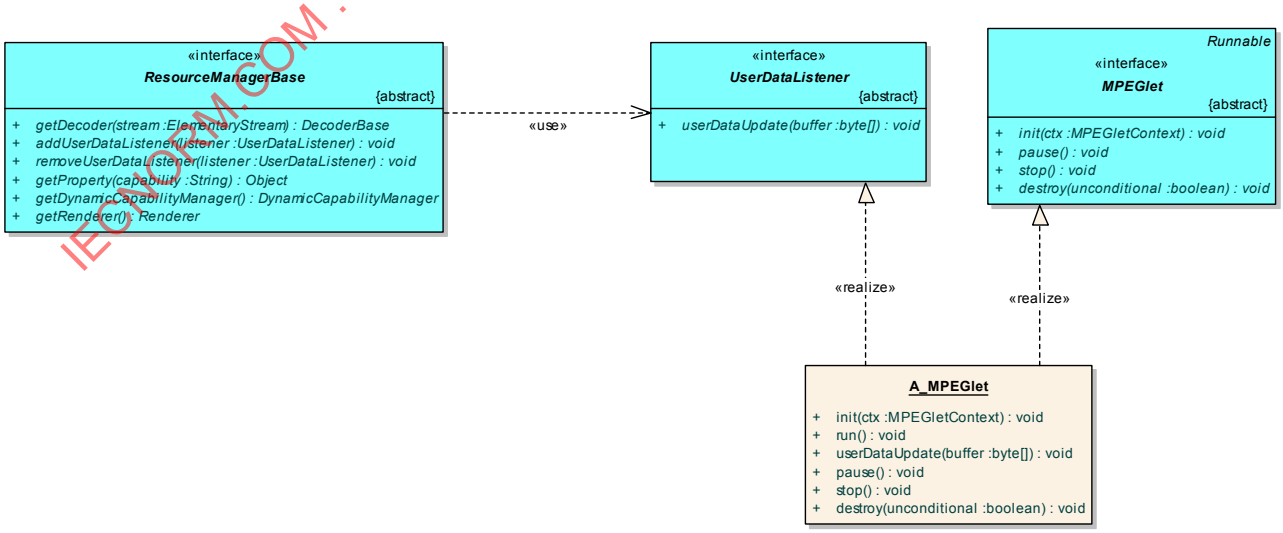


Figure 16 — Example: usage of UserDataListener interface to access JavaStreamHeader user data.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

The methods to access Java stream header user data is as follows:

— In **ResourceManager**

  — ***addUserDataListener(UserDataListener listener)*** – registers a **UserDataListener** to receive user data from Java stream.

  — ***removeUserDataListener(UserDataListener listener)*** – removes a **UserDataListener** from receiving user data from Java stream

— In **UserDataListener**

  — ***userDataUpdate(byte[] buffer)*** provides **MPEGlet** with the payload of the JavaStreamHeader user data.

NOTE MPEGlets should not take too long in ***userDataUpdate()*** as they may risk blocking systems processing.

5.7 Application descriptor

An application can carry descriptive information about itself in the manifest file that is contained in its JAR file under the name of META-INF/MANIFEST.MF. This text file can also be streamed in the JavaStreamHeader class in the previous subclause. The application descriptor follows the JAR Manifest specification [8] and contains various attributes in the form <attribute-name> : <attribute-value>.

Such attributes can be read by MPEGlets using the method ***MPEGletContext.getProperty(key)***. Listing 2 shows an example of application descriptor.

Listing 2 - An Example Of a Java Application Descriptor (JAD) File

```
MPEGlet-Class: com.mycompany.MyMPEGlet
MPEGlet-Name: My super duper application
MPEGlet-Version: 1.2.3
MPEGlet-Vendor: MyCompany Inc.
MPEGlet-URL: http://www.mycompany.com/mpeglets/mympeglet.jar
MPEGlet-Jar-Size: 1234
Renderer-Class: com.mycompany.renderer
Renderer-Version: 1.0
Renderer-URI: http://www.mycompany.com/mpeglets/myrenderer.jar
```

Required attributes:

- **MPEGlet-Class**. Specifies the fully qualified name of the application.
- **MPEGlet-Version**. The MPEGlet version.
- **MPEGlet-Vendor**. The MPEGlet vendor.
- **MPEGlet-URL**. The URL from where the application can be downloaded.
- **Renderer-Class**. Specific the name of the class that implements Renderer interface.

Optional attributes:

- **MPEGlet-Name**. The friendly name of the application that is displayed to the user.
- **MPEGlet-Jar-Size**. The size of the JAR file.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

- **MPEGlet-Description.** A brief description of the application for the user
- **MPEGlet-Icon**. An icon that will be associated with the application (if terminal supports it). PNG file format is used.
- **Renderer-Version.** Version of the renderer. This is a number in the form **major [. minor [. build]]**. For example 1.2.0. If minor or build is not specified, this means, respectively, the latest minor version of the specified major version of the renderer, or the latest build version of the specified major.minor version. For example, if Renderer-Version = 1 and the player has Renderer 1.2 and 1.5, it shall use version 1.5.
- **Renderer-URI.** Universal Resource Identifier (URI) for the renderer.

Application specific attributes:

- Any attributes others than those listed in this subclause can be used to configure the application.

5.8 Terminal properties

As for application-specific properties, terminal properties can be retrieved by calling **String MPEGletContext.getProperty(String key)** where key is one of the followings:

- **Terminal-Renderers**. Returns a space-separated list of renderers supported by the terminal. Calling MPEGletContext.getProperty() with a renderer name as the parameter return the version number of that renderer.

5.9 Examples (informative)

5.9.1 Using Java bindings to OpenGL ES

The following example shows how to use the GFX MPEGlet API with JSR-239 Java bindings to OpenGL ES. This example displays a multi-colored cube.

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

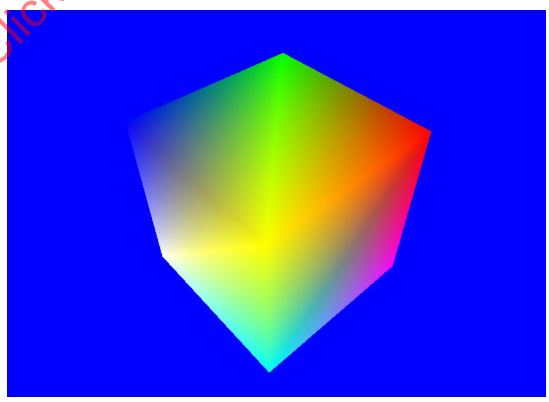


Figure 17 — Output of MPEGletTest.

```
public abstract class MPEGletTest implements MPEGlet
{
  MPEGletContext      _ctx;
  ResourceManager     _resMgr;
  boolean             shouldPause, shouldStop;
  FloatBuffer         vb,cb;
  ByteBuffer          ib;
```

© ISO/IEC 2006 — All rights reserved

```
Boolean             resized;
Canvas              cv;
Int                 width, height;
GL                  gl;
EGL                 egl;

float[] cubeVertices =
{
  -1.0f,-1.0f, 1.0f , // 0
  1.0f,-1.0f, 1.0f , // 1
  1.0f, 1.0f, 1.0f , // 2
  -1.0f, 1.0f, 1.0f , // 3
  -1.0f,-1.0f,-1.0f , // 4
  -1.0f, 1.0f,-1.0f , // 5
  1.0f, 1.0f,-1.0f , // 6
  1.0f,-1.0f,-1.0f, // 7
};

float[] cubeColors =
{
   1.0f,0.0f,0.0f, 1,  // 0
   0.0f,1.0f,0.0f, 1,  // 1
   0.0f,0.0f,1.0f, 1,  // 2
   1.0f,1.0f,0.0f, 1,  // 3
   1.0f,0.0f,1.0f, 1,  // 4
   0.0f,1.0f,1.0f, 1,  // 5
   1.0f,1.0f,1.0f, 1,  // 6
   1.0f,0.0f,0.0f, 1,  // 7
};

byte cubeIndices[] =
{
  0, 1, 2, 3, // Quad 0
  4, 5, 6, 7, // Quad 1
  5, 3, 2, 6, // Quad 2
  4, 7, 1, 0, // Quad 3
  7, 6, 2, 1, // Quad 4
  4, 0, 3, 5  // Quad 5
};

// swap last 2 indices per quad for a triangle strip
byte cubeIndicesTriStrip[] =
{
    0, 1, 3, 2, // Quad 0
    4, 5, 7, 6, // Quad 1
    5, 3, 6, 2, // Quad 2
    4, 7, 0, 1, // Quad 3
    7, 6, 1, 2, // Quad 4
    4, 0, 5, 3  // Quad 5
};

public void init(MPEGletContext ctx) throws MPEGletStateChangeException
{
  System.out.println("[" + getName() + "] initialized");
  _ctx = ctx;
  _resMgr = (ResourceManager) ctx.getTerminalContext().getResourceManager();
}

public synchronized void pause()
{
  System.out.println("[" + getName() + "] paused");
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

```
  shouldPause = !shouldPause;
  notify();
}

public synchronized void stop()
{
  System.out.println("[" + getName() + "] stopped");
  shouldStop = true;
  notify();
}

public void destroy(boolean unconditional) throws MPEGletStateChangeException
{
  // TODO Auto-generated method stub
}

public void run()
{
  doInit();

  shouldStop = false;
  shouldPause = false;

  while (!shouldStop)
  {
    synchronized (this)
    {
      while (shouldPause)
      {
        try
        {
          wait();
        }
        catch (Exception ex)
        {
        }
      }
    }

    // do any processing
    doRun();

    synchronized (this)
    {
      try
      {
        wait(1);
      }
      catch (Exception ex)
      {
      }
    }
  }

  // app is over, let's clean resources
  doStop();
}

 void doInit()
 {
  cv=new Canvas();
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved

```
    cv.addComponentListener(new ComponentAdapter() {
      public void componentResized(ComponentEvent e)
      {
        resized=true;
        width=cv.getWidth();
        height=cv.getHeight();
      }
    });

    ((Display)_ctx.getDisplay()).setCurrent(cv); // for MIDP
    //((Frame)_ctx.getDisplay()).add(cv); // for non-MIDP

    // init EGL
    GLRenderer renderer=(GLRenderer)
_resMgr.createRenderer(ResourceManager.GL_RENDERER);
    egl = renderer.getEGL();

    /* get an EGL display connection */
    display = egl.eglGetDisplay(EGL.EGL_DEFAULT_DISPLAY);

    /* initialize the EGL display connection */
    int[] major= {0},minor= {0};
    egl.eglInitialize(display, major, minor);

    /* get an appropriate EGL frame buffer configuration */
    int attribute_list[] = {
        EGL.EGL_RED_SIZE, 8, // at least 1-bit red
        EGL.EGL_GREEN_SIZE, 8, //
        EGL.EGL_BLUE_SIZE, 8, //
        EGL.EGL_ALPHA_SIZE,     EGL.EGL_DONT_CARE,
        EGL.EGL_DEPTH_SIZE,     32, // 16,
        EGL.EGL_STENCIL_SIZE,   EGL.EGL_DONT_CARE,
        EGL.EGL_SURFACE_TYPE,   EGL.EGL_WINDOW_BIT,
        EGL.EGL_NONE,
    };
    EGLConfig configs[] = new EGLConfig[1];
    int numConfigs[] = { 0 };
    egl.eglChooseConfig(display, attribute_list, configs, configs.length,
numConfigs);

    /* create an EGL window surface */
    drawSurface = egl.eglCreateWindowSurface(display, configs[0], cv, null);

    /* create an EGL rendering context */
    context = egl.eglCreateContext(display, configs[0], EGL.EGL_NO_CONTEXT,
null);

    /* connect the context to the surface */
    egl.eglMakeCurrent(display, drawSurface, drawSurface, context);

    gl = renderer.getGL();

    // init our scene
    vb=FloatBuffer.allocate(cubeVertices.length);
    vb.put(cubeVertices);
    cb=FloatBuffer.allocate(cubeColors.length);
    cb.put(cubeColors);
    ib=ByteBuffer.allocate(cubeIndicesTriStrip.length);
    ib.put(cubeIndicesTriStrip);

    gl.glEnableClientState(GL.GL_VERTEX_ARRAY);
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 14496-21:2006

© ISO/IEC 2006 — All rights reserved