
**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*



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

Documents
Document Sharing Hub
© 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.

© 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

Contents		Page
Foreword.....		iv
1	Scope	1
2	Normative references	1
3	Symbols and abbreviated terms	1
4	Notations	1
5	MPEG-J Graphics Framework eXtension	2
5.1	Introduction	2
5.2	Architecture	3
5.3	Static view	5
5.4	Dynamic view	20
5.5	Considerations	24
5.6	Application-specific data in MPEG-J stream	24
5.7	Application descriptor	26
5.8	Terminal properties	27
5.9	Examples (informative)	27
Annex A (normative) GFX API listing		35
Annex B (normative) Buffers, formats and data types		37
Bibliography		38

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*

- *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 (LSeR) and Simple Aggregation Format (SAF)*
- *Part 21: MPEG-J Graphics Framework eXtension (GFX)*
- *Part 22: Open Font Format*

Information technology — Coding of audio-visual objects —

Part 21:

MPEG-J Graphics Framework eXtensions (GFX)

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.

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.

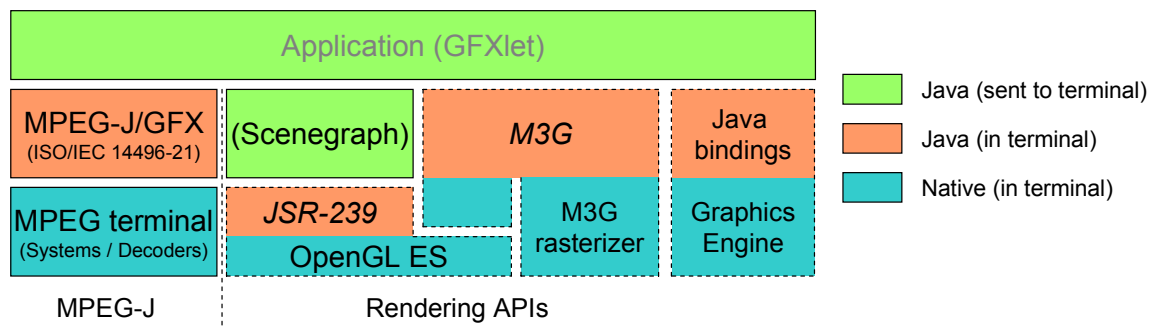


Figure 1 — Block diagram of an MPEG-4 Player with MPEG-J extensions for rendering.

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.

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.

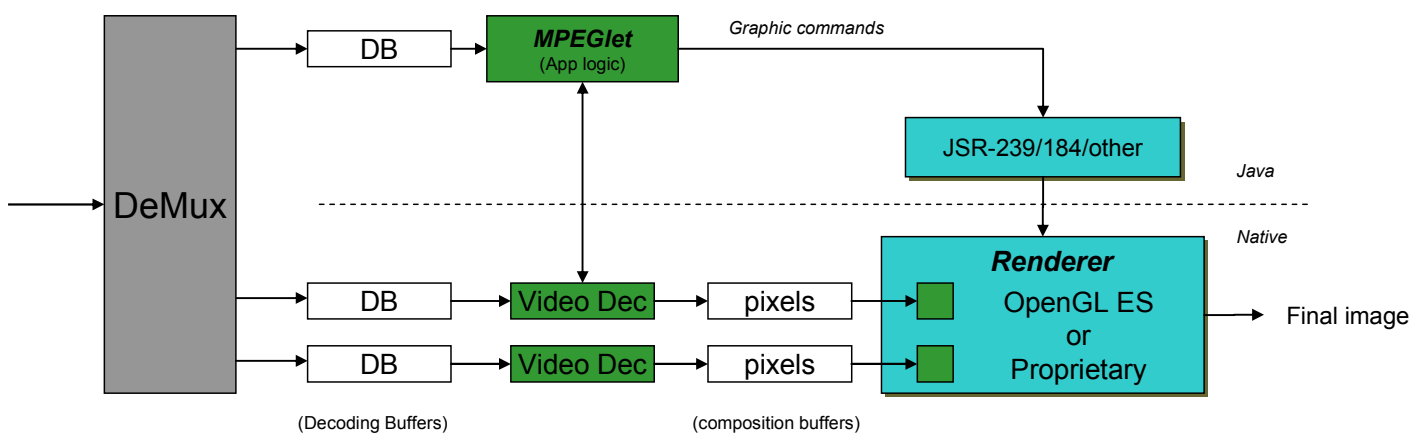


Figure 2 — Conceptual workflow.

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

In this specification, such an abstraction is represented by the concepts of **DataSource**, **Player**, and **Controls** 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 **DataSources**, **Players**, **Renderers**, and **MPEGlet**. **DataSources**, **Players**, and streams may expose controls for the MPEGlet.

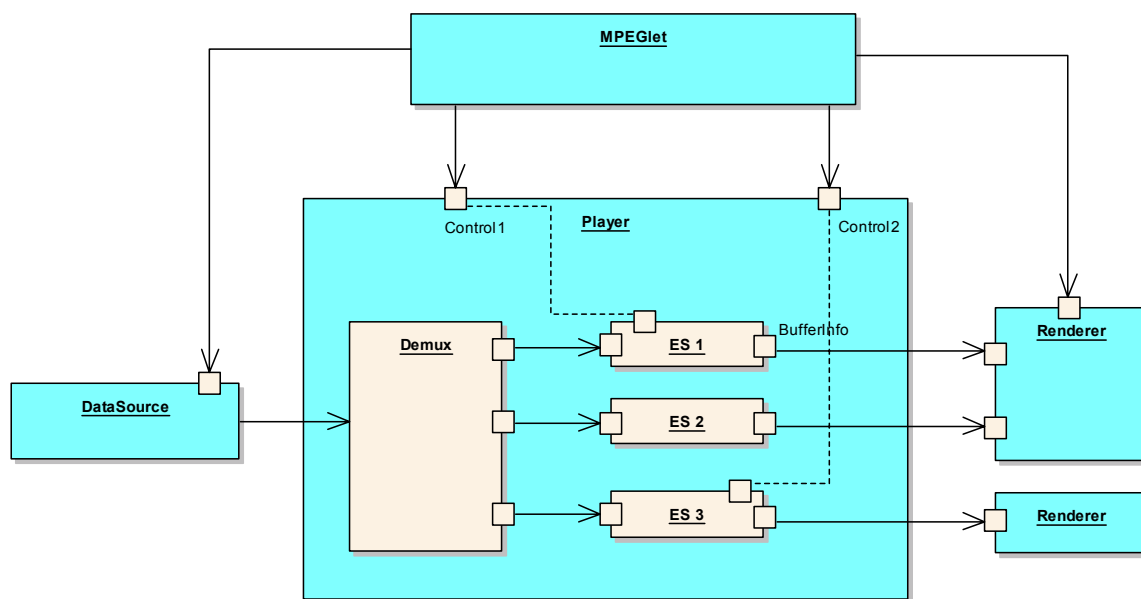


Figure 3 — Conceptual view of the terminal from an application.

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

- For a video decoder, **GLBuffer** wraps a byte array of pixels in an optimized format for the graphic card.
- For other decoders, specific **BufferInfos** 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

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).

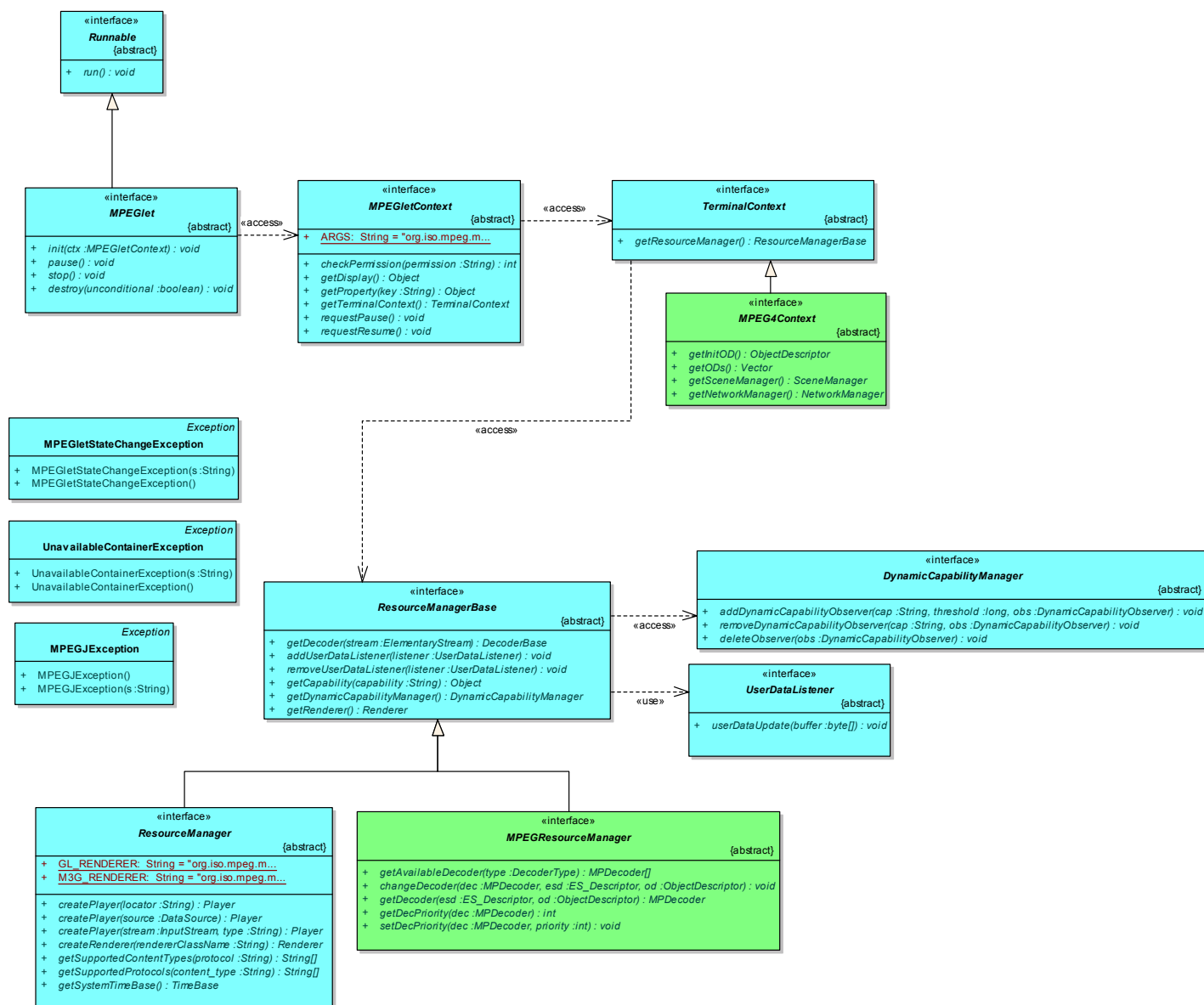


Figure 4 — GFX MPEGlets architecture.

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

- **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** 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.microedition.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`.

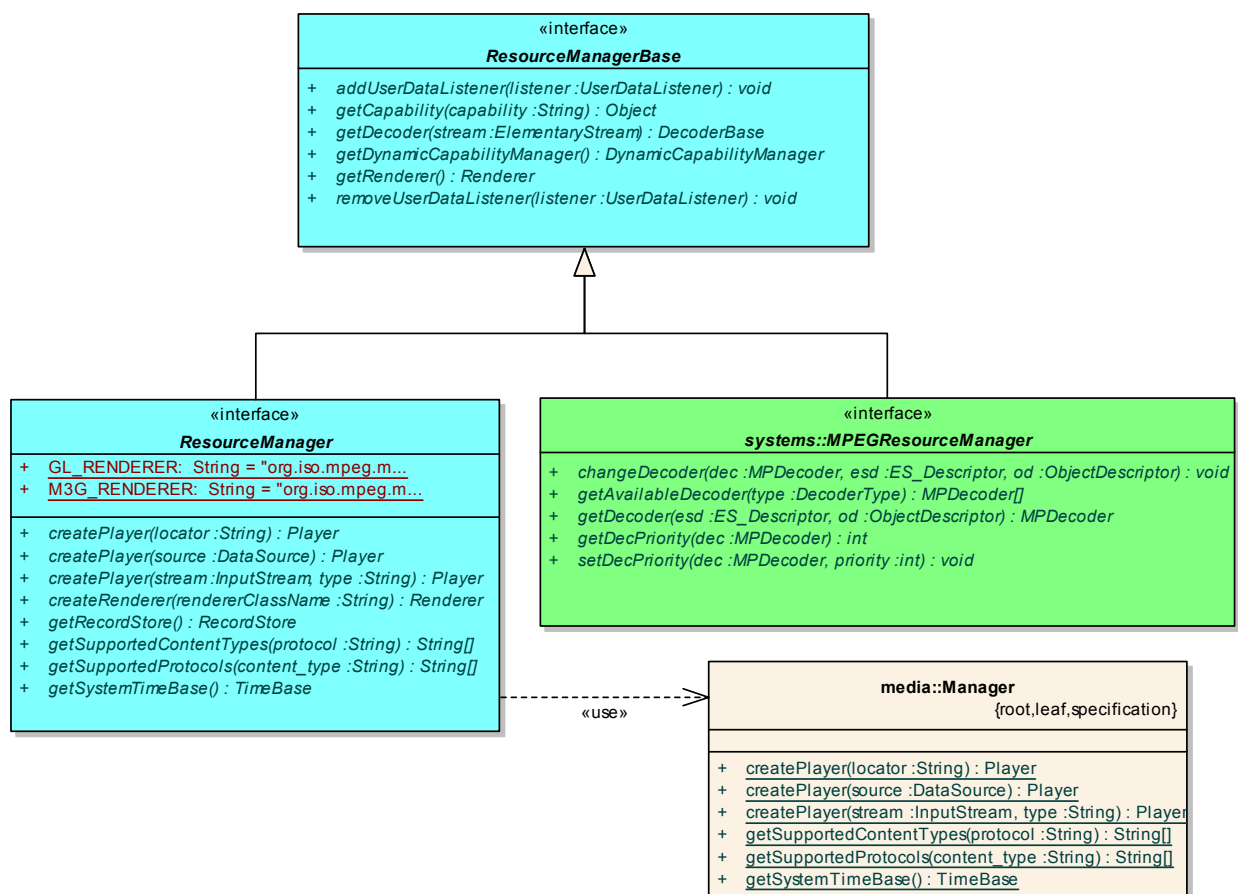


Figure 5 — ResourceManager and its factory methods.

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"