ETSI TS 101 600 V1.1.1 (2012-05)

Digital Video Broadcasting (DVB);
GEM Profile for Plano-Stereoscopic 3DTV

EBU·UER

DVB Digital Video Broadcasting

Reference
DTS/JTC-DVB-317

Keywords
3DTV, DVB, GEM, MHP, stereoscopic

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights 5
Foreword 5
Introduction 6
1 Scope 7
2 References 7
2.1 Normative references 7
2.2 Informative references 8
3 Definitions and abbreviations 8
3.1 Definitions 8
3.2 Abbreviations 9
4 GEM 3D profile 9
4.1 Application scenarios (informative) 10
4.2 Graphics reference model 11
4.2.1 Graphics modes 13
4.2.1.1 Conventional 2D mode 13
4.2.1.2 2D mode with volumetric 3D graphics 13
4.2.1.3 Limited Stereoscopic 3D mode with a single graphics plane 13
4.2.1.4 Stereoscopic 3D mode with pseudo-3D graphics (2 graphics planes) 14
4.2.1.5 Stereoscopic 3D mode with volumetric 3D graphics 14
4.3 Display engine 15
4.3.1 Disparity handling 17
4.3.1.1 Video Disparity 18
4.3.1.2 Subtitle Disparity 18
4.3.2 Disparity events 18
4.4 Stereoscopic OpenGL ES 18
4.4.1 Stereoscopic OpenGL API extensions 19
4.4.1.1 Definitions for specific terms used in this clause 19
4.4.1.2 Implementation guideline for stereoscopic OpenGL rendering pipeline 19
4.4.1.2.1 Stereo 3D scene rendering pipeline. 20
4.4.1.3 Clarifications and restrictions on JSR 239 20
5 GEM 3D API Packages 21
5.1 Display and Device capabilities 22
5.2 Graphics modes 26
5.3 Video Controls 29
5.4 Drawing API 35
6 Service Information 42
6.1 JavaTV Service Information APIs 43
6.1.1 javax.tv.service.ServiceType 43
6.1.2 javax.tv.service.navigation.StreamType 44
6.2 DVB Service Information APIs 45
6.2.1 Service Type 45
6.2.2 Component type 45
6.2.3 Content Descriptor 45
6.3 Signaling of 3D Applications 46
6.3.1 Graphics Constraints Descriptor 46
Annex A (informative): Sample Code 47
A.1 AWT stereoscopic drawing / Disparity change handling 47
A.2 Stereoscopic OpenGL 50
A.3 Video Controls and Listeners 54

Annex B (informative): Bibliography 56
History 57

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Specification (TS) has been produced by Joint Technical Committee (JTC) Broadcast of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI).

NOTE: The EBU/ETSI JTC Broadcast was established in 1990 to co-ordinate the drafting of standards in the specific field of broadcasting and related fields. Since 1995 the JTC Broadcast became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has active members in about 60 countries in the European broadcasting area; its headquarters is in Geneva.

European Broadcasting Union
CH-1218 GRAND SACONNEX (Geneva)
Switzerland
Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

The Digital Video Broadcasting Project (DVB) is an industry-led consortium of broadcasters, manufacturers, network operators, software developers, regulatory bodies, content owners and others committed to designing global standards for the delivery of digital television and data services. DVB fosters market driven solutions that meet the needs and economic circumstances of broadcast industry stakeholders and consumers. DVB standards cover all aspects of digital television from transmission through interfacing, conditional access and interactivity for digital video, audio and data. The consortium came together in 1993 to provide global standardisation, interoperability and future proof specifications.

Introduction

A plano-stereoscopic system creates a 3D illusion for the viewer by delivering two images (left and right) that are seen simultaneously by the left and right eyes. This can be achieved by various techniques on a TV-set, such as shutter glasses, polarizer glasses and others.

Since 2010 many 3D-enabled consumer products have appeared on the market starting with Blu-ray players and TV sets. DVB took into account the need to support planoscopic 3DTV for broadcast services and developed a specification to define the delivery system for plano-stereoscopic 3DTV services, that are compatible with 3DTV capable displays that already available in the market [3].

GEM [1] based TV middleware specifications are adopted in different markets. The present document specifies an extension to the DVB GEM middleware with a new horizontal 3D profile. This 3D GEM profile ensures that interactive applications and graphics are aware of 3D content and can handle drawing situations, where the application graphics are superimposed to 3D movie content.

1 Scope

The present document specifies the extensions to the DVB GEM [1] middleware to enable applications that are aware of frame compatible plano-stereoscopic 3DTV services. This is achieved with a new horizontal 3D profile which can be applied to all GEM targets. The 3D profile extends the display model and presentation APIs, to enable additional 3D display modes and drawing functions, with varying capabilities. For the more capable 3D modes a new drawing API is introduced, which is a stereoscopic extension of the java.awt.Graphics2D API set.

On devices with hardware support for accelerated 3D graphics based on OpenGL ES, the present document specifices an additional 3D mode, which enables the rendering of a 3D volumetric scene to a stereoscopic display. On these devices Open GL ES based drawing is also available for 2D display modes.

NOTE: Although the present document defines the 3D profile for the 1.3 version of the GEM specification, it is also applicable for previous GEM versions.

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

[1] ETSI TS 102 728: "Digital Video Broadcasting (DVB); Globally Executable MHP (GEM) Specification 1.3 (including OTT and hybrid broadcast/broadband)".

NOTE: Available at http://www.etsi.org/deliver/etsi_ts/102700_102799/102728/01.02.01_60/ts_102728v010201p.pdf

[2] ETSI TS 102 727: "Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.2.2".

NOTE: Available at http://www.etsi.org/deliver/etsi_ts/102700_102799/102727/01.01.01_60/ts_102727v010101p.pdf

[3] ETSI TS 101 547: "Digital Video Broadcasting (DVB); Frame Compatible Plano-stereoscopic 3DTV".

NOTE: Available at: http://www.dvb.org/technology/standards/a154_DVB-3DTV_Spec.pdf

[4] ETSI EN 300 468: "Digital Video Broadcasting (DVB); Specification for Service Information (SI) in DVB systems".

NOTE: Available at http://www.dvb.org/technology/standards/a038_DVB-SI_dEN300468v1.12.1.pdf.

[5] ETSI EN 300 743 (V1.4.1): "Digital Video Broadcasting (DVB); Subtitling systems".

NOTE: Available at http://www.etsi.org/deliver/etsi_en/300700_300799/300743/01.04.01_60/en_300743v010401p.pdf

[6] Java Specification Request JSR 239: JavaTM Binding for the OpenGL® ES API.

NOTE: Available at: http://jcp.org/en/jsr/summary?id=239

2.2 Informative references

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] OpenCable™ Specifications: Content Encoding Profiles 3.0 Specification (OC-SP-CEP3.0-I02-110131).

NOTE: Available at: http://www.cablelabs.com/specifications/OC-SP-CEP3.0-I02-110131.pdf

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in TS 101 547 [3] and the following apply:

3DTV: DVB frame compatible plano-stereoscopic three-dimensional television

comfort zone: amount of disparity that is acceptable to most people

disparity: difference between the horizontal positions of a pixel representing the same point in space in the right and left views

NOTE: Positive disparity (horizontal right coordinate greater than horizontal left coordinate) implies a position behind the plane of display, and negative disparity implies a position in front of the display.

Frame Compatible (FC): arrangement of the Left and Right images in a spatial multiplex which results in an image which can be treated like a normal HDTV image by the receiver demodulator and compression decoder

Open GL ES (Open Graphics Library for Embedded Systems): standard set of graphics primitives for rendering 2D and 3D computer graphics on embedded devices

pixel arrangements: arrangement of horizontal and vertical image samples

NOTE: This has an impact on vertical, horizontal, or diagonal resolution.

plano-stereoscopic: three-dimensional picture that uses two single pictures, Left and Right, displayed on a single plane surface (the TV screen in the case of 3DTV)

production disparity hint: method describing how disparity information for 3DTV content are transmitted in the video range descriptor

NOTE: In this method, minimum and maximum disparity are transmitted and are measured as a number of pixels on a reference screen with a horizontal resolution of 11 520 pixels [4].

Side-by-Side (SbS): arrangement of the Frame Compatible spatial multiplex such that the horizontally anamorphic Left eye picture is placed in a spatial multiplex to occupy the first half of each line, and the Right eye picture is placed in the spatial multiplex to occupy the second half of each line

service guide: usually information on programme choice displayed on the screen, and often derived from now and next information broadcast in the multiplex

Top-and-Bottom (TaB): arrangement of the Frame Compatible spatial multiplex such that the vertically anamorphic Left eye picture is placed in the spatial multiplex to occupy the first (top) half of a single HD video frame, and the Right eye picture is placed in the spatial multiplex to occupy the second (bottom) half of a single HD video frame

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AIT	Application Information Table
API	Application Program Interface
AWT	Abstract Windowing Toolkit
EGL	Embedded-System Graphics Library
EPG	Electronic Programme Guide
ES	Embedded Systems
FM	Frequency Modulation
GEM S3D	GEM Stereoscopic 3D Profile, the profile defined in the present document
GEM	Globally Executable MHP
GL	Graphics Library
HD	High-Definition Television
HW	Hardware
IPI	Internet Protocol Infrastructure
JMF	Java Media Framework
JSR	Java Specification Request
MHP	Multimedia Home Platform
MUG	MHP Umbrella Group
NVOD	Near Video on Demand
OCAP	Open Cable Application Platform
SbS	Side-by-Side
SD	Standard-Definition Television
SI	Service Information
TaB	Top-and-Bottom
VM	(Java) Virtual Machine

4 GEM 3D profile

The APIs for stereoscopic 3D supports different video encoding models for plano-stereoscopic 3D, such as frame sequential and frame compatible 3D modes. They provide a common API abstraction, which is agnostic of the implementation details of the actual 3D video encoding model. It supports all plano-stereoscopic video modes as defined in TS 101 547 [3].

GEM TS 102 728 [1] and the present document does not put any restriction on the 3D video formats and 3D encoding schemes.

NOTE: The GEM APIs also support the stereoscopic 3D video modes as defined by the OpenCable Stereoscopic 3D Formatting Specification in clause 10 of Content Encoding Profiles 3.0 [i.1].

The GEM Stereoscopic 3D Profile (GEM S3D) is a horizontal extension to the GEM targets, and can be applied to enhance the display model with 3D stereoscopic graphics.

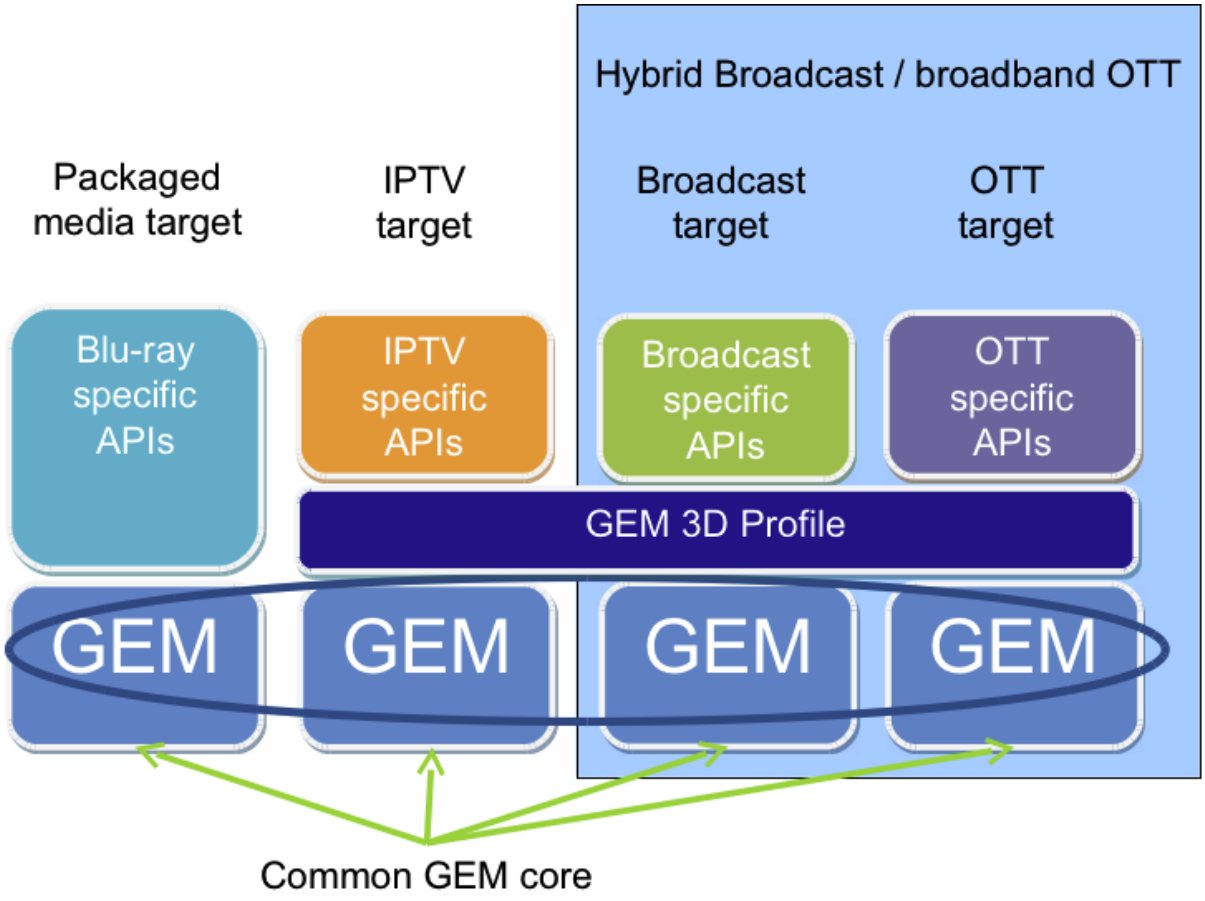

Figure 1: GEM S3D Profile

4.1 Application scenarios (informative)

The following application scenarios are intended to give an informative overview about some use cases for the 3D profile. They are not comprehensive and are included to facilitate a better understanding of the normative API description in the following clauses.

1) An application positions a 2D (flat) graphics layer in front of the 3D movie scene. The graphics layer is always positioned in front of the 3D video scene so it does not interfere with the video and subtitles.

2) An application can move the 2D graphics layer closer to the user or further back within the bounds of the comfort zone. The display system ensures that the graphics layer does not interfere with the 3D movie.

3) An application positions a 3D graphics layer in front of the 3D movie scene. The graphics layer is always positioned in front of the 3D video scene + subtitles so it does not interfere with the video.

4) An EPG application is aware of content that is transmitted in 3D and displays a special logo for 3D movie content.

5) An application positions application-generated subtitles at variable depth positioned over the persons in a scene.

6) An application queries the maximum scene depth and creates a 3D graphics scene, which takes up the remaining depth within the comfort zone.

7) An application switches the 3D video scene temporarily into 2D mode to display a 3D graphics layer. After the end of the users interaction with the graphics layer, the application switches back to 3D.

8) An application shifts the 3D video scene temporarily backwards to a position, which leaves enough room for the graphics layer. After the end of the users interaction with the graphics layer, the application switches back the video position to normal.

9) The application knows that content is 3D although it was signalled as 2D content. The application switches the player engine to 3D. The application can force the video rendering to 3D, 2D or follow the signalling.

4.2 Graphics reference model

The graphics reference model builds on the GEM graphics reference model and is enhanced to support plano-stereoscopic 3D movies and related graphics and display modes.

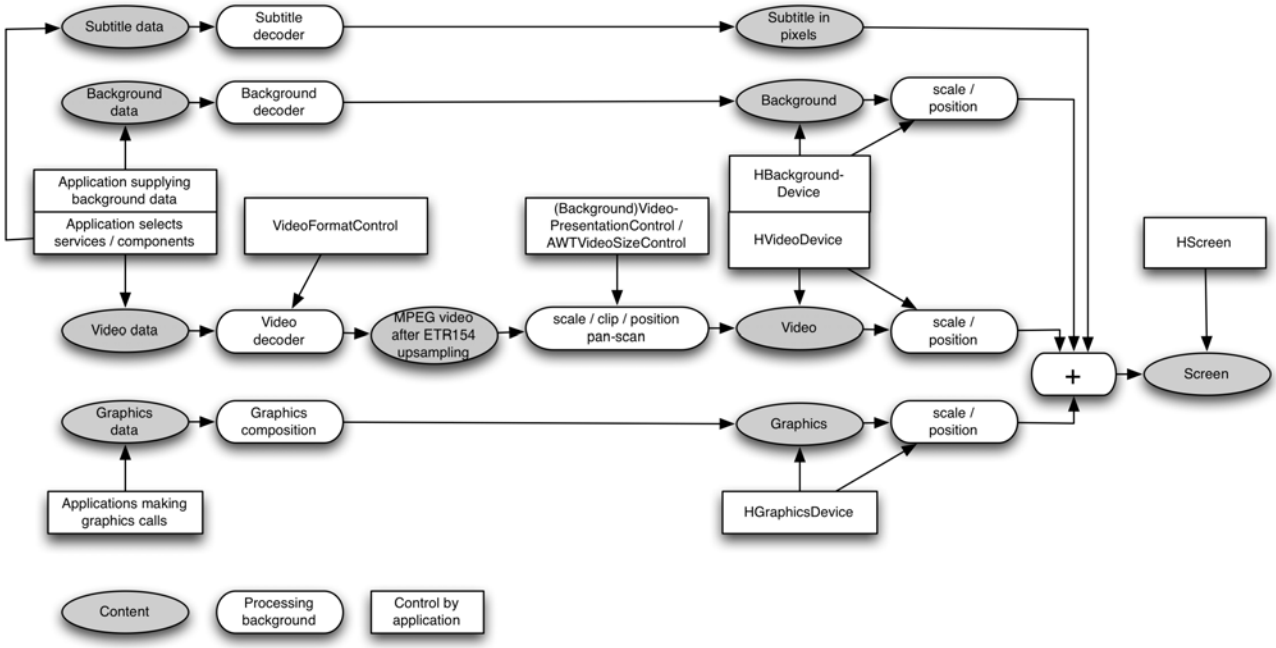

Figure 2: GEM graphics model

The GEM graphics model is extended to support the following 3D modes:

- 2D mode with volumetric (typically HW accelerated) 3D graphics.
- Stereoscopic 3D mode with a single graphics plane.
- Stereoscopic 3D mode with pseudo-3D graphics.
- Stereoscopic 3D mode with volumetric (typically HW accelerated) 3D graphics.

Table 1 shows the GEM display and draw modes that are defined in the present document.

Table 1: GEM display and draw modes

Graphics Mode	Display Mode	Graphic objects	Drawing API	Drawing model
Conventional 2D mode	2D display	Single graphics plane	AWT Graphics2D APIs, HAVi APIs	All AWT and HAVi drawing primitives are rendered on a 2D display
2D mode with volumetric 3D graphics	2D display	Volumetric 3D graphics	OpenGL ES API (JSR 239)	Geometric 3D objects are rendered to a 2D display
Limited stereoscopic 3D mode	Stereoscopic 3D display	Single graphics plane	AWT Graphics2D APIs, HAVi APIs	All AWT and HAVi drawing primitives on a single virtual plane
Full stereoscopic 3D mode	Stereoscopic 3D display	2 graphics planes	Synchronized drawing API as defined in clause 5.4.	Flat graphics on several virtual planes
Full stereoscopic 3D mode with volumetric 3D graphics	Stereoscopic 3D display	Volumetric 3D graphics	OpenGL ES API (JSR 239 [6])	Geometric 3D objects in a stereoscopic scene

The conventional 2D mode with a single graphics plane mode is available on all GEM terminals today.

All stereoscopic GEM terminals shall support at least the "Limited stereoscopic 3D mode". To ensure interoperability among different GEM terminals to the widest possible extent it is strongly recommended to also support the "Full stereoscopic 3D mode". These new stereoscopic modes and their capabilities and limitations are described in the following clauses.

The GEM graphics model is extended to support the new stereoscopic modes as illustrated in figure 3. There are several 3D specific controls that shall be returned on GEM platforms that support stereoscopic rendering. If a GEM terminal does not support stereoscopic video decoding, these new 3D controls shall not be available.

NOTE: Applications have to successfully execute both on platforms with stereoscopic 3D support and on non-3D platforms. To avoid problems on eager linking VM implementations it is strongly recommended to also include these classes on platforms without 3D support.

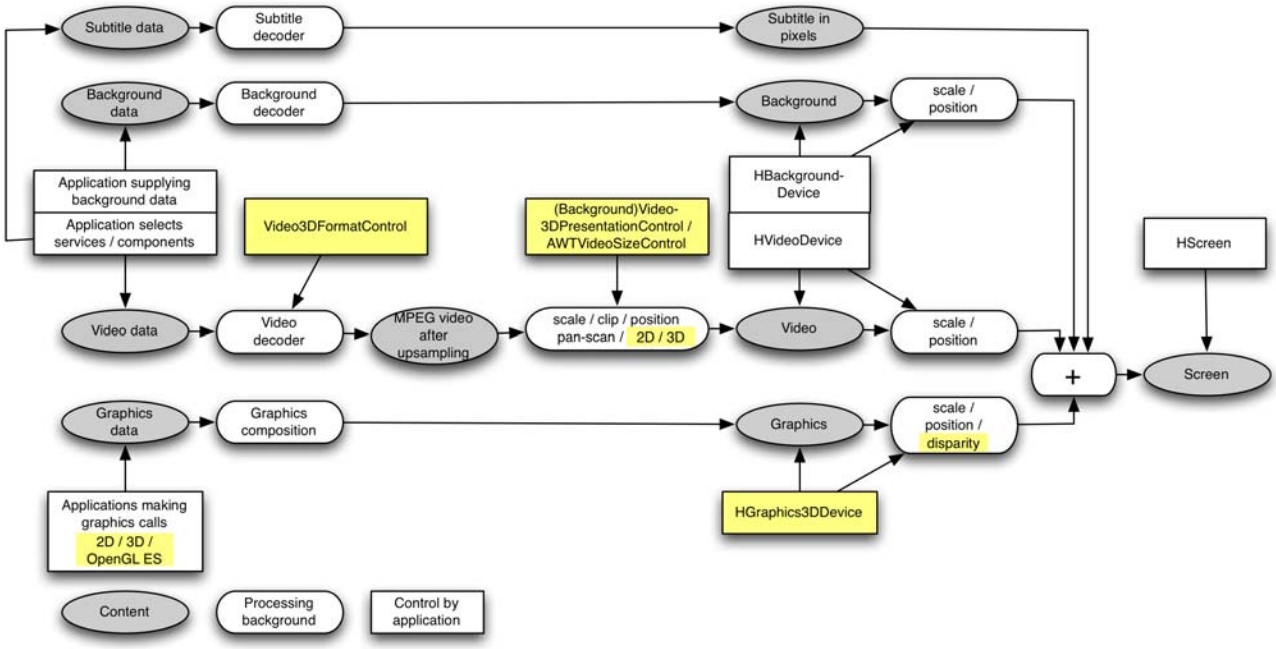

Figure 3: GEM 3D-enabled graphics model

4.2.1 Graphics modes

4.2.1.1 Conventional 2D mode

The 2D mode of GEM is fully specified in TS 102 728 [1] and is supported in all targets of the current GEM specification. The present document does not put any new requirements on this mode.

4.2.1.2 2D mode with volumetric 3D graphics

The 2D mode with volumentric 3D graphic objects allows to create a 3D scene consisting of OpenGL ES objects. The scene is logically positioned in front of the video and subtitle scene and is rendered to the 2D graphics plane.

This mode is supporting the OpenGL ES 1.1 API as defined in JSR 239 [6] with the restrictions and definitions as specified in clause 4.4.1.3.

4.2.1.3 Limited Stereoscopic 3D mode with a single graphics plane

The limited stereoscopic 3D mode with a single graphics plane is the basic 3D mode, where the 2D graphics plane is positioned in front of the video scene. The regular AWT and HAVi drawing APIs are used to draw to the 2D plane, an offset-setting API of the `HGraphics3DDevice` class (see clause 5.2) can be used to shift the depth position of the virtual 2D plane to always be in front of the video and subtitles.

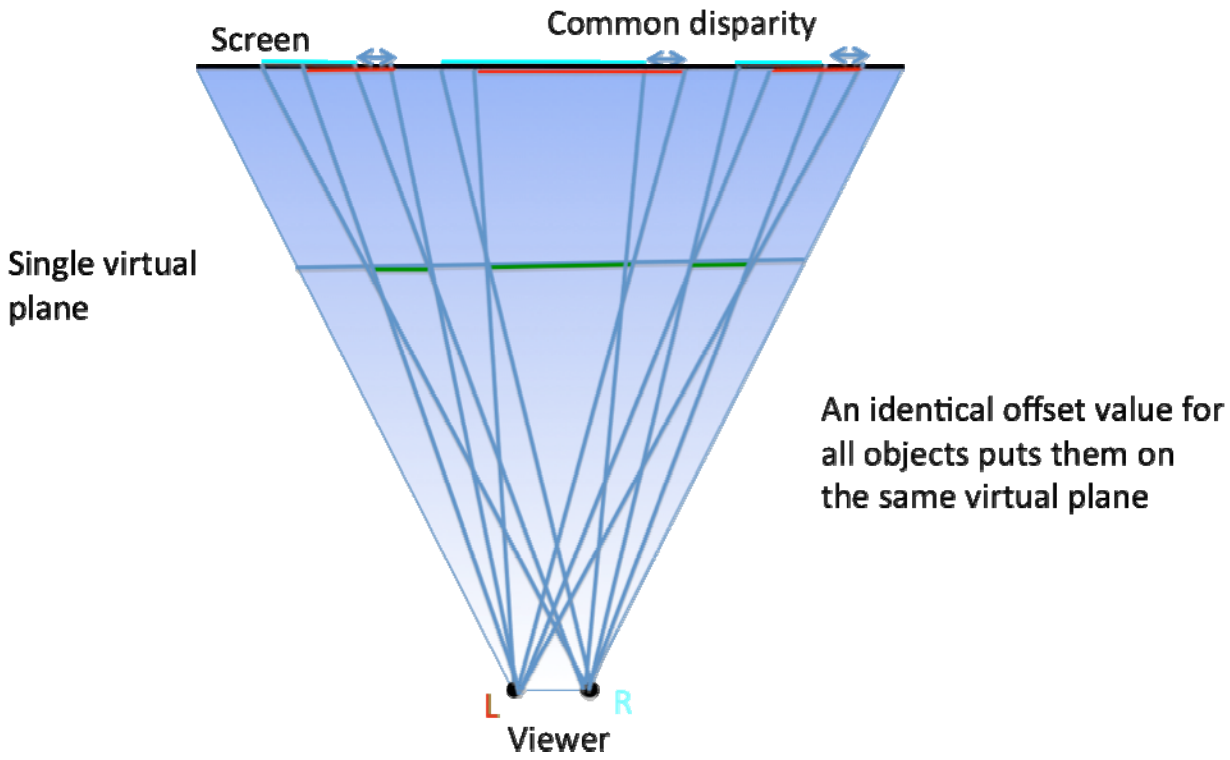

Figure 4: Single graphics plane mode

In this case the image buffers for the left and for the right eyes are exactly the same. They are shifted by a common disparity offset and as a result a user is seeing a single flat, virtual plane placed at a fixed depth in the 3D space.

To avoid any flickering the display engine must be synchronized with the TV refresh rate and in a case when any change in the displayed scene appears it shall wait to the end of screen refreshment. The platform ensures that all draw operations and graphic updates are always synchronized.

4.2.1.4 Stereoscopic 3D mode with pseudo-3D graphics (2 graphics planes)

The stereoscopic 3D mode with pseudeo-3D graphics objects allows to place flat objects at various depths in front of the video and subtitle scene. This is achieved with 2 graphics planes (one for each eye) and a drawing API (see clause 5.4 for details), which ensures that all draw operations and graphic updates to the two planes are always synchronized.

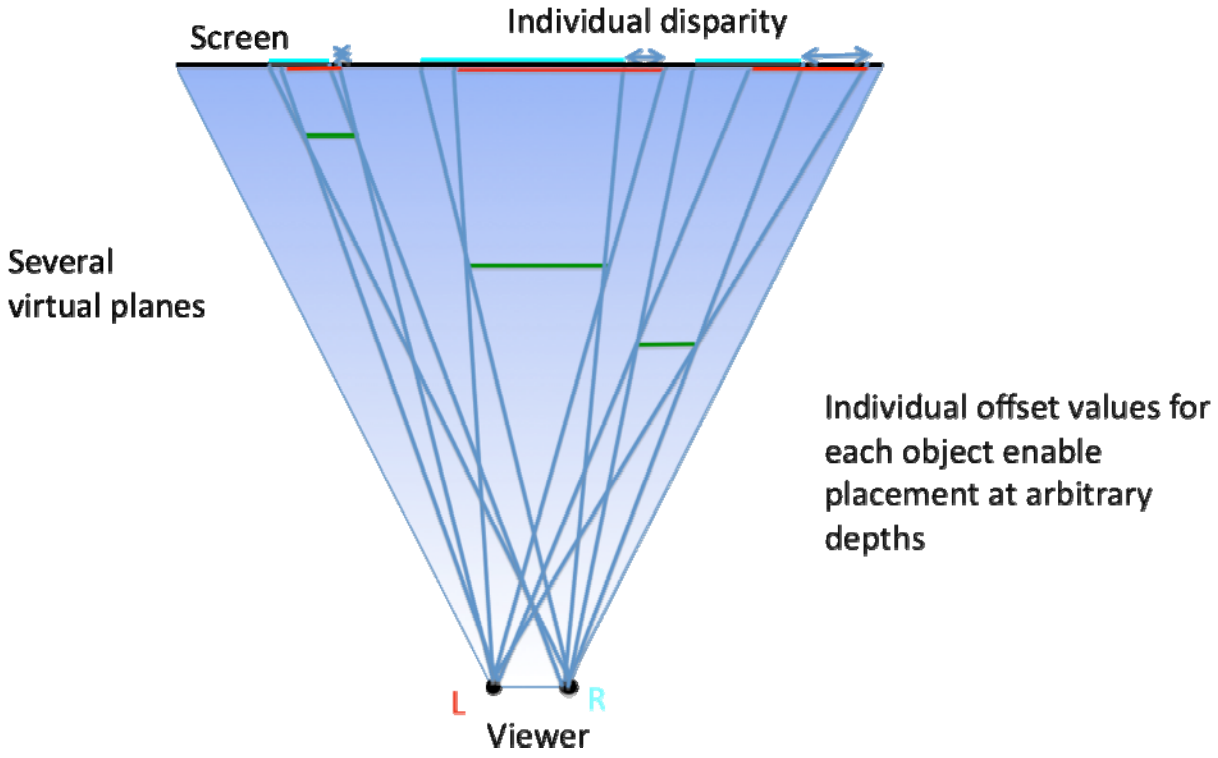

Figure 5: Two graphics plane mode

In this case images for the left and for the right eyes are different. Each of the graphic objects can be on a separate virtual plane with a different depth. As a consequence they have a different position in the image buffers for the left and right eye.

4.2.1.5 Stereoscopic 3D mode with volumetric 3D graphics

The stereoscopic 3D mode with volumentric-3D graphics objects allows to create a 3D scene consisting of volumetric 3D objects. The scene is logically positioned in front of the video scene and is rendered to 2 graphics planes (one for each eye).

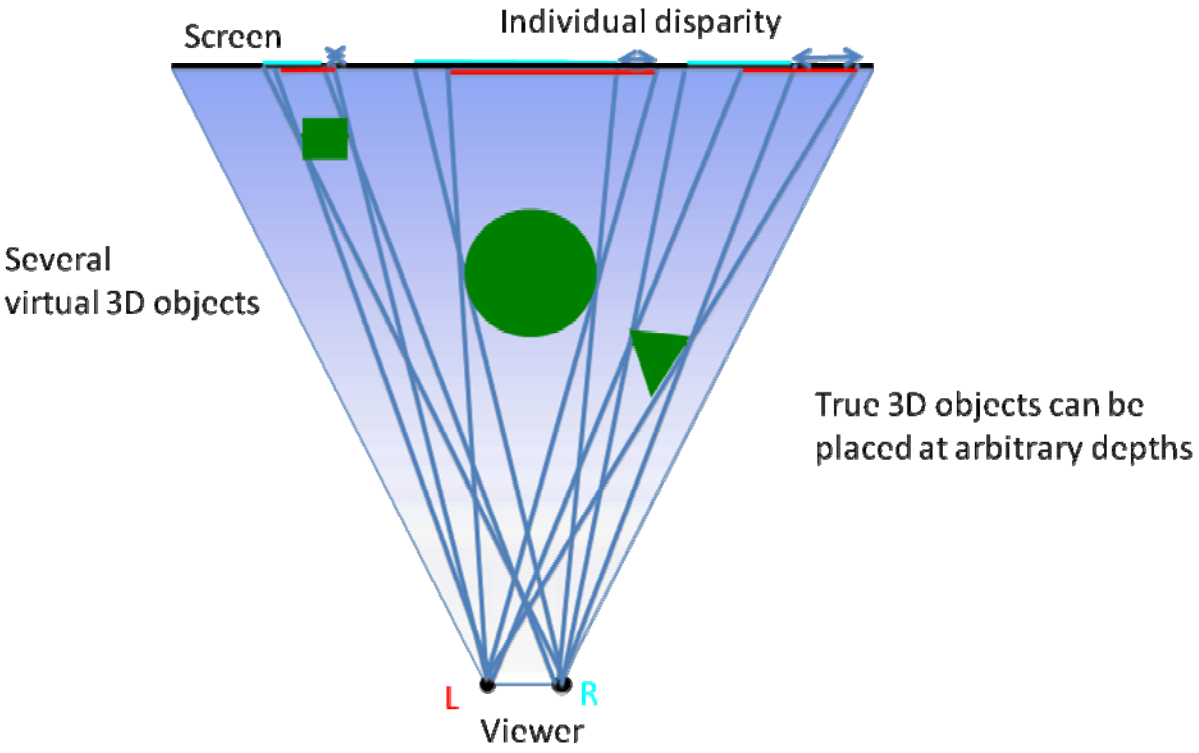

Figure 6: Volumetric 3D mode

This 3D mode is only applicable on devices, which support hardware accelerated 3D graphics.

In this case the image buffers for the left and the right eyes are different. Each 3D object can have a different distance to the viewer and as a consequence they must have a different position in the image buffers for the left and right eye.

This mode is supporting the OpenGL ES 1.1 API as defined in JSR 239 [6] with the extensions, clarifications and restrictions as described in clause 4.4.

4.3 Display engine

The display engine is responsible for a proper placing of any on-screen graphic on the top of a 3DTV content. This approach guarantees that any existing application will be properly displayed on the top of a 3DTV content. The display engine is aware of the current values of minimum and maximum disparity of the 3DTV content and sets a proper initial disparity offset for the on-screen graphics taking into account the comfort zone preference.

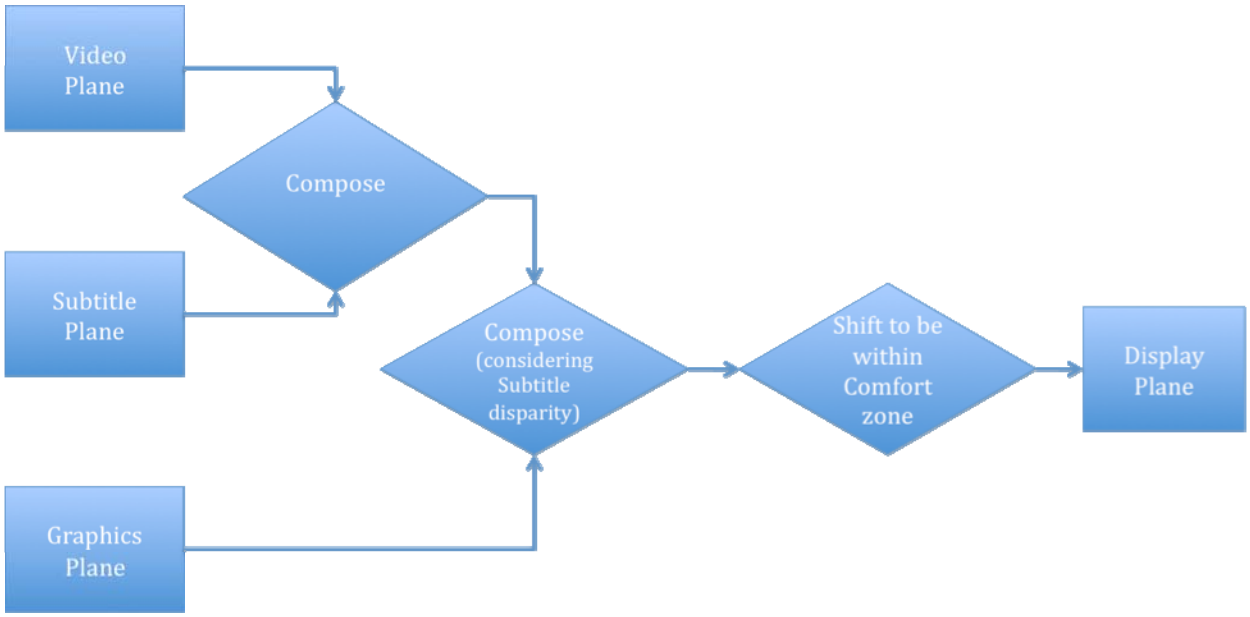

Figure 7: Layer composition / blending

The display engine takes into consideration the following cases when a comfort zone and disparity of a displayed scene (3DTV content + Subtitle planes + Graphic planes) overlaps each other:

a) Minimum disparity of the scene is closer to a user than comfort zone's minimum value. In this case the display engine shall put the scene further to a place where the comfort zone starts (shift offset = minimum disparity - minimum comfort zone).

b) Maximum disparity of the scene is further away than comfort zone's maximum value. In this case display engine shall put the scene closer to a place where the comfort zone ends, but not closer than comfort zone begins (shift offset = maximum disparity - maximum comfort zone).

c) Both a) and b) cases appears at the same time - the scene exceed minimum and maximum comfort zone values. In this case a) scenario shall have higher priority, therefore the scene shall be put further to a place where the comfort zone starts.

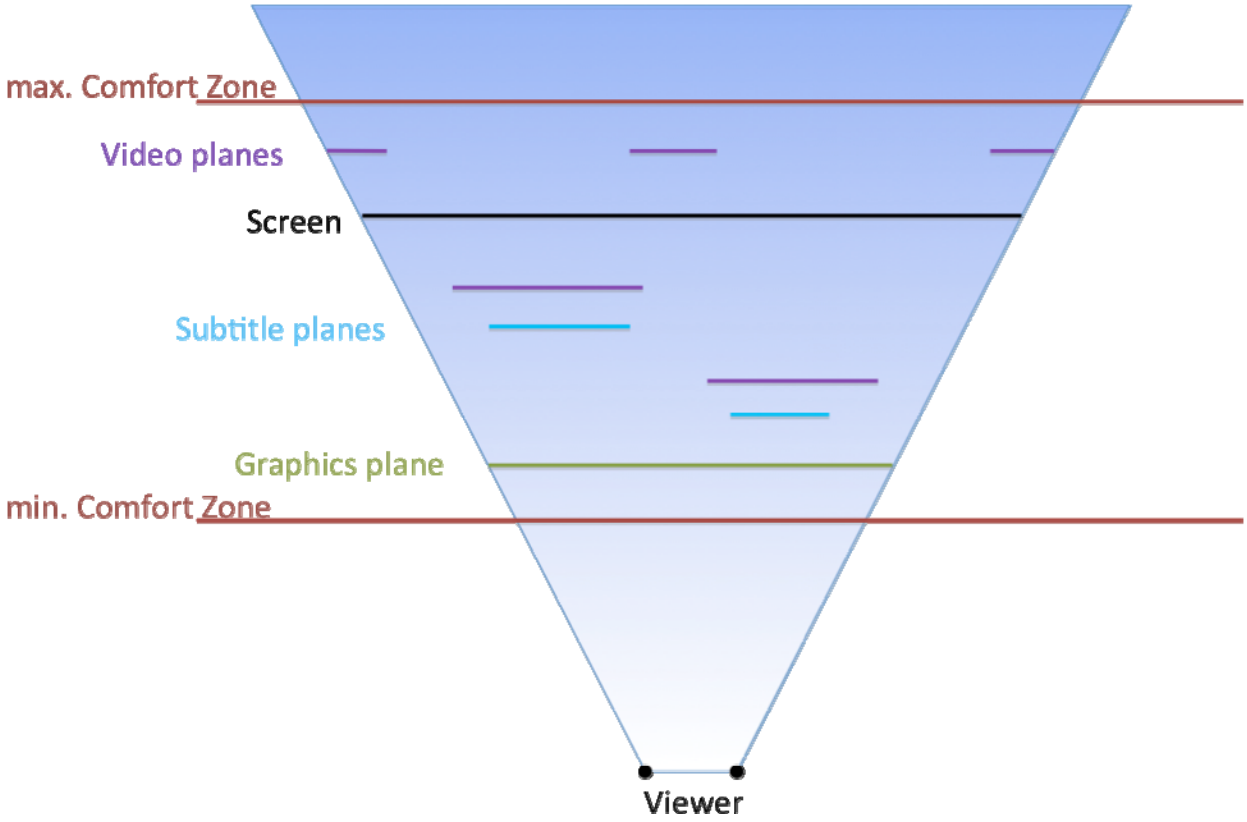

Figure 8: Video, subtitle and graphics plane places within Comfort Zone range

The display subsystem can automatically ensure that the graphics planes are always positioned in front of the subtitles. This can be achieved without application influence by the display subsystem, which is applying the maximum negative disparity value of all subtitle regions as an offset to the graphics planes for the left and right eye. The application can choose, whether it uses the automatic shifting of the graphics plane or handles the disparity offset by itself. In the latter case it can switch off the shifting in the `Display` class.

4.3.1 Disparity handling

Disparity denotes the difference between the horizontal positions of a pixel representing the same point in space in the right and left views. Positive disparity (horizontal right coordinate greater than horizontal left coordinate) implies a position behind the plane of display, and negative disparity implies a position in front of the display.

Disparity information for 3DTV video content is transmitted in the video depth range descriptor, specified in clause 6.4.12 of [4].

Disparity information for the subtitles is transmitted with DVB subtitles as described in clause 7.2.7 of the DVB subtitle specification [5].

The disparity is measured in number of pixels on reference screen (a screen with a horizontal size of 11 520 pixels).

Minimum disparity: Minimum disparity identifies the intended smallest disparity according to the current production guidelines, which corresponds to an object closest by the viewer.

Maximum disparity: Maximum disparity identifies the intended largest disparity according to the current production guidelines, which corresponds to an object at infinity, away from the viewer. If infinity disparity is unknown, then the disparity of the "furthest away object" should be given.

Comfort Zone: The comfort zone represents the amount of disparity that is acceptable to most people. In a typical TV set at home, where people sit close to a TV, the comfort zone ranges are from about -1 inch to 1 inch of disparity. Objects with -1 inch of disparity will appear at about 70 % of the distance from a user to the screen, and those objects with 1 inch of disparity at 165 % of the screen distance. In the case of children, the comfort zone is narrower than for adults. It is because the distance between child eyes is narrower than adults.

The value of the comfort zone depends on the viewing environment of the user and on personal viewing preferences. GEM includes an API to allow setting the comfort zone in clause 5.1.

4.3.1.1 Video Disparity

Video disparity information for 3DTV content is transmitted in the video depth range descriptor as specified in clause 6.4.12 of [4]. It is measured as the number of pixels on a reference screen with a horizontal resolution of 11 520 pixels.

The video disparity information may be used without application interference by the display engine to ensure proper placement of the subtitle and graphics.

4.3.1.2 Subtitle Disparity

3D subtitles can occupy several regions with different disparity values as specified in [5]. To avoid collisions between the graphics layer and the subtitle regions all changes of subtitle disparity are indicated to the application with a disparity event.

4.3.2 Disparity events

`DisparityChangeEvent` is used to inform an application about changes in the disparity of presented TV service. The application can use this value in a case when it is positioning its graphical planes on the top of other by itself.

`DisparityChangeEvent` informs about the minimum disparity. It means, when subtitles are visible, changes in subtitle's disparity shall be taken into consideration together with disparity changes in video planes, otherwise disparity changes in video plane is used.

It is a risk that disparity may be changed with every frame. To prevent device overloading with too many events per second, the `Video3DFormatControl` responsible for `DisparityChangeEvent` may limit the amount of them to an implementation dependent rate (e.g. maximum 5 per second).

4.4 Stereoscopic OpenGL ES

The OpenGL ES API as defined in JSR 239 [6] provides a Java binding for rendering of an OpenGL ES scene to a single graphics plane.

Stereoscopic support for JSR 239 is made available via an extension of the display subsystem, which permits to set scene parameters for the stereoscopic rendering to two frame buffers. This extension is transparent to the OpenGL ES API, i.e. JSR 239 [6] is used as it would be used for monoscopic rendering. There are several parameters, that influence the stereoscopic rendering of a 3D scene as illustrated in figure 9.

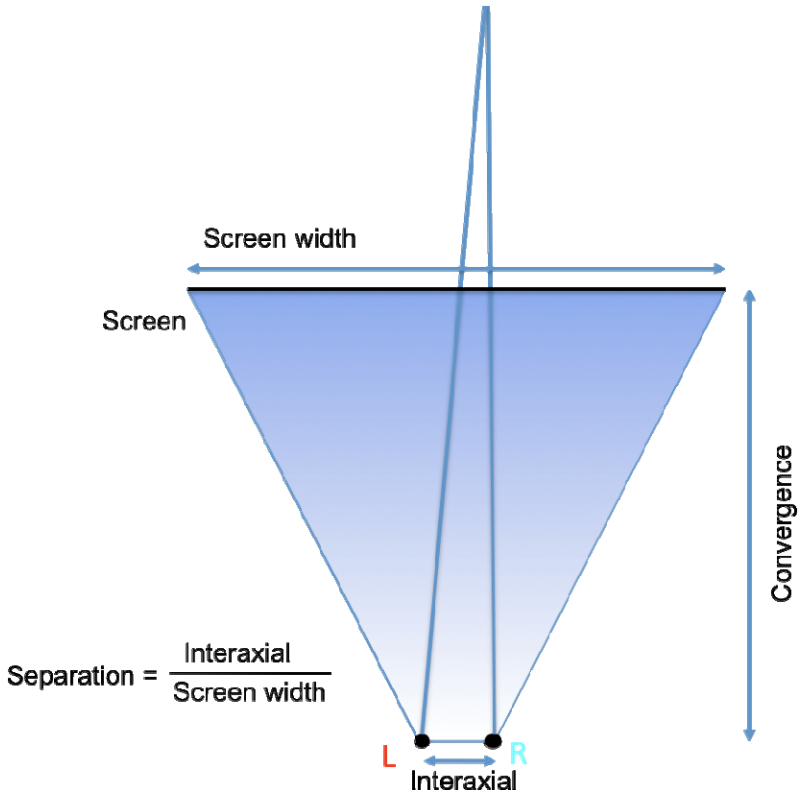

Figure 9: Stereoscopic OpenGL ES

4.4.1 Stereoscopic OpenGL API extensions

4.4.1.1 Definitions for specific terms used in this clause

Depth: A distance from the camera to a vertex.

OpenGL World Units: OpenGL specific units used to define 3D scene. In the Projection and Viewport Transform steps of the OpenGL 3D scene rendering pipeline these units are converted to pixels.

Parallax: Term similar to Disparity, a difference is in units: disparity is measured in pixels, parallax is expressed in OpenGL World Units.

The **convergence** parameter denotes which part of 3D scene will be placed at the screen distance from a viewer. After rendering 3D scene and splitting it for the left and right images, vertex with depth equals to convergence value will have 0 parallax and for a viewer those points will be placed at the screen; vertex with depth greater than convergence will have positive parallax and will be placed further than a screen; vertex with depth smaller than convergence will have negative parallax and will be placed closer than a screen.

The convergence is represented in the OpenGL World Units.

The convergence must be set by a 3D scene designer before scene rendering.

The convergence parameter influence the rendering of the OpenGL ES scene only; it is not relevant for the AWT-based stereoscopic modes. The convergence value can be queried and set with the corresponding methods of the `Display` class (see clause 5.1 for details).

4.4.1.2 Implementation guideline for stereoscopic OpenGL rendering pipeline

The **separation** parameter determines the normalized form of the inter-eye distance. Vertex with parallax value approximate to this value will be placed at infinity for a viewer.

The Separation value can be calculated from the maximum comfort zone value (described in clause 4.3.1) with the following formula:

$$Separation = \frac{Comfort\ zone's\ maximum\ value}{11520} * HScreen's\ horizontal\ resolution$$

A larger separation value increases the perceived 3D effect, a separation value of 0 results in a monoscopic image.

Comfort zone's maximum value needed to calculate Separation can be queried from `Display` class (see clause 5.1 for details).

4.4.1.2.1 Stereo 3D scene rendering pipeline.

In the standard mono 3D scene rendering, the following pipeline is always being used:

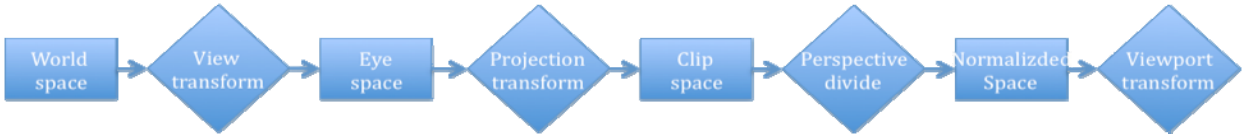

Figure 10: Standard mono 3D rendering pipeline

In the stereo 3D scene rendering one more step has been added and pipeline looks like in figure 11:

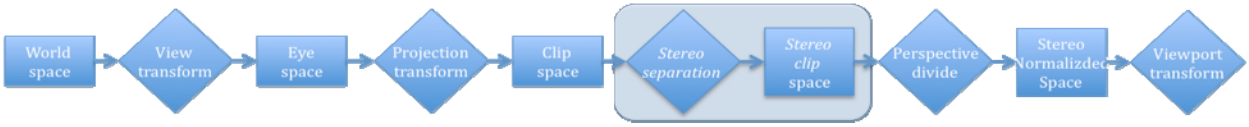

Figure 11: Stereo 3D rendering pipeline

Stereo separation step generates two images: one for the left eye, the second for the right. This step offsets the x position of each vertex by the parallax amount using the following formula:

$$clipPos.x+ = EyeSign * Separation * (clipPos.w - Convergence)$$

$$EyeSign = \begin{cases} +1\ for\ right \\ -1\ for\ left \end{cases}$$

When Stereo Normalized Space is ready, next step is Viewport Transform. In this step, points coordinates are transformed from OpenGL Normalized Units (a OpenGL World Units in the range [-1, 1]) into pixels appropriately to fit into defined viewport.

4.4.1.3 Clarifications and restrictions on JSR 239

An EGL object is obtained by calling `EGLContext.getEGL()`. JSR 239 permits that the returned object may implement the EGL10 interface only (if the underlying EGL implementation only supports EGL 1.0), or it may additionally implement the EGL11 interface (if the underlying implementation supports EGL 1.1).

To ensure interoperability, all implementations conformant to the present document shall support EGL 1.1, i.e. the return value of `EGLContext.getEGL()` is an instance of `javax.microedition.khronos.egl.EGL11`.

When Open GL ES is supported, EGL rendering is to instances of `org.havi.ui.HScene`, to the instance returned from the `javax.tv.graphics.TVContainer.getRootContainer(XletContext)` and to instances of `java.awt.image.BufferedImage` and `java.awt.image.VolatileImage` and `pBuffer`.
These instances must be supported as the `native_window` parameter to `eglCreateWindowSurface()`.

The EGL_DEFAULT_DISPLAY token is used to specify a display, only the default display is guaranteed to be available.

5 GEM 3D API Packages

This package contains the API packages, which define the GEM stereoscopic 3D profile.

Package Summary	
org.dvb.stereoscopic.graphics	All stereoscopic drawing APIs shall synchronize all drawing operations to the left and right eye to avoid flickering / tearing artifacts.
org.dvb.stereoscopic.media	The media package contains classes and interfaces associated to 3D video decoding.
org.dvb.stereoscopic.system	Classes and interfaces provide information about display and device capabilities related to 3D.
org.dvb.stereoscopic.ui	This package extends the HAVi and AWT drawing model with stereoscopic 3D draw modes and drawing APIs.

It extends the GEM platform with the following functionalities:

- A stereoscopic drawing API in the package org.dvb.stereoscopic.graphics.
- An API to handle the stereoscopic video presentation in the package org.dvb.stereoscopic.media.
- An API to query the stereoscopic capabilities of the environment in the package org.dvb.stereoscopic.system.
- An API to switch between different stereoscopic display modes in the package org.dvb.stereoscopic.ui.

A compliant implementation of GEMS3D shall include all APIs from these packages, subsetting is not permitted.

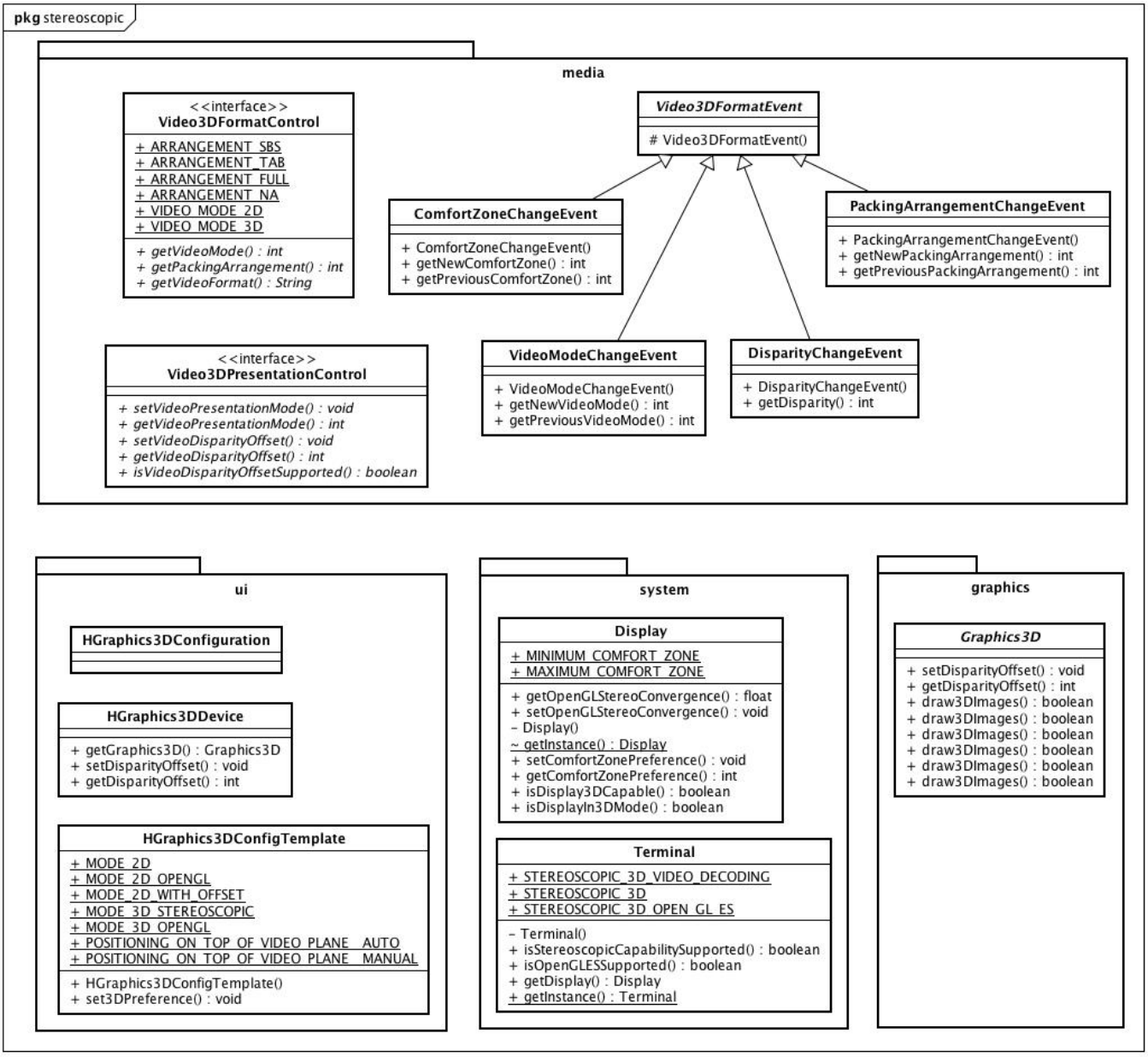

Figure 12: Architecture Overview

5.1 Display and Device capabilities

Package org.dvb.stereoscopic.system

Classes and interfaces provide information about display and device capabilities related to 3D.

Class Summary	
Display	This class provides a way to query the capabilities of the display subsystem.
Terminal	The terminal allows an application to check stereoscopic and OpenGL ES capability of terminal and to obtain display instance.

Display

```
java.lang.Object
  └org.dvb.stereoscopic.system.Display
```

```
public class Display
extends java.lang.Object
```

This class provides a way to query the capabilities of the display subsystem.

Fields

MAXIMUM_COMFORT_ZONE

```
public static final int MAXIMUM_COMFORT_ZONE
```

A value for use to specify furthest point in the 3D image for viewer comfort zone.

MINIMUM_COMFORT_ZONE

```
public static final int MINIMUM_COMFORT_ZONE
```

A value for use to specify nearest point in the 3D image for viewer comfort zone.

Methods

getComfortZonePreference

```
public int getComfortZonePreference(int preference)
                                    throws java.lang.IllegalArgumentException
```

Return the of comfort zone preference in pixels.

Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels).

Parameters:

`preference` - the comfort zone preference to be indicated. Valid values are: MINIMUM_COMFORT_ZONE, MAXIMUM_COMFORT_ZONE.

Returns:

the number of pixels for the specified comfort zone preference.

Throws:

`java.lang.IllegalArgumentException` - is thrown if is not a valid value of comfort zone preference.

getOpenGLStereoConvergence

```
public float getOpenGLStereoConvergence()
```

This method returns the stereo convergence. Convergence in measured in OpenGL World Units

Returns:

the stereoConvergence

isDisplay3DCapable

```
public boolean isDisplay3DCapable()
```

If the display has 3D capability, this method returns true, otherwise false.

Returns:

If the display has 3D capability, this method returns true, otherwise false.

isDisplayIn3DMode

```
public boolean isDisplayIn3DMode()
```

If the display is currently in 3D mode, this method returns true, otherwise false.

Returns:

If the display is currently in 3D mode, this method returns true, otherwise false.

setComfortZonePreference

```
public void setComfortZonePreference(int preference,
                                     int pixels)
                              throws java.lang.IllegalArgumentException
```

Set the comfort zone preference in pixels.

Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels).

Parameters:

`preference` - the comfort zone preference to be indicated. Valid values are: MINIMUM_COMFORT_ZONE, MAXIMUM_COMFORT_ZONE.

`pixels` - the number of pixels associated with comfort zone preference.

Throws:

`java.lang.IllegalArgumentException` - is thrown if is not a valid value of comfort zone preference.

setOpenGLStereoConvergence

```
public void setOpenGLStereoConvergence(float stereoConvergence)
                                throws java.lang.IllegalArgumentException
```

This method sets the stereo convergence

Parameters:

`stereoConvergence` - the stereoConvergence to set in OpenGL World Units

Throws:

`java.lang.IllegalArgumentException`

Terminal

```
java.lang.Object
  └org.dvb.stereoscopic.system.Terminal
```

```
public class Terminal
extends java.lang.Object
```

The terminal allows an application to check stereoscopic and OpenGL ES capability of terminal and to obtain display instance.

Fields

STEREOSCOPIC_3D
`public static final int` **`STEREOSCOPIC_3D`**

Stereoscopic 3D capability

STEREOSCOPIC_3D_OPEN_GL_ES
`public static final int` **`STEREOSCOPIC_3D_OPEN_GL_ES`**

Stereoscopic 3D OpenGL ES capability

STEREOSCOPIC_3D_VIDEO_DECODING
`public static final int` **`STEREOSCOPIC_3D_VIDEO_DECODING`**

Stereoscopic 3D video decoding capability

Methods

getDisplay
`public Display` **`getDisplay`**`()`

Returns the Display instance

Returns:

the Display instance

getInstance
`public static Terminal` **`getInstance`**`()`

This method returns the sole instance of the Terminal class. The Terminal class is a singleton.

Returns:

the instance of the Terminal.

isOpenGLESSupported
`public boolean` **`isOpenGLESSupported`**`()`

If the terminal supports OpenGL ES, this method returns true, otherwise false.

Returns:

true if terminal supports OpenGL ES, false otherwise

isStereoscopicCapabilitySupported
`public boolean` **`isStereoscopicCapabilitySupported`**`(int capability)`

If the terminal supports given stereoscopic capability, this method returns true, otherwise false.

Parameters:

`capability` - the capability to check. Valid values are: STEREOSCOPIC_3D_VIDEO_DECODING, STEREOSCOPIC_3D, STEREOSCOPIC_3D_OPEN_GL_ES

Returns:

true if terminal supports specified stereoscopic capability, otherwise false

5.2 Graphics modes

Package org.dvb.stereoscopic.ui

This package extends the HAVi and AWT drawing model with stereoscopic 3D draw modes and drawing APIs.

Class Summary	
HGraphics3DConfigTemplate	The HGraphics3DConfigTemplate class is used to obtain a valid `HGraphics3DConfiguration`.
HGraphics3DConfiguration	The `HGraphics3DConfiguration` class describes the characteristics (settings) of an `HGraphics3DDevice`.
HGraphics3DDevice	The `HGraphics3DDevice` class describes the stereoscopic raster graphics devices that are available for a particular `HScreen`.

HGraphics3DConfigTemplate

```
java.lang.Object
  └org.havi.ui.HScreenConfigTemplate
      └org.havi.ui.HGraphicsConfigTemplate
          └org.dvb.stereoscopic.ui.HGraphics3DConfigTemplate
```

```
public class HGraphics3DConfigTemplate
extends org.havi.ui.HGraphicsConfigTemplate
```

The HGraphics3DConfigTemplate class is used to obtain a valid `HGraphics3DConfiguration`. An application instantiates one of these objects and then sets all non-default attributes as desired. The `HGraphicsDevice.getBestConfiguration(org.havi.ui.HGraphicsConfigTemplate)` method found in the `HGraphicsDevice` class is then called with this `HGraphics3DConfigTemplate` . A valid `HGraphics3DConfiguration` is returned that meets or exceeds what was requested in the `HGraphics3DConfigTemplate`.

A new instance of the HGraphics3DConfigTemplate has a default value of MODE_2D with a default priority HScreenConfigTemplate.REQUIRED.

Fields

MODE_2D

```
public static final int MODE_2D
```

A constant value indicating a graphics device mode of 2D presentation. In this mode, there is one framebuffer that presents to both eyes at the same position.

MODE_2D_OPENGL

```
public static final int MODE_2D_OPENGL
```

A constant value indicating a graphics device that presents an Open GL ES scene to a single 2D frame buffer.

MODE_2D_WITH_OFFSET

```
public static final int MODE_2D_WITH_OFFSET
```

A constant value indicating a graphics device mode of a single 2D framebuffer that is presented in a 3D view at a given depth.

MODE_3D_OPENGL
```
public static final int MODE_3D_OPENGL
```

A constant value indicating a graphics device that presents a full 3D scene by having two framebuffers, one for each eye.

MODE_3D_STEREOSCOPIC
```
public static final int MODE_3D_STEREOSCOPIC
```

A constant value indicating a graphics device that presents a full 3D scene by having two framebuffers, one for each eye.

POSITIONING_ON_TOP_OF_VIDEO_PLANE__AUTO
```
public static final int POSITIONING_ON_TOP_OF_VIDEO_PLANE__AUTO
```

A constant value indicating a display engine that it has to position presented scene on top of other planes (video and subtitle)

NOTE: This property does not have any influence when MODE_3D_OPENGL is being used

POSITIONING_ON_TOP_OF_VIDEO_PLANE__MANUAL
```
public static final int POSITIONING_ON_TOP_OF_VIDEO_PLANE__MANUAL
```

A constant value indicating that xlet will be positioning presented scene on top of other planes (video and subtitle) by itself

NOTE: This property does not have any influence when MODE_3D_OPENGL is being used

REQUIRED
```
public static final int REQUIRED
```

Constructors

HGraphics3DConfigTemplate
```
public HGraphics3DConfigTemplate()
```

Creates an `HGraphics3DConfigTemplate` object.

Methods

set3DPreference
```
public void set3DPreference(int value, int priority)
```

Set the preference for 3D display to the given value, and assign it the given priority. If this preference has been previously set, then the previous value and priority shall be overwritten.

The HGraphics3DConfigTemplate has default values:

- MODE_2D and a default priority of HScreenConfigTemplate.REQUIRED,
- POSITIONING_ON_TOP_OF_VIDEO_PLANE__AUTO and a default priority of HScreenConfigTemplate.REQUIRED

Parameters:

`value` - The value of the 3D preference, either MODE_2D, MODE_2D_OPENGL, MODE_2D_WITH_OFFSET or MODE_3D_STEREOSCOPIC, MODE_3D_OPENGL using the constants defined in HGraphicsConfiguration3D.

`priority` - The priority of the preference. Valid values include: HScreenConfigTemplate.REQUIRED, HScreenConfigTemplate.PREFERRED, HScreenConfigTemplate.DONT_CARE, HScreenConfigTemplate.PREFERRED_NOT and HScreenConfigTemplate.REQUIRED_NOT

Throws:

`java.lang.IllegalArgumentException` - if value is valid as defined here, or if priority is not valid as defined here.

HGraphics3DConfiguration

```
java.lang.Object
  └org.havi.ui.HScreenConfiguration
      └org.havi.ui.HGraphicsConfiguration
          └org.dvb.stereoscopic.ui.HGraphics3DConfiguration
```

```
public class HGraphics3DConfiguration
extends org.havi.ui.HGraphicsConfiguration
```

The `HGraphics3DConfiguration` class describes the characteristics (settings) of an `HGraphics3DDevice`. There can be many `HGraphics3DConfiguration` objects associated with a single `HGraphics3DDevice`.

On a 3D-capable GEM terminal, all platform methods that are specified to return `HGraphicsConfiguration` shall return an instance of `HGraphics3DDevice` under all circumstances.

Constructors

HGraphics3DConfiguration

```
protected HGraphics3DConfiguration()
```

HGraphics3DDevice

```
java.lang.Object
  └org.havi.ui.HScreenDevice
      └org.havi.ui.HGraphicsDevice
          └org.dvb.stereoscopic.ui.HGraphics3DDevice
```

All Implemented Interfaces:

org.davic.resources.ResourceProxy, org.davic.resources.ResourceServer

```
public class HGraphics3DDevice
extends org.havi.ui.HGraphicsDevice
```

The `HGraphics3DDevice` class describes the stereoscopic raster graphics devices that are available for a particular `HScreen`. Each `HGraphics3DDevice` has one or more `HGraphics3DConfiguration` objects associated with it. These objects specify the different configurations (settings) in which the `HGraphics3DDevice` can be used.

Constructors

HGraphics3DDevice

```
public HGraphics3DDevice()
```

Methods

getDisparityOffset

```
public int getDisparityOffset()
```

Returns the current graphics disparity offset in pixels.

Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels).

Returns:

the current video disparity offset in pixels.

getGraphics3D

```
public Graphics3D getGraphics3D()
```

Returns the Graphics3D instance associated with this HGraphics3DDevice

Returns:

the Graphics3D instance associated with this HGraphics3DDevice

setDisparityOffset

```
public void setDisparityOffset(int disparity)
                        throws java.lang.IllegalArgumentException
```

This method allows to set a disparity offset for the HGraphics3DDevice. A positive offset shifts the graphics layer further away from the viewer, a negative offset brings it closer.
A disparity offset of zero places the virtual plane on the screen.

Parameters:

`disparity` - pixel offset to be used for shifting the left and right graphics layers.

Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels).

Throws:

`java.lang.IllegalArgumentException`

5.3 Video Controls

Package org.dvb.stereoscopic.media

The media package contains classes and interfaces associated to 3D video decoding.

Interface Summary	
Video3DFormatControl	Provides a means for applications to get information associated with the 3D video being presented to the user.
Video3DPresentationControl	The Video3DPresentationControl is a JMF control that allows to select the video decoding mode (2D or 3D) and allows to set an additional disparity offset to the video, if supported by the platform.

Class Summary	
DisparityChangeEvent	Report that disparity of most on top plane (video or subtitle) in a plano-stereoscopic 3DTV service has been changed.
PackingArrangementChangeEvent	Event signaling that the packing arrangement of the transmitted video has changed.
Video3DFormatEvent	The base class for all other events relating to changes in 3D video format.
VideoModeChangeEvent	Event signaling that the video mode of the transmitted video has changed.

Description

This package defines two new JMF controls, that can be used to obtain information about the stereoscopic format of the current stereoscopic movie and to change the video presentation mode (2D or 3D, disparity offset).

The `Video3DPresentationControl` can be used to select or query the video presentation mode (2D or 3D) and allows to set an additional disparity offset shift to the video, if this feature is supported by the platform.

The `Video3DFormatControl` enables applications to query information about the current video stream (video mode and frame packing arrangement).

Video3DFormatControl

All Superinterfaces:
javax.media.Control, org.dvb.media.VideoFormatControl

```
public interface Video3DFormatControl
extends org.dvb.media.VideoFormatControl
```

Provides a means for applications to get information associated with the 3D video being presented to the user.

Fields

ARRANGEMENT_FULL
```
static final int ARRANGEMENT_FULL
```

Describes stereoscopic video with full frame video for the left and right eye

ARRANGEMENT_NA
```
static final int ARRANGEMENT_NA
```

Describes a case when packing arrangement is not available (e.g. 2D content)

ARRANGEMENT_SBS
```
static final int ARRANGEMENT_SBS
```

Describes Side By Side packing arrangement

ARRANGEMENT_TAB
```
static final int ARRANGEMENT_TAB
```

Describes Top And Bottom packing arrangement

VIDEO_MODE_2D
```
static final int VIDEO_MODE_2D
```

Describes 2D video service

VIDEO_MODE_3D

```
static final int VIDEO_MODE_3D
```

Describes 3D video service

Methods

getPackingArrangement

```
int getPackingArrangement()
```

Returns current packing arrangement in 3D content

Returns:

packing arrangement

getVideoFormat

```
java.lang.String getVideoFormat()
```

Get the video format of associated stream

Formats are encoded as MPEG-7 term IDs as extended by the DVB IPI handbook, e.g. MPEG-2 video, main profile, main level is encoded as "2.2.2".

Returns:

video format of the stream

getVideoMode

```
int getVideoMode()
```

Returns content's video mode: 2D or 3D

Returns:

content's video mode

Video3DPresentationControl

All Superinterfaces:
javax.media.Control, org.dvb.media.VideoPresentationControl

```
public interface Video3DPresentationControl
extends org.dvb.media.VideoPresentationControl
```

The Video3DPresentationControl is a JMF control that allows to select the video decoding mode (2D or 3D) and allows to set an additional disparity offset to the video, if supported by the platform.

Methods

getVideoDisparityOffset

```
int getVideoDisparityOffset()
```

Returns the current video disparity offset in pixels.

Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels).

Returns:

the current video disparity offset in pixels.

getVideoPresentationMode

```
int getVideoPresentationMode()
```

Returns current video presentation mode: 2D or 3D

Returns:

current video mode (VIDEO_MODE_2D, VIDEO_MODE_3D) as defined by Video3DFormatControl

isVideoDisparityOffsetSupported

```
boolean isVideoDisparityOffsetSupported()
```

Return value indicating whether the platform supports to set the video disparity.

Returns:

flag indicating whether the platform supports to set the video disparity.

setVideoDisparityOffset

```
void setVideoDisparityOffset(int theOffset)
                             throws java.lang.IllegalArgumentException,
                                    java.lang.UnsupportedOperationException
```

This method allows to set a video disparity offset.

Parameters:

`theOffset` - pixel offset to be added to the video disparity. A positive offset shifts the video further away from the viewer, a negative offset brings it closer.

Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels).

Throws:

`java.lang.IllegalArgumentException`

`java.lang.UnsupportedOperationException`

setVideoPresentationMode

```
void setVideoPresentationMode(int theMode)
                          throws java.lang.IllegalArgumentException,
                                 java.lang.UnsupportedOperationException
```

Sets the current video presentation (decoding) mode to 2D or 3D.

Parameters:

`theMode` - new video mode (VIDEO_MODE_2D, VIDEO_MODE_3D) as defined by Video3DFormatControl

Throws:

`java.lang.IllegalArgumentException` - If the new mode is not VIDEO_MODE_2D or VIDEO_MODE_3D.

`java.lang.UnsupportedOperationException` - If the new mode cannot be set.

DisparityChangeEvent

```
java.lang.Object
  └java.util.EventObject
      └org.dvb.media.VideoFormatEvent
          └org.dvb.stereoscopic.media.Video3DFormatEvent
```

```
└org.dvb.stereoscopic.media.DisparityChangeEvent
```

All Implemented Interfaces:
java.io.Serializable

```
public class DisparityChangeEvent
extends Video3DFormatEvent
```

Report that disparity of most on top plane (video or subtitle) in a plano-stereoscopic 3DTV service has been changed.

Constructors

DisparityChangeEvent
```
public DisparityChangeEvent(java.lang.Object source,
                            int disparity)
```

Constructor

Parameters:

`source` - the source of the event. The platform shall always pass in the JMF player presenting the subtitles

`disparity` - the current disparity of the most on top plane (video plane or subtitle plane if subtitles are presented to the user), (Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels)

Methods

getDisparity
```
public int getDisparity()
```

Returns the current disparity of the most on top plane (video or subtitle).

Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels).

Returns:

the current disparity of the most on top plane

PackingArrangementChangeEvent

```
java.lang.Object
  └java.util.EventObject
      └org.dvb.media.VideoFormatEvent
          └org.dvb.stereoscopic.media.Video3DFormatEvent
              └org.dvb.stereoscopic.media.PackingArrangementChangeEvent
```

All Implemented Interfaces:
java.io.Serializable

```
public class PackingArrangementChangeEvent
extends Video3DFormatEvent
```

Event signaling that the packing arrangement of the transmitted video has changed.

Constructors

PackingArrangementChangeEvent
```
public PackingArrangementChangeEvent(java.lang.Object source,
                                     int previousPackingArrangement,
                                     int newPackingArrangement)
```

Construct the event.

Parameters:

`source` - the source of the event

`previousPackingArrangement` - the previous packing arrangement of the transmitted video

`newPackingArrangement` - the new packing arrangement of the transmitted video

Methods

getNewPackingArrangement

`public int` **`getNewPackingArrangement`**`()`

Get the new packing arrangement of the transmitted video.

Returns:

the new packing arrangement of the video. The value of this is represented by one of the constants from the Video3DFormatControl class and shall be the value passed into the constructor of the event.

getPreviousPackingArrangement

`public int` **`getPreviousPackingArrangement`**`()`

Get the previous packing arrangement of the transmitted video.

Returns:

the previous packing arrangement of the video. The value of this is represented by one of the constants from the Video3DFormatControl class and shall be the value passed into the constructor of the event.

Video3DFormatEvent

```
java.lang.Object
  └java.util.EventObject
      └org.dvb.media.VideoFormatEvent
          └org.dvb.stereoscopic.media.Video3DFormatEvent
```

All Implemented Interfaces:
java.io.Serializable
Direct Known Subclasses:
DisparityChangeEvent, PackingArrangementChangeEvent, VideoModeChangeEvent

`public abstract class` **`Video3DFormatEvent`**
`extends org.dvb.media.VideoFormatEvent`

The base class for all other events relating to changes in 3D video format.

Constructors

Video3DFormatEvent

`protected` **`Video3DFormatEvent`**`(java.lang.Object source)`

Constructor.

Parameters:

`source` - the source of the event. The platform shall always pass in the JMF Player presenting the 3D video whose format changed.

VideoModeChangeEvent

```
java.lang.Object
  └java.util.EventObject
      └org.dvb.media.VideoFormatEvent
          └org.dvb.stereoscopic.media.Video3DFormatEvent
              └org.dvb.stereoscopic.media.VideoModeChangeEvent
```

All Implemented Interfaces:
java.io.Serializable

```
public class VideoModeChangeEvent
extends Video3DFormatEvent
```

Event signaling that the video mode of the transmitted video has changed.

Constructors

VideoModeChangeEvent

```
public VideoModeChangeEvent(java.lang.Object source,
                            int previousVideoMode,
                            int newVideoMode)
```

Construct the event.

Parameters:

`source` - the source of the event

`previousVideoMode` - the previous video mode of the transmitted video

`newVideoMode` - the new video mode of the transmitted video

Methods

getNewVideoMode

```
public int getNewVideoMode()
```

Get the new video mode of the transmitted video.

Returns:

the new video mode of the video. The value of this is represented by one of the constants from the Video3DFormatControl class and shall be the value passed into the constructor of the event.

5.4 Drawing API

Package org.dvb.stereoscopic.graphics

All stereoscopic drawing APIs shall synchronize all drawing operations to the left and right eye to avoid flickering / tearing artifacts.

Class Summary	
Graphics3D	This `Graphics3D` class extends the `DVBGraphics` class to enable stereoscopic AWT rendering.

Description

All stereoscopic drawing APIs shall synchronize drawing operations to the left and right eye to avoid flickering / tearing artifacts. The drawing API for flat 2D graphics mode is identical to the existing drawing API.

The drawing API for pseudo-3D graphics shall allow drawing 2D components (images, graphical shapes, text, lines).

The full-3D drawing API shall be based on standardized 3D libraries and support graphics hardware acceleration.

Graphics3D

```
java.lang.Object
  └java.awt.Graphics
      └org.dvb.ui.DVBGraphics
          └org.dvb.stereoscopic.graphics.Graphics3D
```

```
public abstract class Graphics3D
extends org.dvb.ui.DVBGraphics
```

This `Graphics3D` class extends the `DVBGraphics` class to enable stereoscopic AWT rendering. It shall be returned on 3D enabled terminals in all cases, where a non-3D enabled terminal would return a DVBGraphics instance.

Graphics Model

The underlying graphics model is a display with two frame buffers, one for the left eye and one for the right eye, where the display system ensures that each eye views its corresponding frame buffers. This can be achieved with different technical means, such as shutter glasses, polarizer glasses and others.

Synchronized Draw Operations

All draw operations from `Graphics2D` (and `Graphics`) are also available in `Graphics3D`. The rendering process of `Graphics2D` is extended to synchronize the rendering to the two frame buffers and guarantees that all draw operations get displayed for the left and right eyes simultaneously. If the display system is in a stereoscopic mode, all draw operations observe the disparity offset value, as described below. Additionally the Graphics3D class defines new image draw operations `draw3DImages` that allow to specify separate images for the left and the right eye.

Disparity Handling

The disparity offset influences the shift of the x position between the draw operations to the left and the right frame buffer. The disparity offset can be set with the method `setDisparityOffset`. The current value can be queried with `getDisparityOffset`.

Constructors

Graphics3D

```
protected Graphics3D(java.awt.Graphics2D g)
```

Methods

draw3DImages

```
public boolean draw3DImages(java.awt.Image imgLeft,
                            java.awt.Image imgRight,
                            int x,
                            int y,
                            java.awt.Color bgcolor,
                            java.awt.image.ImageObserver observer)
                     throws java.lang.IllegalStateException
```

Draws the two images as specified in the imgLeft and imgRight parameters to the corresponding frame buffers. The display system shall ensure that the drawing of both images is synchronized and they appear simultaneously on the screen.

The behavior of this method follows the definition of the corresponding `Graphics.drawImage` method.

The `draw3DImages` method takes the current value of the disparity offset into account, if the display system is in `MODE_3D_STEREOSCOPIC`.

If the image has not yet been completely loaded, then `drawImage` returns `false`. As more of the image becomes available, the process that draws the image notifies the specified image observer.

Parameters:

`imgLeft` - the specified image to be drawn to the left frame buffer.

`imgRight` - the specified image to be drawn to the right frame buffer.

`x` - the *x* coordinate.

`y` - the *y* coordinate.

`observer` - object to be notified as more of the image is converted.

`bgcolor` - the background color to paint under the non-opaque portions of the image.

Returns:

`true` if the image is completely loaded; `false` otherwise.

Throws:

`java.lang.IllegalStateException` - if the display system is not in a stereoscopic mode

draw3DImages

```
public boolean draw3DImages(java.awt.Image imgLeft,
                            java.awt.Image imgRight,
                            int x,
                            int y,
                            java.awt.image.ImageObserver observer)
                     throws java.lang.IllegalStateException
```

Draws the two images as specified in the imgLeft and imgRight parameters to the corresponding frame buffers. The display system shall ensure that the drawing of both images is synchronized and they appear simultaneously on the screen.

The behavior of this method follows the definition in the corresponding `Graphics.drawImage` method.

The `draw3DImages` method takes the current value of the disparity offset into account, if the display system is in `MODE_3D_STEREOSCOPIC`.

If the image has not yet been completely loaded, then `drawImage` returns `false`. As more of the image becomes available, the process that draws the image notifies the specified image observer.

Parameters:

`imgLeft` - the specified image to be drawn to the left frame buffer.

`imgRight` - the specified image to be drawn to the right frame buffer.

`x` - the *x* coordinate.

`y` - the *y* coordinate.

`observer` - object to be notified as more of the image is converted.

Returns:

`true` if the image is completely loaded; `false` otherwise.

Throws:

`java.lang.IllegalStateException` - if the display system is not in a stereoscopic mode

draw3DImages

```
public boolean draw3DImages(java.awt.Image imgLeft,
                            java.awt.Image imgRight,
                            int x,
                            int y,
                            int width,
                            int height,
                            java.awt.Color bgcolor,
                            java.awt.image.ImageObserver observer)
                     throws java.lang.IllegalStateException
```

Draws the two images as specified in the imgLeft and imgRight parameters to the corresponding frame buffers. The display system shall ensure that the drawing of both images is synchronized and they appear simultaneously on the screen.

This operation is equivalent to filling a rectangle of the width and height of the specified image with the given color and then drawing the image on top of it, but possibly more efficient.

The behavior of this method follows the definition of the corresponding `Graphics.drawImage` method.

The `draw3DImages` method takes the current value of the disparity offset into account, if the display system is in `MODE_3D_STEREOSCOPIC`.

If the image has not yet been completely loaded, then `drawImage` returns `false`. As more of the image becomes available, the process that draws the image notifies the specified image observer.

Parameters:

`imgLeft` - the specified image to be drawn to the left frame buffer.

`imgRight` - the specified image to be drawn to the right frame buffer.

`x` - the *x* coordinate.

`y` - the *y* coordinate.

`width` - the width of the rectangle.

`height` - the height of the rectangle.

`observer` - object to be notified as more of the image is converted.

`bgcolor` - the background color to paint under the non-opaque portions of the image.

Returns:

`true` if the image is completely loaded; `false` otherwise.

Throws:

`java.lang.IllegalStateException` - if the display system is not in a stereoscopic mode

draw3DImages

```
public boolean draw3DImages(java.awt.Image imgLeft,
                            java.awt.Image imgRight,
                            int x,
                            int y,
                            int width,
                            int height,
                            java.awt.image.ImageObserver observer)
                     throws java.lang.IllegalStateException
```

Draws the two images as specified in the imgLeft and imgRight parameters to the corresponding frame buffers. The display system shall ensure that the drawing of both images is synchronized and they appear simultaneously on the screen.

The image is drawn inside the specified rectangle of this graphics context's coordinate space, and is scaled if necessary. Transparent pixels do not affect whatever pixels are already there.

The behavior of this method follows the definition of the corresponding `Graphics.drawImage` method.

The `draw3DImages` method takes the current value of the disparity offset into account, if the display system is in `MODE_3D_STEREOSCOPIC`.

If the image has not yet been completely loaded, then `drawImage` returns `false`. As more of the image becomes available, the process that draws the image notifies the specified image observer.

Parameters:

`imgLeft` - the specified image to be drawn to the left frame buffer.

`imgRight` - the specified image to be drawn to the right frame buffer.

`x` - the *x* coordinate.

`y` - the *y* coordinate.

`width` - the width of the rectangle.

`height` - the height of the rectangle.

`observer` - object to be notified as more of the image is converted.

Returns:

`true` if the image is completely loaded; `false` otherwise.

Throws:

`java.lang.IllegalStateException` - if the display system is not in a stereoscopic mode.

draw3DImages

```
public boolean draw3DImages(java.awt.Image imgLeft,
                            java.awt.Image imgRight,
                            int dx1,
                            int dy1,
                            int dx2,
                            int dy2,
                            int sx1,
                            int sy1,
                            int sx2,
                            int sy2,
                            java.awt.Color bgcolor,
                            java.awt.image.ImageObserver observer)
                     throws java.lang.IllegalStateException
```

Draws the two images as specified in the imgLeft and imgRight parameters to the corresponding frame buffers. The display system shall ensure that the drawing of both images is synchronized and they appear simultaneously on the screen.

Transparent pixels are drawn in the specified background color. This operation is equivalent to filling a rectangle of the width and height of the specified image with the given color and then drawing the image on top of it, but possibly more efficient.

The behavior of this method follows the definition of the corresponding `Graphics.drawImage` method.

The `draw3DImages` method takes the current value of the disparity offset into account, if the display system is in `MODE_3D_STEREOSCOPIC`.

Parameters:

`imgLeft` - the specified image to be drawn to the left frame buffer.

`imgRight` - the specified image to be drawn to the right frame buffer.

`dx1` - the *x* coordinate of the first corner of the destination rectangle.

`dy1` - the *y* coordinate of the first corner of the destination rectangle.

`dx2` - the *x* coordinate of the second corner of the destination rectangle.

`dy2` - the *y* coordinate of the second corner of the destination rectangle.

`sx1` - the *x* coordinate of the first corner of the source rectangle.

`sy1` - the *y* coordinate of the first corner of the source rectangle.

`sx2` - the *x* coordinate of the second corner of the source rectangle.

`sy2` - the *y* coordinate of the second corner of the source rectangle.

`bgcolor` - the background color to paint under the non-opaque portions of the image.

`observer` - object to be notified as more of the image is scaled and converted.

Returns:

`true` if the current output representation is complete; `false` otherwise.

Throws:

`java.lang.IllegalStateException` - if the display system is not in a stereoscopic mode.

draw3DImages

```
public boolean draw3DImages(java.awt.Image imgLeft,
                            java.awt.Image imgRight,
                            int dx1,
                            int dy1,
                            int dx2,
                            int dy2,
                            int sx1,
                            int sy1,
                            int sx2,
                            int sy2,
                            java.awt.image.ImageObserver observer)
                     throws java.lang.IllegalStateException
```

Draws the two images as specified in the imgLeft and imgRight parameters to the corresponding frame buffers. The display system shall ensure that the drawing of both images is synchronized and they appear simultaneously on the screen.

This operation draws as much of the specified area of the specified image as is currently available, scaling it on the fly to fit inside the specified area of the destination drawable surface. Transparent pixels do not affect whatever pixels are already there.

The behavior of this method follows the definition of the corresponding `Graphics.drawImage` method.

The `draw3DImages` method takes the current value of the disparity offset into account, if the display system is in `MODE_3D_STEREOSCOPIC`.

Parameters:

`imgLeft` - the specified image to be drawn to the left frame buffer.

`imgRight` - the specified image to be drawn to the right frame buffer.

`dx1` - the *x* coordinate of the first corner of the destination rectangle.

`dy1` - the *y* coordinate of the first corner of the destination rectangle.

`dx2` - the *x* coordinate of the second corner of the destination rectangle.

`dy2` - the *y* coordinate of the second corner of the destination rectangle.

`sx1` - the *x* coordinate of the first corner of the source rectangle.

`sy1` - the *y* coordinate of the first corner of the source rectangle.

`sx2` - the *x* coordinate of the second corner of the source rectangle.

`sy2` - the *y* coordinate of the second corner of the source rectangle.

`observer` - object to be notified as more of the image is scaled and converted.

Returns:

`true` if the current output representation is complete; `false` otherwise.

Throws:

`java.lang.IllegalStateException` - if the display system is not in a stereoscopic mode

getDisparityOffset

```
public int getDisparityOffset()
```

Returns the current graphics disparity offset in pixels of the Graphics3D instance. If the device is not in a 3D mode, a disparity of 0 is returned.

Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels).

Returns:

the current video disparity offset in pixels.

setDisparityOffset

```
public void setDisparityOffset(int disparity)
                        throws java.lang.IllegalArgumentException,
                               java.lang.IllegalStateException
```

This method allows to set a disparity offset for the Graphics3D instance, which is used for subsequent drawing operations. A positive offset shifts the graphics further away from the viewer, a negative offset brings it closer. A disparity offset of zero places the virtual plane on the screen.

Parameters:

`disparity` - pixel offset to be used for shifting the left and right graphics layers.

Number of pixels is a number of pixels on reference screen (a screen with a horizontal size of 11520 pixels).

Throws:

`java.lang.IllegalArgumentException` - if the disparity value exceeds the system boundaries

`java.lang.IllegalStateException` - if the display system is not in a stereoscopic mode.

6 Service Information

This clause defines the extensions of the service information APIs for planostereoscopic 3D. It describes the handling of DVB Service Information and only applies to DVB broadcast systems using DVB service signalling such as TS 102 727 [2]. For other GEM terminal specifications supporting alternative broadcasting standards, a functionally equivalent API and mapping shall be defined.

In a GEM terminal which is supporting the broadcast target there are two service information APIs:

- JavaTV defines a generic API to handle service information.
  This is defined in the package javax.tv.service.

- A broadcast signalling specific service information API.
  This is defined in the package org.dvb.si for DVB services.

A mapping of the signalling of DVB service information to the service information APIs is defined in the following clauses to accommodate extensions for the signalling of planostereoscopic TV services.

6.1 JavaTV Service Information APIs

A basic mapping of the Java TV Service Information APIs to DVB service information is defined in annex O of MHP TS 102 727 [2]. This mapping is extended in the following clauses.

NOTE: It is expected that a later revision of the MHP specification will include the mapping defined in the present document.

6.1.1 javax.tv.service.ServiceType

The DVB SI service type values are defined in table 87 of EN 300 468 [4].

The following table defines the mapping of the DVB service types to the JavaTV `ServiceType` and extends the mapping as defined in MHP TS 102 727 [2], clause O.2.3. In the case of a conflict the present document has precedence over the mapping defined in MHP.

Table 2: Mapping DVB to Java TV service types

DVB Service type code	DVB Service Type Description	Java TV Service Type
0x01	Digital television service	DIGITAL_TV
0x02	Digital radio sound service	DIGITAL_RADIO
0x03	Teletext service	DATA_BROADCAST
0x04	NVOD Reference service	NVOD_REFERENCE
0x05	NVOD time-shifted service	NVOD_TIME_SHIFTED
0x06	Mosaic service	DIGITAL_TV
0x07	FM Radio service	ANALOG_RADIO
0x0A	Advanced codec digital radio sound service	DIGITAL_RADIO
0x0B	Advanced codec mosaic service	DIGITAL_TV
0x0C	Data broadcast service	DATA_BROADCAST
0x10	DVB MHP service	DATA_APPLICATION
0x11	MPEG-2 HD digital television service	DIGITAL_TV
0x16	Advanced codec SD digital television service	DIGITAL_TV
0x17	Advanced codec SD NVOD time-shifted service	NVOD_TIME_SHIFTED
0x18	Advanced codec SD NVOD reference service	NVOD_REFERENCE
0x19	Advanced codec HD digital television service	
0x1A	Advanced codec HD NVOD time-shifted service	NVOD_TIME_SHIFTED
0x1B	Advanced codec HD NVOD reference service	NVOD_REFERENCE
0x1C	Advanced codec frame compatible plano-stereoscopic HD digital television service	DIGITAL_TV
0x1D	Advanced codec frame compatible plano-stereoscopic HD NVOD time-shifted service	NVOD_TIME_SHIFTED
0x1E	Advanced codec frame compatible plano-stereoscopic HD NVOD reference service	NVOD_REFERENCE
0x00, 0x08, 0x09, 0x0D to 0x0F, 0x12 to 0x15 0x1F to 0xFF		UNKNOWN

6.1.2 javax.tv.service.navigation.StreamType

The DVB SI `stream_content` and `component_type` values are defined in table 26 of EN 300 468 [4].

Table 3 defines a mapping of the DVB stream and component types to the Java TV `StreamType` and extends the mapping as defined in MHP TS 102 727 [2], clause O.2.4. In the case of a conflict the present document has precedence over the mapping defined in MHP.

Table 3: Mapping DVB stream and component types to Java TV

DVB stream_content	DVB component_type	Java TV Stream type
0x01	0x00 to 0xff	VIDEO
0x02	0x00 to 0xff	AUDIO
0x03	0x01, 0x10 to 0x15, 0x20 to 0x25	SUBTITLES
0x03	0x02	DATA
0x03	0x00, 0x16 to 0x1F, 0x26 to 0xFF	UNKNOWN
0x03	0x01 to 0x15, 0x20 to 0x25	UNKNOWN
0x04	0x00 to 0xFF	AUDIO
0x05	0x01, 0x03 to 0x05, 0x07 to 0x08,0x0B to 0x0C,0x80 to 0x83	VIDEO
0x05	0x00,0x02,0x06,0x09 to 0x0A, 0x0D to 0x0E,0x11 to 0x7F,0x84 to 0xFF	UNKNOWN
0x06	0x01, 0x03, 0x05, 0x40 to 0x4A	AUDIO
0x06	0x00 to 0x02, 0x04, 0x06 to 0x3F, 0x4B to 0xFF	UNKNOWN
0x07	0x00 to 0x7F	AUDIO
0x08 to 0xFF	0x00 to 0xFF	UNKNOWN

6.2 DVB Service Information APIs

The DVB SI API, as defined in clause 11.6.1 of MHP TS 102 727 [2] is extended in the following clauses.

6.2.1 Service Type

The following DVB SI service types have been introduced in table 87 of EN 300 468 [4]:

Table 4: Service type coding

Service_type	Description	DVB SI Service Type
0x1C	advanced codec frame compatible plano-stereoscopic HD digital television service	`HD_3D_DIGITAL_TELEVISION`
0x1D	advanced codec frame compatible plano-stereoscopic HD NVOD time-shifted service	`HD_3D_NVOD_TIME_SHIFTED`
0x1E	advanced codec frame compatible plano-stereoscopic HD NVOD reference service	`HD_3D_NVOD_REFERENCE`

The interface `org.dvb.si.SIServiceType` has been extended with appropriate new constants.

6.2.2 Component type

The component type descriptor as defined in clause 6.2.8 of EN 300 468 [4] carries the video format definitions to indicate plano-stereoscopic content.

The new SI component types for these video formats are described in clause 6.2.2 of the 3DTV specification TS 101 547 [3].

There is no specific GEM API to access these component types. Signaling in the content descriptor is considered to be sufficient, since the frame packing arrangement information is always signaled in the video stream.

6.2.3 Content Descriptor

The content descriptor as defined in EN 300 468 [4] carries the signaling information to indicate plano-stereoscopic content.

Plano-stereoscopic services are signaled with Content_nibble_level_1 = 0xB and Content_nibble_level_2 = 0x4 as described in clause 6.2.3 of the 3DTV specification TS 101 547 [3].

An application can query the content descriptor of an `SIEvent` with the method `org.dvb.si.SIEvent.getContentNibbles()`.

6.3 Signaling of 3D Applications

6.3.1 Graphics Constraints Descriptor

Applications using any of the GEM 3D API Packages defined in the present document may be explicitly signalled through the graphical constraints descriptor defined in GEM 1.3 (TS 102 728 [1]), clause 10.4.3.3, by using new values (5 to 7) added into table 20 of TS 102 728 [1].

Table 5: Graphics configuration byte values

Value	Meaning
0	Reserved
1	Full screen standard definition
2	Full screen 960x540 (OCAP)
3	Full screen 1 280x720
4	Full screen 1 920x1 080
5	**Full screen 3D 960x540 (OCAP)**
6	**Full screen 3D 1 280x720**
7	**Full screen 3D 1 920x1 080**
8 to 31	Reserved for future use by DVB project
32 to 255	Reserved for future use

The graphics constraints descriptor is part of the AIT, signalling of this information is not mandatory.

If a terminal does not support a specific application graphics configuration (as signalled in the graphics constraint descriptor) it shall not start the application.

Annex A (informative): Sample Code

A.1 AWT stereoscopic drawing / Disparity change handling

The following sample code illustrates the use of the stereoscopic AWT drawing API. It draws a menu where the selected item is closer to the user and the non-selected items are further behind.

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.KeyEvent;
import java.awt.image.ImageObserver;

import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletContext;
import javax.tv.xlet.XletStateChangeException;

import org.dvb.event.EventManager;
import org.dvb.event.UserEvent;
import org.dvb.event.UserEventListener;
import org.dvb.event.UserEventRepository;
import org.dvb.stereoscopic.graphics.Graphics3D;
import org.dvb.stereoscopic.system.Terminal;
import org.dvb.stereoscopic.ui.HGraphics3DConfigTemplate;
import org.dvb.stereoscopic.ui.HGraphics3DConfiguration;
import org.dvb.stereoscopic.ui.HGraphics3DDevice;
import org.havi.ui.HConfigurationException;
import org.havi.ui.HContainer;
import org.havi.ui.HGraphicsConfigTemplate;
import org.havi.ui.HPermissionDeniedException;
import org.havi.ui.HScene;
import org.havi.ui.HSceneFactory;
import org.havi.ui.HScreen;
import org.havi.ui.event.HRcEvent;

public class AWTStereoscopicDrawingExample implements Xlet, ImageObserver
{
   private static final String IMAGE_BUTTON_FILENAME = "button.jpg";
   private static final String IMAGE_BACKGROUND_FILENAME = "background.jpg";

   private static final String[] MENU_ITEMS = new String[] { "Option 1", "Option 2",
        "Option 3", "Option 4", };

   /** x position of a menu item */
   private static final int MENU_ITEM_X = 100;
   /** y position of the first menu item */
   private static final int MENU_ITEM_Y = 100;
   /** vertical distance between two menu items */
   private static final int MENU_ITEM_Y_OFFSET = 150;

   /** A disparity value where menu will be drawn */
   private static final int MENU_DISPARITY = 0;
   /** Indicates how closer  a menu item will be drawn to a viewer */
   private static final int MENU_ITEM_DISPARITY_OFFSET = 20;
   /** Indicates that focused item will be drawn closer than others */
   private static final int FOCUSED_ITEM_DISPARITY_OFFSET = 10;

   /** Index of focused item in MENU_ITEMS array */
   private int focusedMenuItemIndex = 0;
```

```
private Image imageButton;
private Image imageBackgroud;

HScene hsc;

/** Indicates if xlet is in paused or started state */
private boolean isPaused = true;

private UserEventListener userEventListener;

public void initXlet(XletContext ctx) throws XletStateChangeException {

   /* Logging of some settings */
   Terminal terminal = Terminal.getInstance();

   System.out.println("Display  : 3D capability " +
        terminal.getDisplay().isDisplay3DCapable());
   System.out.println("Terminal : 3D capability " +
        terminal.isStereoscopicCapabilitySupported(Terminal.STEREOSCOPIC_3D));
   System.out.println("Terminal : 3D OpenGL ES capability " +
        terminal.isStereoscopicCapabilitySupported(Terminal.STEREOSCOPIC_3D_OPEN_GL_ES
        ));
   System.out.println("Terminal : 3D video decoding capability " +
        terminal.isStereoscopicCapabilitySupported(Terminal.STEREOSCOPIC_3D_VIDEO_DECO
        DING));

   /* switch the terminal to 3D stereoscopic mode */

   //We create a HGraphics3DConfigTemplate ...
   HGraphics3DConfigTemplate hgdt=new HGraphics3DConfigTemplate();
   // ... and select MODE_3D_STEREOSCOPIC
   hgdt.setPreference(HGraphics3DConfigTemplate.MODE_3D_STEREOSCOPIC,
         HGraphics3DConfigTemplate.REQUIRED);
   HGraphicsConfigTemplate[] hgcta=new HGraphicsConfigTemplate[1];
   hgcta[0]=hgdt;

   HScreen hs=HScreen.getDefaultHScreen();
   // Since we specified a HGraphics3DConfigTemplate we get a
   // HGraphics3DConfiguration instance and we can cast appropriately.
   HGraphics3DConfiguration hgc=(HGraphics3DConfiguration)
        hs.getBestConfiguration(hgcta);
   // We get a HGraphics3DDevice and cast
   HGraphics3DDevice hgd=(HGraphics3DDevice) hs.getDefaultHGraphicsDevice();
   try {
     // Now we can switch to stereoscopic 3D
      hgd.setGraphicsConfiguration(hgc);

      // For the sake of this example - set a disparity offset of the entire graphics
      // device
      hgd.setDisparityOffset(6*20);

      initResources();
      initUserEventListener();
      initScene();

   } catch (SecurityException e) {
      e.printStackTrace();
   } catch (HPermissionDeniedException e) {
      e.printStackTrace();
   } catch (HConfigurationException e) {
      e.printStackTrace();
   }
}

private void initResources() {
   imageButton = Toolkit.getDefaultToolkit().createImage(IMAGE_BUTTON_FILENAME);
   imageButton = Toolkit.getDefaultToolkit().createImage(IMAGE_BACKGROUND_FILENAME);
}
```

```
private void initUserEventListener() {
   UserEventRepository uer = new UserEventRepository("KeyEventListener");
   uer.addKey(HRcEvent.VK_UP);
   uer.addKey(HRcEvent.VK_DOWN);
   userEventListener = new UserEventListener() {
      public void userEventReceived(UserEvent event) {
         if (isPaused) return;
         if (event.getType() != KeyEvent.KEY_PRESSED) return;
         boolean invalidate = false;
         if ( event.getCode() == HRcEvent.VK_UP ) {
            focusedMenuItemIndex = ( focusedMenuItemIndex - 1 + MENU_ITEMS.length ) %
                  MENU_ITEMS.length;
            invalidate = true;
         } else if ( event.getCode() == HRcEvent.VK_DOWN ) {
            focusedMenuItemIndex = ( focusedMenuItemIndex + 1 ) % MENU_ITEMS.length;
            invalidate = true;
         }
         if ( invalidate ) {
            hsc.invalidate();
            hsc.repaint();
         }
      }
   };
   EventManager.getInstance().addUserEventListener(userEventListener, uer);
}

private void initScene() {
   /* Now we get the default HScene */
   hsc = HSceneFactory.getInstance().getDefaultHScene();

   HContainer container = new HContainer(0, 0, hsc.getWidth(), hsc.getHeight()) {
      public void paint(Graphics g) {
         super.paint(g);
         drawMenu((Graphics3D)g);
      }
   };
   container.setBackground(Color.black);
   container.setVisible(true);
   hsc.add(container);
}

private void drawMenu(Graphics3D g3d) {
   /* actual stereoscopic drawing */
   int oldDisparityValue = g3d.getDisparityOffset();
   g3d.setDisparityOffset(MENU_DISPARITY);
   g3d.drawImage(imageBackgroud, 0, 0, this);
   for ( int i = 0; i < MENU_ITEMS.length; i++ ) {
      drawMenuItem(g3d, i);
   }
   g3d.setDisparityOffset(oldDisparityValue);
}

private void drawMenuItem(Graphics3D g3d, int index)  {
   int y = MENU_ITEM_Y + index * MENU_ITEM_Y_OFFSET;

   int oldDisparity = g3d.getDisparityOffset();
   g3d.setDisparityOffset(oldDisparity - MENU_ITEM_DISPARITY_OFFSET);

   // Focused item is being drawn closer
   if ( index == focusedMenuItemIndex ) {
      g3d.setDisparityOffset(g3d.getDisparityOffset() -
            FOCUSED_ITEM_DISPARITY_OFFSET);
   }

   g3d.drawImage(imageButton, MENU_ITEM_X, y, this);

   g3d.setColor(Color.black);
   g3d.drawString(MENU_ITEMS[index], MENU_ITEM_X + 10, y + 10);

   g3d.setDisparityOffset(oldDisparity);
```

```
    }

    public void startXlet() throws XletStateChangeException {
        hsc.setVisible(true);
        isPaused = false;
    }

    public void pauseXlet() {
        hsc.setVisible(false);
        isPaused = true;
    }

    public void destroyXlet(boolean unconditional) throws XletStateChangeException {
        EventManager.getInstance().removeUserEventListener(userEventListener);
    }

    public boolean imageUpdate(Image img, int infoflags, int x, int y, int width, int
         height) {
        return false;
    }
}
```

A.2 Stereoscopic OpenGL

The following sample code illustrates the use of the stereoscopic OpenGL extension. It draws a 3D cube and rotates it along the x- and y-axis.

```
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;

import javax.microedition.khronos.egl.EGL10;
import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.egl.EGLContext;
import javax.microedition.khronos.egl.EGLDisplay;
import javax.microedition.khronos.egl.EGLSurface;
import javax.microedition.khronos.opengles.GL10;
import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletContext;
import javax.tv.xlet.XletStateChangeException;

import org.dvb.stereoscopic.system.Terminal;
import org.dvb.stereoscopic.ui.HGraphics3DConfigTemplate;
import org.dvb.stereoscopic.ui.HGraphics3DConfiguration;
import org.dvb.stereoscopic.ui.HGraphics3DDevice;
import org.havi.ui.HConfigurationException;
import org.havi.ui.HGraphicsConfigTemplate;
import org.havi.ui.HPermissionDeniedException;
import org.havi.ui.HScene;
import org.havi.ui.HSceneFactory;
import org.havi.ui.HScreen;

public class OpenGLExample implements Runnable, Xlet {

  private static final float CAMERA_DISTANCE = 10.0f;
  private static final float ROTATE_ANGEL_PER_SECOND_Y = 90.0f / 1000.0f;
  private static final float ROTATE_ANGEL_PER_SECOND_X = 60.0f / 1000.0f;
  private static final long  FRAME_DISPLAY_TIME_DELTA = 20;

  private XletContext       xletcontext = null;
  private Thread            mainThread;
  private HScene            rootContainer;
  private volatile boolean  alive;
  private boolean           killed      = true;
  private IntBuffer         vertexBuffer;
  private long              initTime;

  public void initXlet(XletContext xletcontext) throws XletStateChangeException {
```

```
    this.xletcontext = xletcontext;
    //We create a HGraphics3DConfigTemplate ...
    HGraphics3DConfigTemplate hgdt=new HGraphics3DConfigTemplate();
    // ... and select MODE_3D_STEREOSCOPIC
    hgdt.setPreference(HGraphics3DConfigTemplate.MODE_3D_OPENGL,
         HGraphics3DConfigTemplate.REQUIRED);
    HGraphicsConfigTemplate[] hgcta=new HGraphicsConfigTemplate[1];
    hgcta[0]=hgdt;

    HScreen hs=HScreen.getDefaultHScreen();
    // Since we specified a HGraphics3DConfigTemplate we get a
    // HGraphics3DConfiguration instance and we can cast appropriately.
    HGraphics3DConfiguration hgc=(HGraphics3DConfiguration)
         hs.getBestConfiguration(hgcta);
    // We get a HGraphics3DDevice and cast
    HGraphics3DDevice hgd=(HGraphics3DDevice) hs.getDefaultHGraphicsDevice();
    try {
      // Now we can switch to stereoscopic OpenGL 3D
      hgd.setGraphicsConfiguration(hgc);
    } catch (SecurityException e) {
      e.printStackTrace();
    } catch (HPermissionDeniedException e) {
      e.printStackTrace();
    } catch (HConfigurationException e) {
      e.printStackTrace();
    }
    System.out.println("Xlet initialized");
  }

  public void startXlet() throws XletStateChangeException {
    System.out.println("xlet started");
    if (mainThread == null) {
      System.out.println("thread created");
      mainThread = new Thread(this);
      System.out.println("thread started");
      mainThread.start();
    }
  }

  public void pauseXlet() {
    System.out.println("Xlet paused");
  }

  public void destroyXlet(boolean unconditional) throws XletStateChangeException {
    System.out.println("Xlet destroing...");
    cleanup();
    System.out.println("Xlet destroyed");
  }

  private void cleanup() {
    alive = false;
    synchronized (this) {
      if (!killed) {
        try {
          wait();
        } catch (InterruptedException e) {
        }
      }
    }
    EGL10 egl = (EGL10) EGLContext.getEGL();
    System.out.println("Terminating EGL...");
    egl.eglTerminate((EGLDisplay) EGL10.EGL_DEFAULT_DISPLAY);
    System.out.println("EGL terminated...");
  }

  public void run() {
    synchronized (this) {
      killed = false;
    }
```

```
System.out.println("Creating HScene...");

rootContainer = HSceneFactory.getInstance().getDefaultHScene();
rootContainer.setBackgroundMode(HScene.NO_BACKGROUND_FILL);
rootContainer.setLayout(null);
rootContainer.setVisible(true);

System.out.println("HScene created.");

EGL10 egl = (EGL10) EGLContext.getEGL();
EGLDisplay eglDisplay = egl.eglGetDisplay(EGL10.EGL_DEFAULT_DISPLAY);
System.out.println("eglDisplay = " + eglDisplay);

int major_minor[] = new int[2];
boolean eglInitStatus = egl.eglInitialize((EGLDisplay) EGL10.EGL_DEFAULT_DISPLAY,
       major_minor);
System.out.println("EGL init status: " + ((eglInitStatus) ? "OK" : "FAILED"));

EGLConfig[] eglConfigs = new EGLConfig[1];
int[] num_config = new int[1];
int attribList[] = {
    EGL10.EGL_RED_SIZE, 8,
    EGL10.EGL_GREEN_SIZE, 8,
    EGL10.EGL_BLUE_SIZE, 8,
    EGL10.EGL_DEPTH_SIZE, 16,
    EGL10.EGL_NATIVE_RENDERABLE, EGL10.EGL_TRUE,
    EGL10.EGL_NONE
};

egl.eglChooseConfig(eglDisplay, attribList, eglConfigs, 1, num_config);

System.out.println("matching configs count = " + num_config[0]);

System.out.println("Create EGLContext...");
EGLContext eglContext = egl.eglCreateContext(eglDisplay, eglConfigs[0],
     EGL10.EGL_NO_CONTEXT, null);
System.out.println("EGLContext created.");

System.out.println("Create EGLSurface...");
EGLSurface eglSurface = egl.eglCreateWindowSurface(eglDisplay, eglConfigs[0],
     rootContainer, null);
System.out.println("EGLSurface created.");

System.out.println("Binding context...");
egl.eglMakeCurrent(eglDisplay, eglSurface, eglSurface, eglContext);
System.out.println("Context bound.");

GL10 gl = (GL10) eglContext.getGL();

gl.glShadeModel(GL10.GL_SMOOTH);
gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
gl.glEnable(GL10.GL_DEPTH_TEST);
gl.glDepthFunc(GL10.GL_LEQUAL);
gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
gl.glShadeModel(GL10.GL_FLAT);
gl.glLineWidth(1.0f);

int vertices[] = {
    // FRONT
    -1, -1, 1, 1, -1, 1,
    -1, 1, 1, 1, 1, 1,
    // BACK
    -1, -1, -1, -1, 1, -1,
    1, -1, -1, 1, 1, -1,
    // LEFT
    -1, -1, 1, -1, 1, 1,
    -1, -1, -1, -1, 1, -1,
    // RIGHT
    1, -1, -1, 1, 1, -1,
    1, -1, 1, 1, 1, 1,
```

```
      // TOP
      -1, 1, 1, 1, 1, 1,
      -1, 1, -1, 1, 1, -1,
      // BOTTOM
      -1, -1, 1, -1, -1, -1,
      1, -1, 1, 1, -1, -1, };
    ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length * 4);
    vbb.order(ByteOrder.nativeOrder());
    vertexBuffer = vbb.asIntBuffer();
    vertexBuffer.put(vertices);
    vertexBuffer.position(0);

  Terminal.getInstance().getDisplay().setOpenGLStereoConvergence(CAMERA_DISTANCE);

  initTime = System.currentTimeMillis();
  long lastTime = initTime;
  long currentTime;

  alive = true;
  while (alive) {
    do {
      currentTime = System.currentTimeMillis();
      if (currentTime >= lastTime + FRAME_DISPLAY_TIME_DELTA) {
        lastTime = currentTime;
        break;
      } else {
        try {
          Thread.sleep(FRAME_DISPLAY_TIME_DELTA + lastTime - currentTime);
        } catch (InterruptedException e) {
        }
      }
    } while (true);

    drawGLScene(gl, currentTime);

    egl.eglSwapBuffers(eglDisplay, eglSurface);

    if (!alive) {
      synchronized (this) {
        egl.eglDestroyContext(eglDisplay, eglContext);
        egl.eglDestroySurface(eglDisplay, eglSurface);
        egl.eglMakeCurrent(eglDisplay, EGL10.EGL_NO_SURFACE, EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_CONTEXT);
        killed = true;
        notify();
      }
      break;
    }
  }
}

private void drawGLScene(GL10 gl, long currentTime) {
  int w = rootContainer.getWidth();
  int h = rootContainer.getHeight();

  gl.glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);

  gl.glViewport(0, 0, w, h);
  gl.glMatrixMode(GL10.GL_PROJECTION);
  gl.glLoadIdentity();
  gl.glFrustumx(-5, 5, -5, 5, 0, 100);

  gl.glMatrixMode(GL10.GL_MODELVIEW);
  gl.glLoadIdentity();
  gl.glTranslatef(0.0f, 0.0f, -CAMERA_DISTANCE);

  long delta = currentTime - initTime;
  gl.glRotatef( delta * ROTATE_ANGEL_PER_SECOND_X, 1.0f, 0.0f, -0.0f);
  gl.glRotatef( delta * ROTATE_ANGEL_PER_SECOND_Y, 0.0f, 1.0f, -0.0f);
```

```
    drawCube(gl);

    gl.glFlush();
  }

  private void drawCube(GL10 gl) {
    gl.glVertexPointer(3, GL10.GL_FIXED, 0, vertexBuffer);
    gl.glColor4f(1, 1, 1, 1);

    // FRONT
    gl.glNormal3f(0, 0, 1);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    // BACK
    gl.glNormal3f(0, 0, -1);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 4, 4);
    // LEFT
    gl.glNormal3f(-1, 0, 0);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 8, 4);
    // RIGHT
    gl.glNormal3f(1, 0, 0);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 12, 4);
    // TOP
    gl.glNormal3f(0, 1, 0);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 16, 4);
    // BOTTOM
    gl.glNormal3f(0, -1, 0);
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 20, 4);
  }
}
```

A.3 Video Controls and Listeners

This example illustrates the use of the new stereoscopic video controls and shows how to use the listeners to be notified about format changes, packaging arrangement changes and disparity changes.

```
import java.io.IOException;

import javax.media.Manager;
import javax.media.NoPlayerException;
import javax.media.Player;
import javax.tv.locator.LocatorFactory;
import javax.tv.locator.MalformedLocatorException;
import javax.tv.xlet.Xlet;
import javax.tv.xlet.XletContext;
import javax.tv.xlet.XletStateChangeException;

import org.davic.media.MediaLocator;
import org.dvb.media.VideoFormatEvent;
import org.dvb.media.VideoFormatListener;
import org.dvb.stereoscopic.media.DisparityChangeEvent;
import org.dvb.stereoscopic.media.PackingArrangementChangeEvent;
import org.dvb.stereoscopic.media.Video3DFormatControl;
import org.dvb.stereoscopic.media.Video3DPresentationControl;
import org.dvb.stereoscopic.media.VideoModeChangeEvent;

public class ListenerExample implements Xlet, VideoFormatListener {

   public void destroyXlet(boolean unconditional)
         throws XletStateChangeException {
   }

   public void initXlet(XletContext ctx) throws XletStateChangeException {
      MediaLocator loc = null;
      try {
               loc = (MediaLocator)
               LocatorFactory.getInstance().createLocator("dvb://123.23.12");
      } catch (MalformedLocatorException e) {
```

```
            e.printStackTrace();
        }

        Player p = null;
        try {
            p = Manager.createPlayer(loc);
        } catch (NoPlayerException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        Video3DFormatControl vfc=(Video3DFormatControl)
            p.getControl("org.dvb.stereoscopic.media.Video3DFormatControl");
        Video3DPresentationControl vpc=(Video3DPresentationControl)
            p.getControl("org.dvb.stereoscopic.media.Video3DPresentationControl");
        vfc.addVideoFormatListener(this);
        // switch to 2D
        vpc.setVideoPresentationMode(Video3DFormatControl.VIDEO_MODE_2D);
        // switch to 3D
        vpc.setVideoPresentationMode(Video3DFormatControl.VIDEO_MODE_3D);
        // select a different disparity offset
        vpc.setVideoDisparityOffset(6*8);
    }

    public void pauseXlet() {
    }

    public void startXlet() throws XletStateChangeException {
    }

    /* This is implementing the VideoFormatEvent interface */
    public void receiveVideoFormatEvent(VideoFormatEvent anEvent) {
        if (anEvent instanceof PackingArrangementChangeEvent) {
            PackingArrangementChangeEvent pe=(PackingArrangementChangeEvent) anEvent;
            System.out.println("Video Packing changed from: "
                +pe.getPreviousPackingArrangement());
            System.out.println("Video Packing changed to: "+pe.getNewPackingArrangement());

        } else if (anEvent instanceof VideoModeChangeEvent) {
            VideoModeChangeEvent me= (VideoModeChangeEvent) anEvent;
            System.out.println("Video Mode changed from: "+me.getPreviousVideoMode());
            System.out.println("Video Mode changed to: "+me.getNewVideoMode());
        } else if (anEvent instanceof DisparityChangeEvent) {
            DisparityChangeEvent de = (DisparityChangeEvent) anEvent;
            System.out.println("Minimum disparity changed to: "+de.getDisparity());
        } else {
            System.out.println("Video format event: "+anEvent);
        }
    }
}
```

Annex B (informative): Bibliography

ETSI TS 101 154: "Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream".".

NOTE: Available at: http://www.etsi.org/deliver/etsi_ts/101100_101199/101154/01.10.01_60/ts_101154v011001p.pdf

Digital Video Broadcasting (DVB); Commercial Requirements for DVB 3D-TV DVB Document A151, July 2010.

NOTE: Available at: http://www.dvb.org/technology/standards/a151_CR_for_DVB-3DTV.pdf

ITU-R Recommendation BT.2160 (11/2009), 'Features of three-dimensional television video systems for broadcasting'.

ETSI TS 101 211: "Digital Video Broadcasting (DVB); Guidelines on implementation and usage of Service Information (SI)".

Siggraph 2011 Stereoscopy Course: 'Stereoscopy, From XY To Z', Presentation Slides.

NOTE: Available at: http://developer.download.nvidia.com/assets/gamedev/docs/Siggraph2011-Stereoscopy_From_XY_to_Z-SG.pdf

History

Document history		
V1.1.1	May 2012	Publication