# OdTrVisRendition interface and GsUpdate Xml protocol description

## Contents

[OdTrVisRendition interface and GsUpdate Xml protocol description 1](#_Toc517260383)

[Contents 1](#_Toc517260384)

[Introduction 7](#_Toc517260385)

[Basic data elements 7](#_Toc517260386)

[OdTrVisRendition interface methods 7](#_Toc517260387)

[onDeviceUpdateBegin/onDeviceUpdateEnd 7](#_Toc517260388)

[C++ 7](#_Toc517260389)

[Xml 7](#_Toc517260390)

[queryRenditionCaps 7](#_Toc517260391)

[C++ 8](#_Toc517260392)

[Xml 8](#_Toc517260393)

[onPartialUpdateInvalidRects 8](#_Toc517260394)

[C++ 8](#_Toc517260395)

[Xml 8](#_Toc517260396)

[onPartialUpdateList 8](#_Toc517260397)

[C++ 8](#_Toc517260398)

[Xml 9](#_Toc517260399)

[querySnapshot 9](#_Toc517260400)

[C++ 9](#_Toc517260401)

[Xml 9](#_Toc517260402)

[createSharingProvider 9](#_Toc517260403)

[C++ 9](#_Toc517260404)

[Xml 9](#_Toc517260405)

[mtDataTypeProcessing 9](#_Toc517260406)

[C++ 10](#_Toc517260407)

[Xml 10](#_Toc517260408)

[idRegistrator 10](#_Toc517260409)

[C++ 10](#_Toc517260410)

[Xml 10](#_Toc517260411)

[queryMetafileStream 10](#_Toc517260412)

[C++ 10](#_Toc517260413)

[Xml 10](#_Toc517260414)

[queryProgramId 10](#_Toc517260415)

[C++ 11](#_Toc517260416)

[Xml 11](#_Toc517260417)

[saveRenditionState 11](#_Toc517260418)

[C++ 11](#_Toc517260419)

[Xml 11](#_Toc517260420)

[loadRenditionState 11](#_Toc517260421)

[C++ 11](#_Toc517260422)

[Xml 11](#_Toc517260423)

[obtainClientSettings 11](#_Toc517260424)

[C++ 11](#_Toc517260425)

[Xml 12](#_Toc517260426)

[onSurfaceChanged 12](#_Toc517260427)

[C++ 12](#_Toc517260428)

[Xml 12](#_Toc517260429)

[onOverlayAdded 13](#_Toc517260430)

[C++ 13](#_Toc517260431)

[Xml 13](#_Toc517260432)

[onOverlayDeleted 14](#_Toc517260433)

[C++ 14](#_Toc517260434)

[Xml 14](#_Toc517260435)

[onOverlayVisibilityChanged 14](#_Toc517260436)

[C++ 14](#_Toc517260437)

[Xml 14](#_Toc517260438)

[onOverlayRenderOrderChanged 14](#_Toc517260439)

[C++ 14](#_Toc517260440)

[Xml 14](#_Toc517260441)

[onViewportAdded 15](#_Toc517260442)

[C++ 15](#_Toc517260443)

[Xml 15](#_Toc517260444)

[onViewportDeleted 15](#_Toc517260445)

[C++ 15](#_Toc517260446)

[Xml 15](#_Toc517260447)

[onViewportInserted 15](#_Toc517260448)

[C++ 15](#_Toc517260449)

[Xml 15](#_Toc517260450)

[onViewportModified 16](#_Toc517260451)

[C++ 16](#_Toc517260452)

[Xml 18](#_Toc517260453)

[onMetafileOrderChanged 21](#_Toc517260454)

[C++ 21](#_Toc517260455)

[Xml 21](#_Toc517260456)

[onMetafileOrderInheritance 22](#_Toc517260457)

[C++ 22](#_Toc517260458)

[Xml 22](#_Toc517260459)

[onLightsListChanged 22](#_Toc517260460)

[C++ 22](#_Toc517260461)

[Xml 23](#_Toc517260462)

[onBackgroundChanged 24](#_Toc517260463)

[C++ 24](#_Toc517260464)

[Xml 24](#_Toc517260465)

[onExtentsChanged 25](#_Toc517260466)

[C++ 25](#_Toc517260467)

[Xml 25](#_Toc517260468)

[onOverlayViewParamsOverride 26](#_Toc517260469)

[C++ 26](#_Toc517260470)

[Xml 26](#_Toc517260471)

[onMetafileAdded 28](#_Toc517260472)

[C++ 28](#_Toc517260473)

[Xml 29](#_Toc517260474)

[onMetafileDeleted 31](#_Toc517260475)

[C++ 31](#_Toc517260476)

[Xml 31](#_Toc517260477)

[onMetafileVisibilityChanged 31](#_Toc517260478)

[C++ 31](#_Toc517260479)

[Xml 31](#_Toc517260480)

[onMetafileHighlightingChanged 31](#_Toc517260481)

[C++ 31](#_Toc517260482)

[Xml 32](#_Toc517260483)

[onMetafileFadingChanged 32](#_Toc517260484)

[C++ 32](#_Toc517260485)

[Xml 32](#_Toc517260486)

[onMetafileHlBranchAttached 32](#_Toc517260487)

[C++ 32](#_Toc517260488)

[Xml 32](#_Toc517260489)

[onMetafileHlBranchDetached 32](#_Toc517260490)

[C++ 32](#_Toc517260491)

[Xml 33](#_Toc517260492)

[onHlBranchAdded 33](#_Toc517260493)

[C++ 33](#_Toc517260494)

[Xml 33](#_Toc517260495)

[onHlBranchModified 33](#_Toc517260496)

[C++ 34](#_Toc517260497)

[Xml 34](#_Toc517260498)

[onHlBranchDeleted 34](#_Toc517260499)

[C++ 34](#_Toc517260500)

[Xml 34](#_Toc517260501)

[onTextureAdded 34](#_Toc517260502)

[C++ 35](#_Toc517260503)

[Xml 35](#_Toc517260504)

[onTextureDeleted 36](#_Toc517260505)

[C++ 36](#_Toc517260506)

[Xml 36](#_Toc517260507)

[onMaterialAdded 36](#_Toc517260508)

[C++ 36](#_Toc517260509)

[Xml 37](#_Toc517260510)

[onMaterialDeleted 38](#_Toc517260511)

[C++ 38](#_Toc517260512)

[Xml 39](#_Toc517260513)

[onVisualStyleAdded 39](#_Toc517260514)

[C++ 39](#_Toc517260515)

[Xml 39](#_Toc517260516)

[onVisualStyleModified 40](#_Toc517260517)

[C++ 40](#_Toc517260518)

[Xml 40](#_Toc517260519)

[onVisualStyleDeleted 41](#_Toc517260520)

[C++ 41](#_Toc517260521)

[Xml 41](#_Toc517260522)

[onLayerAdded 41](#_Toc517260523)

[C++ 42](#_Toc517260524)

[Xml 42](#_Toc517260525)

[onLayerModified 42](#_Toc517260526)

[C++ 42](#_Toc517260527)

[Xml 42](#_Toc517260528)

[onLayerDeleted 43](#_Toc517260529)

[C++ 43](#_Toc517260530)

[Xml 43](#_Toc517260531)

[onLayerViewportPropsOverride 43](#_Toc517260532)

[C++ 43](#_Toc517260533)

[Xml 43](#_Toc517260534)

[Obsolete: onVertexShader 44](#_Toc517260535)

[C++ 44](#_Toc517260536)

[Xml 44](#_Toc517260537)

[Obsolete: onFragmentShader 44](#_Toc517260538)

[C++ 44](#_Toc517260539)

[Xml 45](#_Toc517260540)

[Obsolete: onShaderProgram 45](#_Toc517260541)

[C++ 45](#_Toc517260542)

[Xml 45](#_Toc517260543)

[Metafile graphics stream 47](#_Toc517260544)

[C++ - binary graphics stream representation 47](#_Toc517260545)

[Binary data array types 48](#_Toc517260546)

[Vertex array 48](#_Toc517260547)

[Normals array 48](#_Toc517260548)

[Colors array 48](#_Toc517260549)

[Texture coordinates array 48](#_Toc517260550)

[Depths array 48](#_Toc517260551)

[Indexes array 48](#_Toc517260552)

[Markers array 48](#_Toc517260553)

[0 – Empty record 48](#_Toc517260554)

[1 – EnableOpt record 49](#_Toc517260555)

[2 – DisableOpt record 49](#_Toc517260556)

[3 – Color record 49](#_Toc517260557)

[4 – EnableArray record 49](#_Toc517260558)

[5 – DisableArray record 50](#_Toc517260559)

[6 – DrawArrays record 50](#_Toc517260560)

[7 – DrawElements record 51](#_Toc517260561)

[8 – CullFace record 51](#_Toc517260562)

[9 – LStipple record 51](#_Toc517260563)

[10 – PStipple record 52](#_Toc517260564)

[11 – VPoint record 52](#_Toc517260565)

[12 – VLine record 52](#_Toc517260566)

[13 – IPoint record 52](#_Toc517260567)

[14 – ILine record 53](#_Toc517260568)

[15 – HLRStencil record 53](#_Toc517260569)

[16 – EnableShading record 53](#_Toc517260570)

[17 – DisableShading record 54](#_Toc517260571)

[18 – Material record 54](#_Toc517260572)

[19 – UserEntry record 54](#_Toc517260573)

[20 – InitTexture record 54](#_Toc517260574)

[21 – UninitTexture record 55](#_Toc517260575)

[22 – SelectionMarker record 55](#_Toc517260576)

[23 – EnableMarkerArray record 55](#_Toc517260577)

[24 – DisableMarkerArray record 55](#_Toc517260578)

[25 – SelectionFlags record 55](#_Toc517260579)

[26 – Lineweight record 56](#_Toc517260580)

[27 – Linestyle record 56](#_Toc517260581)

[28 – Program record 56](#_Toc517260582)

[29 – TtfText record 57](#_Toc517260583)

[30 – PushMatrix record 57](#_Toc517260584)

[31 – PopMatrix record 57](#_Toc517260585)

[32 – Metafile record 57](#_Toc517260586)

[33 – GeomMarker record 58](#_Toc517260587)

[34 – VisualStyle record 58](#_Toc517260588)

[35 – MetafileMarker record 58](#_Toc517260589)

[Xml graphics stream representation 59](#_Toc517260590)

[Metafile data array types 60](#_Toc517260591)

[Vertex array 60](#_Toc517260592)

[Normals array 60](#_Toc517260593)

[Colors array 60](#_Toc517260594)

[Texture coordinates array 61](#_Toc517260595)

[Depths array 61](#_Toc517260596)

[Indexes array 61](#_Toc517260597)

[Markers array 61](#_Toc517260598)

[EnableOpt record 62](#_Toc517260599)

[DisableOpt record 62](#_Toc517260600)

[Color record 62](#_Toc517260601)

[EnableArray record 63](#_Toc517260602)

[DisableArray record 63](#_Toc517260603)

[DrawArrays record 63](#_Toc517260604)

[DrawElements record 64](#_Toc517260605)

[CullFace record 64](#_Toc517260606)

[LStipple record 64](#_Toc517260607)

[PStipple record 65](#_Toc517260608)

[VPoint record 65](#_Toc517260609)

[VLine record 65](#_Toc517260610)

[IPoint record 65](#_Toc517260611)

[ILine record 65](#_Toc517260612)

[HLRStencil record 66](#_Toc517260613)

[EnableShading record 66](#_Toc517260614)

[DisableShading record 66](#_Toc517260615)

[Material record 67](#_Toc517260616)

[UserEntry record 67](#_Toc517260617)

[InitTexture record 67](#_Toc517260618)

[UninitTexture record 67](#_Toc517260619)

[SelectionMarker record 67](#_Toc517260620)

[EnableMarkerArray record 68](#_Toc517260621)

[DisableMarkerArray record 68](#_Toc517260622)

[SelectionFlags record 68](#_Toc517260623)

[Lineweight record 68](#_Toc517260624)

[Linestyle record 69](#_Toc517260625)

[Program record 69](#_Toc517260626)

[TtfText record 69](#_Toc517260627)

[PushMatrix record 70](#_Toc517260628)

[PopMatrix record 70](#_Toc517260629)

[Metafile record 71](#_Toc517260630)

[GeomMarker record 71](#_Toc517260631)

[VisualStyle record 71](#_Toc517260632)

[MetafileMarker record 71](#_Toc517260633)

[Remarks 72](#_Toc517260634)

## Introduction

OdTrVisRendition C++ class is a base class for all client rendering modules based onto Teigha Visualizer vectorization device. Client rendering module can override OdTrVisRendition methods to process output data for own needs: render, store into file, convert or export, sent using net and etc. OdTrVisRendition methods are callbacks which will be called if something was changed in graphics scene.

GsUpdate Xml protocol is used by XmlGLES2.txv module based onto Teigha Visualizer vectorization device to convert OdTrVisRendition data into Xml format.

## Basic data elements

Basic elements which manage OdTrVisRendition interface:

* Viewports – represents views on a graphic surface.
* Metafiles – represents graphic streams for render onto graphic surface.
* Highlight branches – represents highlighting states for graphic trees.
* Textures – represents pixel images.
* Materials – represents settings for 3D shading.
* Visual styles – represents rendering settings.
* Layers – represents global settings linked with graphic streams.
* Overlays – represents separate frame buffers onto graphic surface.

## OdTrVisRendition interface methods

### onDeviceUpdateBegin/onDeviceUpdateEnd

**virtual** **void** onDeviceUpdateBegin(OdTrVisRenderClient \*pDevice) = 0;

**virtual** **void** onDeviceUpdateEnd(OdTrVisRenderClient \*pDevice) = 0;

#### C++

This method will be called on each graphic scene updating. onDeviceUpdateBegin always called before any other OdTrVisRendition interface methods called. onDeviceUpdateEnd always called after any other OdTrVisRendition interface methods called. onDeviceUpdateEnd method override is useful for finally render cached graphics onto screen surface.

#### Xml

Example:

<GsUpdate>

</GsUpdate>

This Xml tag always wraps all output graphics data. It is useful to detect that graphics data is started in Xml stream.

### queryRenditionCaps

**virtual** OdUInt32 queryRenditionCaps(OdUInt32 requestFor = 0xFFFFFFFF) **const** = 0;

#### C++

This method is called by vectorizer to check rendition settings and capabilities. queryRenditionCaps method can return following bit flags:

* OdTrVisRendition::kSupportPartialUpdate – set if current OdTrVisRendition support partial update data. Partial update mode is useful for local clients which render graphics directly onto screen.
* OdTrVisRendition::kSupportSnapshotQueries – set if current OdTrVisRendition can return currently rendered frame as RGB image.
* OdTrVisRendition::kSupportCompositeMetafiles – client implementation of OdTrVisRendition can set this flag if it is supports composite metafiles (i. e. single metafile for all render modes).
* OdTrVisRendition::kSupportVisualStyles – client implementation of OdTrVisRendition can take into account Visual Styles during metafiles rendering (if this flag doesn’t set, this is means that client implementation of OdTrVisRendition will take into account render modes only).
* OdTrVisRendition::kSupportGsStateSaving – set if current OdTrVisRendition supports saving/loading their state using OdGsFiler interface.
* OdTrVisRendition::kSupportOverlayBuffers – set if current OdTrVisRendition supports geometry rendering onto separate frame buffers.
* OdTrVisRendition::kUpdateClientSectionOnly – client implementation of OdTrVisRendition interface can set this flag if it requires only update for viewports and related data and not require update for graphics cache (metafiles, textures and etc.).
* OdTrVisRendition::kUpdateShareableSectionOnly – client implementation of OdTrVisRendition interface can set this flag if it requires only update for metafiles and related data and not require update for client-dependent data (viewports and etc.).

#### Xml

Not related to Xml protocol.

### onPartialUpdateInvalidRects

**virtual** **void** onPartialUpdateInvalidRects(OdTrVisOverlayId overlayId, **const** OdGsDCRectArray &invRects) = 0;

#### C++

This method called by vectorizer to setup array of updated screen rectangles per overlay buffer (if supports) if partial update is supported by current OdTrVisRendition.

#### Xml

Not support.

### onPartialUpdateList

**virtual** **void** onPartialUpdateList(OdTrVisViewportId viewportId, OdTrVisOverlayId overlayId, **const** OdTrVisDisplayId \*pList, OdUInt32 nEntries) = 0;

#### C++

This method called by vectorizer to directly set list of metafiles for rendering of current frame. This method will be called for each render frame and for each overlay buffer (if supports) only if partial update is supported by current OdTrVisRendition.

Display list can additionally contain following special codes:

* OdTrVisRendition::kDisplayCodeZero – empty code, can be silently skipped by renderer.
* OdTrVisRendition::kDisplayCodeHighlight – informs renderer that metafiles which will come after this code must be highlighted.
* OdTrVisRendition::kDisplayCodeUnhighlight – informs renderer that highlighting must be disabled for all metafiles which will come after this code.
* OdTrVisRendition::kDisplayCodePushMatrix – informs renderer that it must apply additional transformation matrix for all metafiles which will come after such code. 16 DisplayId’s in a list after this code represents 4x4 matrix elements as double floating point values. Matrices can be nested, so renderer must multiply this matrix with previous matrices if them available. Best practice is to use matrix stack for manage such display codes.
* OdTrVisRendition::kDisplayCodePopMatrix – informs renderer that previously applied transformation matrix must be removed from a matrices stack.
* OdTrVisRendition::kDisplayCodeSelMarkers – sets current highlighting branch directly from a display list. Next DisplayId after this code represents pointer onto OdGsHlBranch data type.
* OdTrVisRendition::kDisplayCodeDrawOrder – provides drawing order for upcoming metafiles. Next DisplayId after this code contains drawing order value as ‘double’ data type.
* OdTrVisRendition::kDisplayCodeMultiOrder – marks that upcoming geometry will contain embedded arrays with drawing order, so separate per-metafile drawing order values can be ignored until next OdTrVisRendition::kDisplayCodeDrawOrder entry.

#### Xml

Not support.

### querySnapshot

**virtual** **void** querySnapshot(OdUInt8 \*pScanlines, **long** minX, **long** minY, **long** nWidth, **long** nHeight) = 0;

#### C++

Vectorizer can call this method if it is require currently rendered image from client OdTrVisRendition. If client OdTrVisRendition supports snapshots it can fill pScanlines array with RGB pixels from requested screen area. RGB scanlines must be aligned to 4 bytes.

#### Xml

Not support.

### createSharingProvider

**virtual** OdTrVisResourceSharingProviderPtr createSharingProvider() = 0;

#### C++

Client implementation of OdTrVisRendition interface must return its own sharing provider in case if it is supports multiple vectorization devices per single database.

#### Xml

Not related to Xml protocol.

### mtDataTypeProcessing

**virtual** MtHandlerType mtDataTypeProcessing(DataType &type) **const** = 0;

#### C++

If client implementations of OdTrVisRendition interface supports multithreaded accessing to graphic resources it is can return how vectorizer must handle multithreading for specific data type.

Client implementation can return following values:

* OdTrVisRendition::kMtSkip – vectorizer must skip specified data type and don’t call related OdTrVisRendition interface methods.
* OdTrVisRendition::kMtRedirect – vectorizer can call related OdTrVisRendition interface methods without any multithreading protection.
* OdTrVisRendition::kMtGlobal – vectorizer must use single global mutex to protect related OdTrVisRendition interface methods calls.
* OdTrVisRendition::kMtCompete – vectorizer must use mutex for specified data type to protect related OdTrVisRendition interface methods calls.
* OdTrVisRendition::kMtRecord – vectorizer must record OdTrVisRendition interface methods calls until multithreading vectorization isn’t completed and play recorded calls into OdTrVisRendition interface after that.

#### Xml

Not related to Xml protocol.

### idRegistrator

**virtual** IdRegistrator \*idRegistrator() **const** = 0;

#### C++

Client implementation of OdTrVisRendition interface can return its own unique identifiers registration handler if it is require unique identifiers.

#### Xml

Not related to Xml protocol.

### queryMetafileStream

**virtual** **bool** queryMetafileStream(OdTrVisDisplayId displayId, **const** OdTrVisFlatMetafileContainer \*\*pStream) **const** = 0;

#### C++

Vectorizer has ability to optimize selection process using metafile stream which is already generated and sent to client. To check that client supports returning of direct metafile stream pointer firstly vectorizer call this method with null arguments and in case if client returns true vectorizer can invoke such method in selection process.

#### Xml

Not related to Xml protocol.

### queryProgramId

**virtual** OdTrVisProgramId queryProgramId(OdGsView::RenderMode mode, OdTrVisVisualStyleId vsId = kTrVisNegativeId, OdTrVisGeomType geomType = OdTrVisGeomType\_Default, OdUInt32 shadingFlags = 0) **const** = 0;

#### C++

Vectorizer has ability to put unique shading program identifiers directly into metafile streams. If renderer supports shading programs and ability to process them it can return their own data identifiers using this override. In case if rendition doesn’t support or not require shading program identifiers this method will return kTrVisNegativeId.

#### Xml

Not related to Xml protocol.

### saveRenditionState

**virtual** **bool** saveRenditionState(OdGsFiler \*pFiler) **const** = 0;

#### C++

This method can be called during saving of Graphic System cache in case if OdTrVisRendition::kSupportGsStateSaving flag specified by OdTrVisRendition in “queryRenditionCaps” call.

#### Xml

Not related to Xml protocol.

### loadRenditionState

**virtual** **bool** loadRenditionState(OdGsFiler \*pFiler, OdTrVisIdMap \*pIdMap) = 0;

#### C++

This method can be called during loading of Graphic System cache in case if OdTrVisRendition::kSupportGsStateSaving flag specified by OdTrVisRendition in “queryRenditionCaps” call.

#### Xml

Not related to Xml protocol.

### obtainClientSettings

**virtual** **void** obtainClientSettings(ClientSettings &clientSets) = 0;

#### C++

Client implementation of OdTrVisRendition interface can return settings which vectorizer will take into account during output data generation.

Client information (text strings):

* m\_ciInfo.m\_glVendor – vendor of OpenGL driver.
* m\_ciInfo.m\_glRenderer – OpenGL renderer name.
* m\_ciInfo.m\_glVersion – OpenGL version.
* m\_ciInfo.m\_glSLVersion – GLSL version.
* m\_ciInfo.m\_glExtensions – supported OpenGL extensions list.

Flags:

* kNonPow2Textures – set to true if client support texture dimensions not scaled to power of 2.
* kSupportsBGR – set to true if client support textures in BGR format.
* kSupportsBGRA – set to true if client support textures in BGRA format.
* kRequireBGR – set to true if client require textures only in BGR/BGRA format.
* kOverrideLut – set to true if client override vectorizer settings related to handling of raster images with palette (1-8 bits per pixel).
* kLutMonochrome – set to true if client can process monochrome textures in luminance format (1 byte per pixel); set to false if client require monochrome textures in RGB/RGBA or BGR/BGRA format. This setting will be used by vectorizer only if kOverrideLut flag is set to true.
* kLutPalette - set to true if client can process 4-8 bits per pixel textures in luminance format (1 byte per pixel); set to false if client require 4-8 bits per pixel textures in RGB/RGBA or BGR/BGRA format. This setting will be used by vectorizer only if kOverrideLut flag is set to true.
* kShadeByVertex – set to true if client haven’t enough resources to process per-pixel lighting in shaded modes (actual for mobile platforms), in this case per-vertex lighting will be enabled for related shading programs.

Limits:

* m\_nMaxTextureSize [OdUInt32] – specifies maximum processable texture dimension.
* m\_nTextureAlignment [OdUInt32] – specifies scanlines alignment for generated textures.
* m\_nMaxLineWidth [float] – specifies maximum processable line width in pixels.
* m\_nMaxPointSize [float] – specifies maximum processable point size in pixels.
* m\_nMaxFragShadUniformVecs [OdUInt32] – specifies maximum processable uniform vectors in fragment shaders.

#### Xml

Not related to GsUpdate Xml protocol.

### onSurfaceChanged

**virtual** **void** onSurfaceChanged(**const** OdTrVisSurfaceDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if rendering surface configuration changed.

OdTrVisSurfaceDef member’s description:

* m\_surfaceSize [OdGsDCRect] – dimensions of output surface.
* m\_backgroundColor [ODCOLORREF] – background color for rendering surface.
* m\_surfaceFlags [OdUInt32] – set of bit flags which describe additional surface properties:
* OdTrVisSurfaceDef::kTransparentBackground – surface contains transparent background; disable color and depth buffer filling for this surface.

#### Xml

Example:

<SurfaceChanged>

<SurfaceSizeXMin>0</SurfaceSizeXMin>

<SurfaceSizeYMin>620</SurfaceSizeYMin>

<SurfaceSizeXMax>1643</SurfaceSizeXMax>

<SurfaceSizeYMax>0</SurfaceSizeYMax>

<BackgroundColor>0, 0, 0, 255</BackgroundColor>

<TransparentBackground>1</TransparentBackground>

</SurfaceChanged>

* SurfaceSizeXMin [integer] – minimal horizontal coordinate of rendering surface (typically zero).
* SurfaceSizeYMin [integer] – minimal vertical coordinate of rendering surface.
* SurfaceSizeXMax [integer] – maximal horizontal coordinate of rendering surface.
* SurfaceSizeYMax [integer] – maximal vertical coordinate of rendering surface (typically zero).
* BackgroundColor [integer \* 4] – set of four color components (R, G, B, A) which specifies background rendering surface color in 0-255 range.
* TransparentBackground [boolean] – marks surface in case if it contains transparent background; disable color and depth buffer filling for this surface.

### onOverlayAdded

**virtual** **void** onOverlayAdded(OdTrVisOverlayId overlayId, **const** OdTrVisOverlayDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if new overlay buffer is added into graphics scene. Default main scene overlay (kTrVisMainOverlayId with value 0) is always present into graphics scene, so for it onOverlayAdded method will not be called.

OdTrVisOverlayDef member’s description:

* m\_overlayFlags [OdUInt16] – set of bit flags which describe additional surface properties:
* OdTrVisOverlayDef::kOBEnabled – overlay buffer is visible.
* OdTrVisOverlayDef::kOBDepthBuffer – depth buffer enabled for this overlay.
* OdTrVisOverlayDef::kOBMergeDepth – overlay depth buffer can be merged with primary graphics scene depth buffer.
* OdTrVisOverlayDef::kOBDirectRender – overlay buffer can be rendered directly onto screen.
* OdTrVisOverlayDef::kOBHighlight – geometry will be rendered onto this overlay buffer with enabled highlighting style.
* OdTrVisOverlayDef::kOBContrast – geometry will be rendered onto this overlay buffer with enabled contrast style.
* m\_renderOrder [OdInt16] – signed integer value which represent relative rendering order of overlay buffer into overlay buffers list.

#### Xml

Example:

<OverlayAdded>

<OverlayID>2</OverlayID>

<Flags>3</Flags>

<RenderOrder>1</RenderOrder>

</OverlayAdded>

* OverlayID [string] – identifier of newly created overlay buffer.
* Flags [integer] – set of bit flags represent generic overlay properties:
* 1 – overlay buffer is visible.
* 2 – depth buffer enabled for this overlay buffer.
* 3 – overlay depth buffer can be merged with primary graphics scene depth buffer.
* 4 – overlay buffer can be rendered directly onto screen.
* 5 – geometry will be rendered onto this overlay buffer with enabled highlighting style.
* 6 – geometry will be rendered onto this overlay buffer with enabled contrast style.
* RenderOrder [integer] – signed integer value which represent relative rendering order of overlay buffer into overlay buffers list.

### onOverlayDeleted

**virtual** **void** onOverlayDeleted(OdTrVisOverlayId overlayId) = 0;

#### C++

This method will be called by vectorizer in case if exist overlay buffer is deleted from graphics scene.

#### Xml

Example:

<OverlayDeleted>

<OverlayID>3</OverlayID>

</OverlayDeleted>

* OverlayID [string] – identifier of overlay buffer to be deleted.

### onOverlayVisibilityChanged

**virtual** **void** onOverlayVisibilityChanged(OdTrVisOverlayId overlayId, **bool** bVisibility) = 0;

#### C++

This method will be called by vectorizer in case if overlay buffer visibility state is to be changed.

#### Xml

Example:

<OverlayVisibilityChanged>

<OverlayID>3</OverlayID>

<Visibility>0</Visibility>

</OverlayVisibilityChanged>

* OverlayID [string] – identifier of overlay buffer to change visibility state.
* Visibility [boolean] – new visibility state.

### onOverlayRenderOrderChanged

**virtual** **void** onOverlayRenderOrderChanged(OdTrVisOverlayId overlayId, OdInt16 nRenderOrder) = 0;

#### C++

This method will be called by vectorizer in case if overlay buffer relative rendering order is to be changed.

#### Xml

Example:

<OverlayRenderOrderChanged>

<OverlayID>3</OverlayID>

<RenderOrder>2</RenderOrder>

</OverlayRenderOrderChanged>

* OverlayID [string] – identifier of overlay buffer to change rendering order.
* RenderOrder [integer] – signed integer value which represent new relative rendering order of overlay buffer into overlay buffers list.

### onViewportAdded

**virtual** **void** onViewportAdded(OdTrVisViewportId viewportId) = 0;

#### C++

This method will be called by vectorizer in case if new viewport is added into graphics scene.

#### Xml

Example:

<ViewportAdded>

<ViewportID>145425480</ViewportID>

</ViewportAdded>

* ViewportID [string] – identifier of newly created viewport.

### onViewportDeleted

**virtual** **void** onViewportDeleted(OdTrVisViewportId viewportId) = 0;

#### C++

This method will be called by vectorizer in case if exist viewport is deleted from graphics scene.

#### Xml

Example:

<ViewportDeleted>

<ViewportID>145425480</ViewportID>

</ViewportDeleted>

* ViewportID [string] – identifier of viewport to be deleted.

### onViewportInserted

**virtual** **void** onViewportInserted(OdTrVisViewportId viewportId, **int** nIndex) = 0;

#### C++

This method will be called by vectorizer in case if new viewport is added into graphics scene at specified position in viewports list.

#### Xml

Example:

<ViewportInserted>

<ViewportID>145425480</ViewportID>

<Index>1</Index>

</ViewportInserted>

* ViewportID [string] – identifier of newly created viewport.
* Index [integer] – position at which newly created viewport must be added into viewports list.

### onViewportModified

**virtual** **void** onViewportModified(OdTrVisViewportId viewportId, **const** OdTrVisViewportDef &pDef, OdUInt32 kindOfMod = kViewportModAll) = 0;

#### C++

This method will be called by vectorizer in case if any of viewport settings is changed. Viewport data in OdTrVisViewportDef is always completely actual, but kindOfMode argument contains bit flags that indicate which fields of OdTrVisViewportDef structure is really modified during last update:

* OdTrVisRendition::kViewportModVisibility – viewport visibility field modified.
* OdTrVisRendition::kViewportModOrientation – any of viewport positioning-related, matrix, clipping or projection fields is modified.
* OdTrVisRendition::kViewportModPersistent – viewport persistent data fields modified (flags or so on, typically them change only during first onViewportModified call).
* OdTrVisRendition::kViewportModContextual – database context settings related to graphics rendered into this viewport (lines rendering properties, graphics fading amounts and etc.).
* OdTrVisRendition::kViewportModRect – viewport rectangle on output surface is modified.
* OdTrVisRendition::kViewportModNrcClip – viewport non-rectangular clipping boundary modified.
* OdTrVisRendition::kViewportModLineweights – set of viewport lineweights or lineweights coefficient is modified.
* OdTrVisRendition::kViewportModShader – basic shading program or rendering flags is modified.
* OdTrVisRendition::kViewportModBorder – viewport border geometry, color or lineweight is modified.

OdTrVisViewportDef member’s description:

* m\_bVisible [bool] – specifies visibility state of this viewport.
* m\_viewParams [OdTrVisViewParamsDef] – specifies view orientation and transformation parameters. OdTrVisViewParamsDef member’s description:
  + m\_screenMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform coordinates from normalized device space to screen space.
  + m\_viewingMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform coordinates from world space to view space.
  + m\_projectionMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform from view space to normalized device space.
  + m\_correctionMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform from screen space to OpenGL viewport space.
  + m\_outputMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform coordinates from metafile space to view space.
  + m\_metafileMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform coordinates from world space to metafile space.
  + m\_viewPosition [OdGePoint3d] – specifies viewer origin in world space.
  + m\_viewBasis [OdGeVector3d[3]] – specifies viewer X, Y and Z axes in world space.
  + m\_bPerspectiveProjection [bool] – set to true if perspective projection is enabled, elsewhere orthogonal projection will be used.
  + m\_fieldWidth [double] – specifies camera field width.
  + m\_fieldHeight [double] – specifies camera field height.
  + m\_lensLength [double] – specifies camera lens length for perspective projection.
  + m\_viewTarget [OdGePoint3d] – specifies viewer target point in world space.
  + m\_frontClip [OdTrVisViewClipped] – specifies setting of view front clipping plane.
  + m\_backClip [OdTrVisViewClipped] – specifies setting of view back clipping plane.
* m\_vptFlags [OdUInt16] – set of bit flags which describe viewport type:
* OdTrVisViewportDef::kPSOverallViewport – viewport is an overall paper space viewport.
* OdTrVisViewportDef::kPSModelViewport – viewport is a paper space viewport which shows model space contents.
* OdTrVisViewportDef::kPSHelperViewport – viewport is a helper paper space viewport.
* m\_overallViewportId [OdTrVisViewportId] – identifier of overall paper space viewport in case if this is paper space viewport which shows model space contents.
* m\_lineStyleConfig [ODCOLORREF] – configuration of lineweights default style encoded as set of color components (red – display lineweight for points; green – default lineweight caps style (0 – butt, 1 – square, 2 – round, 3 – diamond); blue – default lineweight joins style (0 – miter, 1 – bevel, 2 – round, 3 – diamond); alpha – set to 1 if lineweight settings is default settings for renderer).
* m\_fadingConfig [ODCOLORREF] – set of fading amounts (in 0-255 range, where maximal value indicates that fading is completely disabled) which can be invoked during geometry rendering (red color component – specifies fading amount for geometry located onto locked layer; green color component – specifies fading amount for metafiles which represents external references geometry; blue color component – specifies fading amount for metafiles which is marked as faded; alpha color component – reserved and should be always set to zero).
* m\_screenRect [OdGsDCRectDouble] – specifies viewport rectangle on output surface.
* m\_nrcCounts [OdIntArray] – in case if non-rectangular viewport clipping enabled this array contains number of vertices for each clipping boundary loop.
* m\_nrcPoints [OdFloatArray] – array of float’s which represents non-rectangular viewport clipping vertices. Pair of float’s per each vertex.
* m\_lineweightsEnum [OdUInt8Array] – array of lineweights in pixel for lineweight indexes in model space.
* m\_lineweightsCoef [double] – lineweights multiplier.
* m\_rmShader [OdTrVisProgramId] – identifier of viewport basic shader program.
* m\_visualStyle [OdTrVisVisualStyleId] – identifier of assigned viewport visual style.
* m\_bDepthBuffer [bool] – set to true if depth buffer must be enabled for this viewport.
* m\_bStencilBuffer [bool] – set to true if stencil buffer must be enabled for this viewport.
* m\_bPolygonOffset [bool] – set to true if polygon offset must be enabled for this viewport.
* m\_bLighting [bool] – set to true if lighting must be enabled for this viewport.
* m\_bFaceNormals [bool] – set to true if lighting must use face normals instead of vertex normals.
* m\_faceFillColor [ODCOLORREF] – specifies face filling color for Hidden Line render mode.
* m\_bBorder [bool] – set to true if viewport border must be rendered.
* m\_borderColor [ODCOLORREF] – specifies viewport border color.
* m\_borderWidth [int] – specified viewport border lines width in pixels.
* m\_boderCounts [OdIntArray] - in case if viewport border enabled this array contains number of vertices for each border loop.
* m\_borderPoints [OdFloatArray] – array of float’s which represents viewport border vertices. Pair of float’s per each vertex.

#### Xml

Example:

<ViewportModified>

<ViewportID>145425480</ViewportID>

<Visibility>1</Visibility>

<ViewParams>

<ScreenMatrix>1643, 0, 0, 0, 0, -620, 0, 620, 0, 0, 1, 0, 0, 0, 0, 1</ScreenMatrix>

<ViewingMatrix>1, -0, -0, -57.791, -0, 0, 1, -81.423, 0, -1, 0, 109.087, 0, 0, 0, 1</ViewingMatrix>

<ProjectionMatrix>0.000999458, 0, -0.00039274, 0.5, 0, 0.00264856, -0.00039274, 0.5, 0, 0, -1.00079, 1, 0, 0, -0.00078548, 1</ProjectionMatrix>

<CorrectionMatrix>2, 0, 0, -1, 0, 2, 0, -1, 0, 0, 2, -1, 0, 0, 0, 1</CorrectionMatrix>

<OutputMatrix>1, -0, -0, -57.791, -0, 0, 1, -81.423, 0, -1, 0, 109.087, 0, 0, 0, 1</OutputMatrix>

<MetafileMatrix>1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1</MetafileMatrix>

<ViewPosition>57.791, -1021.1, 81.423</ViewPosition>

<ViewBasisX>1, 0, -0</ViewBasisX>

<ViewBasisY>0, 0, 1</ViewBasisY>

<ViewBasisZ>0, -1, 0</ViewBasisZ>

<PerspectiveProjection>1</PerspectiveProjection>

<FieldWidth>871.05</FieldWidth>

<FieldHeight>377.563</FieldHeight>

<LensLength>50</LensLength>

<ViewTarget>57.791, 109.087, 81.423</ViewTarget>

<FrontClip>1</FrontClip>

<FrontClipDist>1.5</FrontClipDist>

<BackClip>1</BackClip>

<BackClipDist>-1.5</BackClipDist>

</ViewParams>

<PSModelViewport>1</PSModelViewport>

<OverallVpId>143255432</OverallVpId>

<PSOverallViewport>0</PSOverallViewport>

<PSHelperViewport>0</PSHelperViewport>

<LineStyleConfig>1, 2, 2, 1</LineStyleConfig>

<FadingConfig>128, 128, 128, 0</FadingConfig>

<ScreenRect>0, 1, 1, 0</ScreenRect>

<NNrcCounts>2</NNrcCounts>

<NNrcPoints>2</NNrcPoints>

<NrcCounts>

<Count>1</Count>

<Count>1</Count>

</NrcCounts>

<NrcPoints>

<Point>1.1, 2.2</Point>

<Point>3.3, 4.4</Point>

</NrcPoints>

<NLineweightEnum>2</NLineweightEnum>

<LineweightCoef>0</LineweightCoef>

<LineweightEnum>

<NPixels>1<NPixels>

<NPixels>2<NPixels>

</LineweightEnum>

<ShaderId>4</ShaderId>

<VisualStyleId>54266422</VisualStyleId>

<DepthBuffer>1</DepthBuffer>

<StencilBuffer>0</StencilBuffer>

<PolygonOffset>1</PolygonOffset>

<Lighting>1</Lighting>

<FaceNormals>0</FaceNormals>

<FaceFillColor>0, 0, 0, 255</FaceFillColor>

<Border>1</Border>

<BorderColor>255, 255, 255, 255</BorderColor>

<BorderWidth>1</BorderWidth>

<NBorderCounts>2</NBorderCounts>

<NBorderPoints>2</NBorderPoints>

<BorderCounts>

<Count>1</Count>

<Count>1</Count>

</BorderCounts>

<BorderPoints>

<Point>1.1, 2.2</Point>

<Point>3.3, 4.4</Point>

</BorderPoints>

</ViewportModified>

*Note*: Depends from viewport modification flags any of fields could be unavailable. During viewport creation on Xml client side all fields should be initialized by defaults.

* ViewportID [string] – identifier of modified viewport.
* KindOfMod [integer] – kind of modification bit flags.
* Visibility [boolean] – set to 1 if viewport is visible.
* ViewParams [tag] – represents block with view orientation and transformation parameters:
  + ScreenMatrix [float \* 16] – matrix for transform coordinates from normalized device space to screen space.
  + ViewingMatrix [float \* 16] – matrix for transform coordinates from world space to view space.
  + ProjectionMatrix [float \* 16] – matrix for transform coordinates from view space to normalized device space.
  + CorrectionMatrix [float \* 16] – matrix for transform coordinates from screen space to OpenGL viewport space.
  + OutputMatrix [float \* 16] – matrix for transform coordinates from metafile space to view space.
  + MetafileMatrix [float \* 16] – matrix for transform coordinates from world space to metafile space.
  + ViewPosition [float \* 3] – camera position in world space.
  + ViewBasisX [float \* 3] – camera X-axis in world space.
  + ViewBasisY [float \* 3] – camera Y-axis in world space.
  + ViewBasisZ [float \* 3] – camera Z-axis in world space.
  + PerspectiveProjection [boolean] – set to 1 if perspective projection is enabled.
  + FieldWidth [float] – camera field width.
  + FieldHeight [float] – camera field height.
  + LensLength [float] – camera lens length for perspective projection.
  + ViewTarget [float \* 3] – camera target position in world space.
  + FrontClip [boolean] – set to 1 if front view clipping is enabled.
  + FrontClipDist [float] – front view clipping distance.
  + BackClip [boolean] – set to 1 if back view clipping is enabled.
  + BackClipDist [float] – back view clipping distance.
* PSModelViewport [boolean] – set to 1 if this is paper space viewport which shows model space contents.
* OverallVpId [string] – identifier of overall paper space viewport in case if this is paper space viewport which shows model space contents.
* PSOverallViewport [boolean] – set to 1 if this is overall paper space viewport.
* PSHelperViewport [boolean] – set to 1 if this is helper paper space viewport.
* LineStyleConfig [integer \* 4] – configuration of lineweights default style encoded as 4 components (1 – display lineweight for points; 2 – default lineweight caps style (0 – butt, 1 – square, 2 – round, 3 – diamond); 3 – default lineweight joins style (0 – miter, 1 – bevel, 2 – round, 3 – diamond); 4 – set to 1 if lineweight settings is default settings for renderer).
* FadingConfig [integer \* 4] – set of fading amounts (in 0-255 range, where maximal value indicates that fading is completely disabled) which can be invoked during geometry rendering (red color component – specifies fading amount for geometry located onto locked layer; green color component – specifies fading amount for metafiles which represents external references geometry; blue color component – specifies fading amount for metafiles which is marked as faded; alpha color component – reserved and should be always set to zero).
* ScreenRect [float \* 4] – viewport rectangle on output surface.
* NNrcCounts [integer] – in case if non-rectangular viewport clipping is enabled this value specifies number of loops in clipping boundary.
* NNrcPoints [integer] – in case if non-rectangular viewport clipping is enabled this value specifies number of vertexes in clipping boundary.
* NrcCounts [tag] – set of <Count> tags [integer] each of which specifies number of vertexes in a clipping boundary loop.
* NrcPoints [tag] – set of <Point> tags [float \* 2] each of which specifies vertex in a clipping boundary.
* NLineweightEnum [integer] – in case if array of lineweights is available this value specifies number of elements in lineweights array.
* LineweightCoef [float] – lineweights multiplier.
* LineweightEnum [tag] – set of <NPixels> tags [integer] each of which specifies lineweight in pixels.
* ShaderId [string] – identifier of basic shader for this viewport.
* VisualStyleId [string] – identifier of assigned viewport visual style.
* DepthBuffer [boolean] – specifies does depth buffer must be enabled during this viewport rendering or not.
* StencilBuffer [boolean] – specifies does stencil buffer must be enabled during this viewport rendering or not.
* PolygonOffset [boolean] – specifies does polygon offset must be enabled during this viewport rendering or not.
* Lighting [boolean] – specifies does lighting must be enabled during this viewport rendering or not.
* FaceNormals [boolean] – specifies does lighting must use face normals instead of vertex normals.
* FaceFillColor [integer \* 4] – specifies faces filling color for Hidden Line render mode as four R, G, B and A integers (0-255 range).
* Border [boolean] – set to 1 in case if viewport border must be rendered.
* BorderColor [integer \* 4] – specifies viewport border color as four R, G, B and A (0-255 range) integers.
* BorderWidth [integer] – specifies viewport border line width in pixels.
* NBorderCounts [integer] – in case if viewport border is enabled this value specifies number of loops in viewport border.
* NBorderPoints [integer] – in case if viewport border is enabled this value specifies number of vertexes in viewport border.
* BorderCounts [tag] – set of <Count> tags [integer] each of which specifies number of vertexes in a viewport border loop.
* BorderPoints [tag] – set of <Point> tags [float \* 2] each of which specifies vertex in a viewport border.

### onMetafileOrderChanged

**virtual** **void** onMetafileOrderChanged(OdTrVisViewportId viewportId, OdTrVisOverlayId overlayId, **const** OdTrVisDisplayId \*pList, OdUInt32 nEntries) = 0;

**virtual** **void** onMetafileOrderChanged(OdTrVisViewportId viewportId, OdTrVisOverlayId overlayId, OdUInt32 nInvalidFrom, OdUInt32 nInvalidLen, **const** OdTrVisDisplayId \*pValidFrom, OdUInt32 nValidLen) = 0;

#### C++

These methods will be called by vectorizer in case if metafiles rendering list is changed. If metafiles rendering list is updated both methods will be called, but client rendition implementation require to override only one of them (depends from rendition needs). First method is useful for local renderers which doesn’t store metafiles list on renderer side and directly render metafiles from pointer which is sent from vectorizer. Second method is useful for renderers which store metafiles list on renderer side; vectorizer sent to this method pointer onto list part which is updated on a last update which provides way for a renderer to substitute invalid part of list with valid data.

#### Xml

Example:

<MetafileOrderChanged>

<ViewportID>145425480</ViewportID>

<OverlayID>0</OverlayID>

<InvalidFrom>0</InvalidFrom>

<InvalidLen>0</InvalidLen>

<ValidLen>8</ValidLen>

<Ordering>

<MetafileID>51778696</MetafileID>

<MetafileID>51763776</MetafileID>

<MetafileID>51614288</MetafileID>

<MetafileID>145564568</MetafileID>

<MetafileID>145990928</MetafileID>

<MetafileID>145815672</MetafileID>

<MetafileID>145590088</MetafileID>

<MetafileID>145555960</MetafileID>

</Ordering>

</MetafileOrderChanged>

*Note*: Metafiles list could contain null identifiers; these identifiers must be silently skipped by renderer during rendering. Null identifiers are used by vectorizer to temporarily reserve place in a metafiles list; them will be substituted by actual metafile identifiers on next metafiles list updates.

* ViewportID [string] – identifier of viewport to modify metafiles list.
* OverlayID [string] – identifier of overlay buffer contains viewport to modify metafiles list.
* InvalidFrom [integer] – index of first invalid entry in a list to be updated.
* InvalidLen [integer] – number of invalid entries in a list to be removed (could be null in case if list doesn’t contain invalid entries to be removed).
* ValidLen [integer] – number of valid entries to be added into list (could be null in case if list updated to remove entries only).
* Ordering [tag] – set of <MetafileID> tags [string] each of which specifies metafile identifier to be added in metafiles list.

### onMetafileOrderInheritance

**virtual** **void** onMetafileOrderInheritance(OdTrVisViewportId viewportId1, OdTrVisOverlayId overlayId, OdTrVisViewportId viewportId2) = 0;

#### C++

This method will be called by vectorizer in case if metafiles rendering list may be shared between different viewports instead of creation rendering lists for each viewport. However vectorizer will still sent **onMetafileOrderChanged** for both viewports. So, if rendition supports rendering lists sharing it has to apply **onMetafileOrderChanged** changes for one viewport and ignore changes for other.

**viewportId2** may be invalid id: in this case rendition should break sharing rendering lists for **viewportId1**.

#### Xml

Example:

<MetafileOrderInheritance>

<ViewportID1>20F</ViewportID1>

<OverlayID>0</OverlayID>

<ViewportID2>204</ViewportID2>

</MetafileOrderInheritance>

* ViewportID1 [string] – identifier of viewport that can use foreign rendering list
* OverlayID [string] – identifier of overlay buffer contains viewport that can use foreign rendering list and viewport whose rendering list can be used as foreign
* ViewportID2 [string] – identifier of viewport whose rendering list can be used as foreign

### onLightsListChanged

**virtual** **void** onLightsListChanged(OdTrVisViewportId viewportId, **float** ambientLight[4], **bool** bDefaultLightsOn, **const** OdTrVisLightDef \*pList, OdUInt32 nEntries) = 0;

#### C++

This method will be called by vectorizer in case if lights list is changed for shaded rendering modes. Additionally vectorizer sent ambient light color which could be added by renderer into shading program for material rendering. bDefaultLightsOn argument informs renderer that default light sources must be taken into account during visualization or not.

OdTrVisLightDef member’s description:

* m\_type [enum] – type of light:
* OdTrVisLightDef::kDistant – emit parallel light rays without attenuation.
* OdTrVisLightDef::kPoint – emit light in all directions.
* OdTrVisLightDef::kSpot – emit light in a specified cone.
* m\_bDefLight [bool] – true in case if this light represents light generated by default.
* m\_position [OdGePoint3d] – position of light in world space (for point and spot light types).
* m\_direction [OdGeVector3d] – direction of light in world space (for distant and spot light types).
* m\_color [float[3]] – diffuse color of light (color components is in 0-1 range).
* m\_constantAttenuation [float] – constant light attenuation component (for point and spot light types).
* m\_linearAttenuation [float] – linear light attenuation component (for point and spot light types).
* m\_quadraticAttenuation [float] – quadratic light attenuation component (for point and spot light types).
* m\_spotCutoff [float] – light spot cutoff angle (for spot light type).
* m\_spotExponent [float] – light spot exponent factor (for spot light type).
* m\_specularFactor [float] – multiplier for specular highlighting.

#### Xml

Example:

<LightsListChanged>

<ViewportID>145425480</ViewportID>

<AmbientLight>0.2, 0.2, 0.2, 1</AmbientLight>

<DefaultLightsOn>0</DefaultLightsOn>

<NLights>1</NLights>

<LightsList>

<Light>

<Type>1</Type>

<DefLight>0</DefLight>

<Position>59.9621, 125.51, 134.27</Position>

<Direction>0, 0, 1</Direction>

<Color>1, 1, 1</Color>

<ConstantAttenuation>1</ConstantAttenuation>

<LinearAttenuation>0</LinearAttenuation>

<QuadraticAttenuation>0</QuadraticAttenuation>

<SpotCutoff>0</SpotCutoff>

<SpotExponent>0</SpotExponent>

<SpecularFactor>1</SpecularFactor>

</Light>

</LightsList>

</LightsListChanged>

* ViewportID [string] – identifier of viewport to modify lights list.
* AmbientLight [float \* 4] – ambient environment color (RGBA color components in 0-1 range).
* DefaultLightsOn [boolean] – informs renderer that default light sources must be taken into account during visualization or not.
* NLights [integer] – number of lights in lights list.
* LightsList [tag] – set of <Light> tags each of which specifies light settings to be added in viewport:
* Type [integer] – set to 0 for distant light type, set to 1 for point light type, or set to 2 for spot light type.
* DefLight [boolean] – set to true if this light is generated by default.
* Position [float \* 3] – light position in world space.
* Direction [float \* 3] – light direction in world space.
* Color [float \* 3] – light color (RGB components in 0-1 range).
* ConstantAttenuation [float] – constant light attenuation component.
* LinearAttenuation [float] – linear light attenuation component.
* QuadraticAttenuation [float] – quadratic light attenuation component.
* SpotCutoff [float] – light spot cutoff angle.
* SpotExponent [float] – light spot exponent factor.
* SpecularFactor [float] – multiplier for specular highlighting.

### onBackgroundChanged

**virtual** **void** onBackgroundChanged(OdTrVisViewportId viewportId, OdTrVisFlatMetafileContainerPtr pStream, OdTrVisProgramId baseProgramId = kTrVisNegativeId) = 0;

#### C++

This method will be called by vectorizer in case if viewport background graphics changed. Background graphics is represented as graphics stream which must be rendered in view coordinate system using shading program which identifier is set to baseProgramId argument.

Metafile graphics stream is described in “Metafile graphics stream” documentation section.

#### Xml

Example:

<BackgroundChanged>

<ViewportID>145425480</ViewportID>

<BaseProgramID>0</BaseProgramID>

<Visible>1</Visible>

<MetafileData>

<NArrays>3</NArrays>

<Array>

<Type>0</Type>

<NData>6</NData>

<ArrayData>-1.33333,-1,0.0916741,1.33333,-1,0.0916741,1.33333,1,0.0916741,-1.33333,-1,0.0916741,1.33333,1,0.0916741,-1.33333,1,0.0916741</ArrayData>

</Array>

<Array>

<Type>1</Type>

<NData>6</NData>

<ArrayData>0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1</ArrayData>

</Array>

<Array>

<Type>3</Type>

<NData>6</NData>

<ArrayData>0,0,1,0,1,1,0,0,1,1,0,1</ArrayData>

</Array>

<MetafileStream>

<Program>

<ProgramID>2</ProgramID>

</Program>

<InitTexture>

<TextureID>145554960</TextureID>

<Repeat>1</Repeat>

</InitTexture>

<EnableArray>

<Type>0</Type>

<NArray>0</NArray>

</EnableArray>

<EnableArray>

<Type>3</Type>

<NArray>1</NArray>

</EnableArray>

<EnableArray>

<Type>2</Type>

<NArray>2</NArray>

</EnableArray>

<DrawArrays>

<Mode>4</Mode>

<First>0</First>

<Count>6</Count>

</DrawArrays>

<DisableArray>

<Type>2</Type>

</DisableArray>

<DisableArray>

<Type>3</Type>

</DisableArray>

<DisableArray>

<Type>0</Type>

</DisableArray>

<UninitTexture>

</UninitTexture>

<Program>

<ProgramID>0</ProgramID>

</Program>

</MetafileStream>

</MetafileData>

</BackgroundChanged>

* ViewportID [string] – identifier of viewport to set background graphics.
* BaseProgramID [string] – identifier of shading program which must be used for background graphics rendering.
* Visible [boolean] – set to 1 if background metafile graphics stream available, elsewhere set to 0 – background not rendered.
* MetafileData [tag] – represents graphics stream for background rendering. Graphics stream must be rendered in view coordinate system.

Metafile graphics stream is described in “Metafile graphics stream” documentation section.

### onExtentsChanged

**virtual** **void** onExtentsChanged(OdTrVisViewportId viewportId, OdTrVisOverlayId overlayId, **const** OdTrVisExtentsDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if geometry extents for graphics inside viewport onto specified overlay are changed. Scene extents is set to extents of entire geometry inside viewport and useful for different rendering optimizations. Real extents are set to extents which is useful for implement “zoom to extents” functionality on renderer side.

OdTrVisExtentsDef member’s description:

* m\_sceneExtents [OdGeExtents3d] – extents of entire geometry inside viewport; useful for different rendering optimizations.
* m\_viewingMatrix [OdGeMatrix3d] – extents which is useful for implement “zoom to extents” functionality on renderer side.

#### Xml

Example:

<ExtentsChanged>

<ViewportID>145425480</ViewportID>

<OverlayID>0</OverlayID>

<SceneMin>-62.838, -6.62359, -0.617878</SceneMin>

<SceneMax>178.42, 224.798, 146.607</SceneMax>

<RealMin>-62.838, -6.62359, -0.617878</RealMin>

<RealMax>178.42, 224.798, 146.607</RealMax>

</ExtentsChanged>

* ViewportID [string] – identifier of viewport to update geometry extents.
* OverlayID [string] – identifier of overlay buffer contains viewport to update geometry extents.
* SceneMin [float \* 3] – minimum geometry extents.
* SceneMax [float \* 3] – maximum geometry extents.
* RealMin [float \* 3] – minimum extents for “zoom to extents” functionality.
* RealMax [float \* 3] – maximum extents for “zoom to extents” functionality.

### onOverlayViewParamsOverride

**virtual** **void** onOverlayViewParamsOverride(OdTrVisViewportId viewportId, OdTrVisOverlayId overlayId, **bool** bOverride, **const** OdTrVisViewParamsDef &pDef = OdTrVisViewParamsDef()) = 0;

#### C++

This method will be called by vectorizer in case if required to use different view orientation and transformation parameters for specified viewport onto specified overlay buffer. In case this method doesn’t call or “bOverride” argument was set to false – viewport will be rendered onto specified overlay buffer with default view orientation and transformation parameters (which is provided with viewport).

OdTrVisViewParamsDef member’s description:

* m\_screenMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform coordinates from normalized device space to screen space.
* m\_viewingMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform coordinates from world space to view space.
* m\_projectionMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform from view space to normalized device space.
* m\_correctionMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform from screen space to OpenGL viewport space.
* m\_outputMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform coordinates from metafile space to view space.
* m\_metafileMatrix [OdGeMatrix3d] – specifies matrix which can be used to transform coordinates from world space to metafile space.
* m\_viewPosition [OdGePoint3d] – specifies viewer origin in world space.
* m\_viewBasis [OdGeVector3d[3]] – specifies viewer X, Y and Z axes in world space.
* m\_bPerspectiveProjection [bool] – set to true if perspective projection is enabled, elsewhere orthogonal projection will be used.
* m\_fieldWidth [double] – specifies camera field width.
* m\_fieldHeight [double] – specifies camera field height.
* m\_lensLength [double] – specifies camera lens length for perspective projection.
* m\_viewTarget [OdGePoint3d] – specifies viewer target point in world space.
* m\_frontClip [OdTrVisViewClipped] – specifies setting of view front clipping plane.
* m\_backClip [OdTrVisViewClipped] – specifies setting of view back clipping plane.

#### Xml

Example:

<OverlayViewParamsOverride>

<ViewportID>145425480</ViewportID>

<OverlayID>0</OverlayID>

<Override>1</Override>

<ViewParams>

<ScreenMatrix>1643, 0, 0, 0, 0, -620, 0, 620, 0, 0, 1, 0, 0, 0, 0, 1</ScreenMatrix>

<ViewingMatrix>1, -0, -0, -57.791, -0, 0, 1, -81.423, 0, -1, 0, 109.087, 0, 0, 0, 1</ViewingMatrix>

<ProjectionMatrix>0.000999458, 0, -0.00039274, 0.5, 0, 0.00264856, -0.00039274, 0.5, 0, 0, -1.00079, 1, 0, 0, -0.00078548, 1</ProjectionMatrix>

<CorrectionMatrix>2, 0, 0, -1, 0, 2, 0, -1, 0, 0, 2, -1, 0, 0, 0, 1</CorrectionMatrix>

<OutputMatrix>1, -0, -0, -57.791, -0, 0, 1, -81.423, 0, -1, 0, 109.087, 0, 0, 0, 1</OutputMatrix>

<MetafileMatrix>1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1</MetafileMatrix>

<ViewPosition>57.791, -1021.1, 81.423</ViewPosition>

<ViewBasisX>1, 0, -0</ViewBasisX>

<ViewBasisY>0, 0, 1</ViewBasisY>

<ViewBasisZ>0, -1, 0</ViewBasisZ>

<PerspectiveProjection>1</PerspectiveProjection>

<FieldWidth>871.05</FieldWidth>

<FieldHeight>377.563</FieldHeight>

<LensLength>50</LensLength>

<ViewTarget>57.791, 109.087, 81.423</ViewTarget>

<FrontClip>1</FrontClip>

<FrontClipDist>1.5</FrontClipDist>

<BackClip>1</BackClip>

<BackClipDist>-1.5</BackClipDist>

</ViewParams>

</OverlayViewParamsOverride>

* ViewportID [string] – identifier of viewport to override view orientation and transformation settings.
* OverlayID [string] – identifier of overlay buffer contains viewport to override view orientation and transformation settings.
* Override [boolean] – set to 1 if view parameters must be overrides for specified overlay or 0 elsewhere.
* ViewParams [tag] – represents block with view orientation and transformation parameters (available only in case if “Override” value was set to 0):
  + ScreenMatrix [float \* 16] – matrix for transform coordinates from normalized device space to screen space.
  + ViewingMatrix [float \* 16] – matrix for transform coordinates from world space to view space.
  + ProjectionMatrix [float \* 16] – matrix for transform coordinates from view space to normalized device space.
  + CorrectionMatrix [float \* 16] – matrix for transform coordinates from screen space to OpenGL viewport space.
  + OutputMatrix [float \* 16] – matrix for transform coordinates from metafile space to view space.
  + MetafileMatrix [float \* 16] – matrix for transform coordinates from world space to metafile space.
  + ViewPosition [float \* 3] – camera position in world space.
  + ViewBasisX [float \* 3] – camera X-axis in world space.
  + ViewBasisY [float \* 3] – camera Y-axis in world space.
  + ViewBasisZ [float \* 3] – camera Z-axis in world space.
  + PerspectiveProjection [boolean] – set to 1 if perspective projection is enabled.
  + FieldWidth [float] – camera field width.
  + FieldHeight [float] – camera field height.
  + LensLength [float] – camera lens length for perspective projection.
  + ViewTarget [float \* 3] – camera target position in world space.
  + FrontClip [boolean] – set to 1 if front view clipping is enabled.
  + FrontClipDist [float] – front view clipping distance.
  + BackClip [boolean] – set to 1 if back view clipping is enabled.
  + BackClipDist [float] – back view clipping distance.

### onMetafileAdded

**virtual** OdTrVisDisplayId onMetafileAdded(OdTrVisMetafileId metafileId, **const** OdTrVisMetafileDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if new graphics metafile is added into graphics scene.

This method returns metafile identifier by default. For optimization purposes renderer implementation could return its own pointer as OdTrVisDisplayId, this pointer will be passed into display list instead of metafile identifier for direct rendering.

OdTrVisMetafileDef member’s description:

* m\_pMetafile [OdTrVisMetafileContainerPtr] – smart pointer onto metafile data:
* OdTrVisMetafileContainer::m\_mfFlags [OdUInt16] – set of metafile bit flags:
* OdTrVisMetafileContainer::kMfVisible – set if metafile graphics is visible.
* OdTrVisMetafileContainer::kMfHighlighted – set if metafile must be rendered as highlighted.
* OdTrVisMetafileContainer::kMfTemporary – set if metafile represents temporary graphics (not related to database elements).
* OdTrVisMetafileContainer::kMfNested – set if metafile represents nested database elements (entities inside blocks).
* OdTrVisMetafileContainer::kMfBlockRef – set if metafile represents database element which contain nested elements (blocks).
* OdTrVisMetafileContainer::kMfUtilitary – set if metafile represents utility cached graphics (for example True Type font glyphs).
* OdTrVisMetafileContainer::kMfSectionable – set if metafile can be clipped by section planes.
* OdTrVisMetafileContainer::kMfComposite – set if metafile contains data for multi-pass rendering.
* OdTrVisMetafileContainer::kMfXRef – set if metafile represents external reference graphics.
* OdTrVisMetafileContainer::kMfRefEditFade – set if metafile must be rendered as faded (in case if fading amount contains reasonable value).
* OdTrVisMetafileContainer::m\_pOwnerId [OdDbStub\*] – database object ID for which metafile graphics was generated.
* OdTrVisMetafileContainer::m\_pHlBranch [OdTrVisHlBranchId] – reserved. Always set to null on onMetafileAdded call.
* OdTrVisMetafileContainer::m\_pLayer [OdTrVisLayerId] – layer object ID linked with this metafile.
* m\_extents [OdGeExtents3d] – metafile geometry extents.
* m\_generatorId [OdTrVisViewportId] – viewport identifier for which this metafile is firstly generated.

OdTrVisMetafileContainer is inherited from OdTrVisFlatMetafileContainer which represents metafile graphics stream. Metafile graphics stream is described in “Metafile graphics stream” documentation section.

#### Xml

Example:

<MetafileAdded>

<MetafileID>51778696</MetafileID>

<OwnerID>1FC</OwnerID>

<LayerID>1957364957</LayerID>

<GeneratorID>145425480</GeneratorID>

<Visibility>1</Visibility>

<Highlighted>0</Highlighted>

<Temporary>0</Temporary>

<Nested>0</Nested>

<BlockRef>0</BlockRef>

<Utilitary>1</Utilitary>

<Sectionable>0</Sectionable>

<Composite>0</Sectionable>

<XRef>0</XRef>

<Faded>0</Faded>

<ExtMin>-62.82, -6.08473, -1.07895e-015</ExtMin>

<ExtMax>177.674, 224.549, 138.624</ExtMax>

<MetafileData>

<NArrays>3</NArrays>

<Array>

<Type>0</Type>

<NData>4</NData>

<ArrayData>177.022,223.897,138.624,177.022,-6.08473,138.624,177.674,-6.08473,0,177.674,224.549,0,177.022,223.897,138.624,177.674</ArrayData>

</Array>

<Array>

<Type>1</Type>

<NData>4</NData>

<ArrayData>0.999989,0,0.0046996,0.999989,0,0.0046996,0.999989,0,0.0046996,0.999989,0,0.0046996</ArrayData>

</Array>

<Array>

<Type>3</Type>

<NData>4</NData>

<ArrayData>4.85429,3.48985,-0.895009,3.48352,-0.891042,0.0179299,4.87455,0.0242791</ArrayData>

</Array>

<MetafileStream>

<Material>

<MaterialID>51693592</MaterialID>

</Material>

<Color>

<Color>255, 255, 255, 255</Color>

</Color>

<EnableOpt>

<Mode>2</Mode>

</EnableOpt>

<EnableArray>

<Type>0</Type>

<NArray>0</NArray>

</EnableArray>

<EnableArray>

<Type>3</Type>

<NArray>1</NArray>

</EnableArray>

<EnableArray>

<Type>2</Type>

<NArray>2</NArray>

</EnableArray>

<DrawArrays>

<Mode>4</Mode>

<First>0</First>

<Count>84</Count>

</DrawArrays>

<DisableArray>

<Type>2</Type>

</DisableArray>

<DisableArray>

<Type>3</Type>

</DisableArray>

<DisableOpt>

<Mode>2</Mode>

</DisableOpt>

<DrawArrays>

<Mode>1</Mode>

<First>84</First>

<Count>104</Count>

</DrawArrays>

<DisableArray>

<Type>0</Type>

</DisableArray>

</MetafileStream>

</MetafileData>

</MetafileAdded>

* MetafileID [string] – identifier of metafile to be added into graphics scene.
* OwnerID [string] – database handle of element for which this metafile was generated.
* LayerID [string] – layer object ID linked with this metafile.
* GeneratorID [string] – identifier of viewport for which this metafile firstly generated.
* Visibility [boolean] – set to 1 if metafile graphics is visible.
* Highlighted [boolean] – set to 1 if metafile must be rendered as highlighted.
* Temporary [boolean] – set to 1 if metafile represents temporary graphics (not related to database elements).
* Nested [boolean] – set to 1 if metafile represents nested database elements (entities inside blocks).
* BlockRef [boolean] – set to 1 if metafile represents database element which contain nested elements (blocks).
* Utilitary [boolean] – set to 1 if metafile represents utility cached graphics (for example True Type font glyphs).
* Sectionable [boolean] – set to 1 if metafile can be clipped by section planes.
* Composite [boolean] – set to 1 if metafile contains data for multi-pass rendering.
* XRef [boolean] – set to 1 if metafile represents external reference graphics.
* Faded [boolean] – set to 1 if metafile must be rendered as faded (in case if fading amount contains reasonable value).
* ExtMin [float \* 3] – minimum geometry extents.
* ExtMax [float \* 3] – maximum geometry extents.
* MetafileData [tag] – represents graphics stream for metafile rendering. Graphics stream must be rendered in metafiles coordinate system.

Metafile graphics stream is described in “Metafile graphics stream” documentation section.

### onMetafileDeleted

**virtual** **void** onMetafileDeleted(OdTrVisMetafileId metafileId) = 0;

#### C++

This method will be called by vectorizer in case if graphics metafile must be deleted from graphics scene.

#### Xml

Example:

<MetafileDeleted>

<MetafileID>145555960</MetafileID>

</MetafileDeleted>

* MetafileID [string] – identifier of metafile to be removed from graphics scene.

### onMetafileVisibilityChanged

**virtual** **void** onMetafileVisibilityChanged(OdTrVisMetafileId metafileId, **bool** bVisibility) = 0;

#### C++

This method will be called by vectorizer in case if graphics metafile visibility state is to be changed.

#### Xml

Example:

<MetafileVisibilityChanged>

<MetafileID>145555960</MetafileID>

<Visibility>0</Visibility>

</MetafileVisibilityChanged>

* MetafileID [string] – identifier of metafile to change visibility state.
* Visibility [boolean] – new visibility state.

### onMetafileHighlightingChanged

**virtual** **void** onMetafileHighlightingChanged(OdTrVisMetafileId metafileId, **bool** bHighlight) = 0;

#### C++

This method will be called by vectorizer in case if graphics metafile highlighting state is to be changed.

#### Xml

Example:

<MetafileHighlightingChanged>

<MetafileID>145555960</MetafileID>

<Highlight>1</Highlight>

</MetafileHighlightingChanged>

* MetafileID [string] – identifier of metafile to change highlighting state.
* Highlight [boolean] – new highlighting state.

### onMetafileFadingChanged

**virtual** **void** onMetafileFadingChanged(OdTrVisMetafileId metafileId, **bool** bFade) = 0;

#### C++

This method will be called by vectorizer in case if graphics metafile fading state is to be changed.

#### Xml

Example:

<MetafileFadingChanged>

<MetafileID>145555960</MetafileID>

<Fade>0</Fade>

</MetafileFadingChanged>

* MetafileID [string] – identifier of metafile to change fading state.
* Fade [boolean] – new fading state.

### onMetafileHlBranchAttached

**virtual** **void** onMetafileHlBranchAttached(OdTrVisMetafileId metafileId, OdTrVisHlBranchId hlBranchId) = 0;

#### C++

This method will be called by vectorizer in case if highlighting branch is to be attached for this graphics metafile.

#### Xml

Example:

<MetafileHlBranchAttached>

<MetafileID>145555960</MetafileID>

<HlBranchID>51778696</HlBranchID>

</MetafileHlBranchAttached>

* MetafileID [string] – identifier of metafile to attach highlighting branch.
* HlBranchID [string] – identifier of attached highlighting branch.

### onMetafileHlBranchDetached

**virtual** **void** onMetafileHlBranchDetached(OdTrVisMetafileId metafileId) = 0;

#### C++

This method will be called by vectorizer in case if highlighting branch is to be detached from this graphics metafile.

#### Xml

Example:

<MetafileHlBranchDetached>

<MetafileID>145555960</MetafileID>

</MetafileHlBranchDetached>

* MetafileID [string] – identifier of metafile to detach highlighting branch.

### onHlBranchAdded

**virtual** **void** onHlBranchAdded(OdTrVisHlBranchId hlBranchId, **const** OdTrVisHlBranchDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if new highlighting branch is added into graphics scene.

Highlighting branch consists from: 1) database object ID; 2) list of child highlighting branches to traverse through highlighting branches tree during rendering process; 3) list of highlighting markers to highlight sub elements of graphic metafiles.

#### Xml

Example:

<HlBranchAdded>

<HlBranchID>51778696</HlBranchID>

<OwnerID>1FC</OwnerID>

<NChilds>1</NChilds>

<NMarkers>2</NMarkers>

<Childs>

<HlBranchId>51763776</HlBranchId>

</Childs>

<Markers>

<Marker>1</Marker>

<Marker>3</Marker>

</Markers>

</HlBranchAdded>

* HlBranchID [string] – identifier of highlighting branch to be added into graphics scene.
* OwnerID [string] – database handle of element for which this highlighting branch was generated.
* NChilds [integer] – number of child highlighting branches for this highlighting branch.
* NMarkers [integer] – number of highlighting markers for this highlighting branch.
* Childs [tag] – set of <HlBranchId> tags [string] each of which specifies child highlighting branch for this highlighting branch.
* Markers [tag] – set of <Marker> tags [64 bit integer] each of which specifies highlighting marker for this highlighting branch.

### onHlBranchModified

**virtual** **void** onHlBranchModified(OdTrVisHlBranchId hlBranchId, **const** OdTrVisHlBranchDef &pDef, **bool** bChildsModified, **bool** bMarkersModified) = 0;

#### C++

This method will be called by vectorizer in case if exist highlighting branch is modified. bChildsModified will be set to true if child highlighting branches list is modified for this highlighting branch. bMarkersModified will be set to true if highlighting markers list is modified for this highlighting branch.

#### Xml

Example:

<HlBranchModified>

<HlBranchID>51778696</HlBranchID>

<NChilds>1</NChilds>

<NMarkers>2</NMarkers>

<Childs>

<HlBranchID>51763776</HlBranchID>

</Childs>

<Markers>

<Marker>1</Marker>

<Marker>3</Marker>

</Markers>

</HlBranchModified>

* HlBranchID [string] – identifier of modified highlighting branch.
* NChilds [integer] – number of new child highlighting branches for this highlighting branch.
* NMarkers [integer] – number of new highlighting markers for this highlighting branch.
* Childs [tag] – set of <HlBranchID> tags [string] each of which specifies child highlighting branch for this highlighting branch.
* Markers [tag] – set of <Marker> tags [64 bit integer] each of which specifies highlighting marker for this highlighting branch.

*Note*: in case if NChilds or NMarkers specified, exist child highlighting branches or highlighting markers must be removed from highlighting branch.

### onHlBranchDeleted

**virtual** **void** onHlBranchDeleted(OdTrVisHlBranchId hlBranchId) = 0;

#### C++

This method will be called by vectorizer in case if exist highlighting branch is removed from graphics scene.

#### Xml

Example:

<HlBranchDeleted>

<HlBranchID>51778696</HlBranchID>

</HlBranchDeleted>

* HlBranchID [string] – identifier of deleted highlighting branch.

### onTextureAdded

**virtual** **void** onTextureAdded(OdTrVisTextureId textureId, **const** OdTrVisTextureDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if new texture is added into graphics scene.

OdTrVisTextureDef member’s description:

* m\_flags [OdUInt16] – set of texture-related flags:
  + OdTrVisTextureDef::kSmoothFilter – enabled if texture could have any type of interpolation for minimization and magnification filters during rendering, elsewhere interpolation must be disabled during rendering.
  + OdTrVisTextureDef::kModulateColor – enabled if texture color could be modulated with underlying vertex color, elsewhere texture color will always replace underlying vertex color.
* m\_pTexture [pointer] – pointer to the OdTrVisTexture texture interface.

#### Xml

Example:

<TextureAdded>

<TextureID>145554768</TextureID>

<Type>1</Type>

<Smooth>1</Smooth>

<Modulate>0</Modulate>

<Width>256</Width>

<Height>128</Height>

<ScanlineLength>768</ScanlineLength>

<DataAlignment>4</DataAlignment>

<TextureData>1C2FEA115D3A2EAA12590DA0DE</TextureData>

<PaletteType>4</PaletteType>

<PaletteWidth>2</PaletteWidth>

<PaletteScanlineLength>768</PaletteScanlineLength>

<PaletteDataAlignment>4</PaletteDataAlignment>

<PaletteData>A2EAA12590DA0DE1C2FEA115D3</PaletteData>

</TextureAdded>

* TextureID [string] – identifier of added texture.
* Type [integer] – texture data type:
* 0 – 3-byte per pixel texture data with RGB colors order.
* 1 – 3-byte per pixel texture data with BGR colors order.
* 2 – 4-byte per pixel texture data with RGBA colors order.
* 3 – 4-byte per pixel texture data with BGRA colors order.
* 4 – 4-float per pixel texture data with RGBA colors order.
* 5 – 1-byte per pixel texture data with palette
* Smooth [boolean] – set to 1 if texture could have any type of interpolation for minimization and magnification filters during rendering, elsewhere interpolation must be disabled during rendering.
* Modulate [boolean] – set to 1 if texture color could be modulated with underlying vertex color, elsewhere texture color will always replace underlying vertex color.
* Width [integer] – texture width in pixels.
* Height [integer] – texture height in pixels.
* ScanlineLength [integer] – length of single texture row in bytes including alignment.
* DataAlignment [integer] – texture data alignment
* TextureData [binary] – encoded texture binary data.
* PaletteType [integer] – texture palette data type, similar to Type
* PaletteWidth [integer] – number of colors in texture palette.
* PaletteScanlineLength [integer] – length of single palette row in bytes including alignment
* PaletteDataAlignment [integer] – palette data alignment
* PaletteData [binary] – encoded texture binary data.

### onTextureDeleted

**virtual** **void** onTextureDeleted(OdTrVisTextureId textureId) = 0;

#### C++

This method will be called by vectorizer in case if exist texture was removed from graphics scene.

#### Xml

Example:

<TextureDeleted>

<TextureID>145554768</TextureID>

</TextureDeleted>

* TextureID [string] – identifier of texture to remove.

### onMaterialAdded

**virtual** **void** onMaterialAdded(OdTrVisMaterialId materialId, **const** OdTrVisMaterialDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if new material is added into graphics scene.

OdTrVisMaterialDef member’s description:

* m\_name – material name
* m\_bDefault – “Default” flag; value “true” is used for ByBlock, ByLayer and GLOBAL materials
* m\_ambientColor [float[3]] – material color for ambient lighting (RGB color components in 0-1 range).
* m\_shadingAmbientColor [float[3]] – material color for ambient lighting in non-realistic shading modes (RGB color components in 0-1 range).
* m\_diffuseColor [float[3]] – material color for diffuse lighting (RGB color components in 0-1 range).
* m\_shadingDiffuseColor [float[3]] – material color for diffuse lighting in non-realistic shading modes (RGB color components in 0-1 range).
* m\_specularColor [float[3]] – material color for specular highlighting (RGB color components in 0-1 range).
* m\_shadingSpecularColor [float[3]] – material color for specular highlighting in non-realistic shading modes (RGB color components in 0-1 range).
* m\_emissionColor [float[3]] – material emission color (RGB color components in 0-1 range).
* m\_shadingEmissionColor [float[3]] – material emission color for non-realistic shading modes (RGB color components in 0-1 range).
* m\_blendFactor [float] – factor to blend between material diffuse color and vertex color (0 – full material color; 1 – full vertex color).
* m\_shadingBlendFactor [float] – factor to blend between material diffuse color and vertex color in non-realistic shading modes (0 – full material color; 1 – full vertex color).
* m\_opacityLevel [float] – material transparency (0 – completely transparent; 1 – opaque).
* m\_shadingOpacityLevel [float] – material transparency in non-realistic shading modes (0 – completely transparent; 1 – opaque).
* m\_specularPower [float] – specular highlighting (gloss) factor.
* m\_textureBlend [float] – factor to blend between texture and material diffuse color (0 – full diffuse color; 1 – full texture color).
* m\_textureId (OdTrVisTextureId) – optional material texture identifier.
* m\_uWrap (enum) – horizontal texture wrapping mode:
* OdTrVisMaterialDef::kWrapRepeat – repeat texture if texture coordinates outside 0-1 range.
* OdTrVisMaterialDef::kWrapClamp – clamp texture to edges if texture coordinates outside 0-1 range.
* OdTrVisMaterialDef::kWrapCrop – crops texture to transparent color if texture coordinates outside 0-1 range.
* OdTrVisMaterialDef::kWrapMirror – similar as repeat, but each odd texture repetition is inverted.
* m\_vWrap (enum) – vertical texture wrapping mode:
* OdTrVisMaterialDef::kWrapRepeat – repeat texture if texture coordinates outside 0-1 range.
* OdTrVisMaterialDef::kWrapClamp – clamp texture to edges if texture coordinates outside 0-1 range.
* OdTrVisMaterialDef::kWrapCrop – crops texture to transparent color if texture coordinates outside 0-1 range.
* OdTrVisMaterialDef::kWrapMirror – similar as repeat, but each odd texture repetition is inverted.

#### Xml

Example:

<MaterialAdded>

<MaterialID>51697328</MaterialID>

<MaterialName>New\_material</MaterialName>

<IsDefault>0</IsDefault>

<AmbientColor>0.501961, 0.501961, 0.501961</AmbientColor>

<ShadingAmbientColor>0.501961, 0.501961, 0.501961</ShadingAmbientColor>

<DiffuseColor>0.501961, 0.501961, 0.501961</DiffuseColor>

<ShadingDiffuseColor>0.501961, 0.501961, 0.501961</ShadingDiffuseColor>

<SpecularColor>1, 1, 1</SpecularColor>

<ShadingSpecularColor>1, 1, 1</ShadingSpecularColor>

<EmissionColor>0, 0, 0</EmissionColor>

<ShadingEmissionColor>0, 0, 0</ShadingEmissionColor>

<BlendFactor>0</BlendFactor>

<ShadingBlendFactor>0</ShadingBlendFactor>

<OpacityLevel>1</OpacityLevel>

<ShadingOpacityLevel>1</ShadingOpacityLevel>

<SpecularPower>64</SpecularPower>

<TextureBlend>1</TextureBlend>

<TextureID>51764384</TextureID>

<UWrapType>0</UWrapType>

<VWrapType>0</VWrapType>

</MaterialAdded>

* MaterialID [string] – identifier of material to add.
* MaterialName[string] – material name
* IsDefault[boolean] – “Default” material flag; 1 used for ByBlock, Bylayer and GLOBAL materials
* AmbientColor [float \* 3] – material color for ambient lighting (RGB color components in 0-1 range).
* ShadingAmbientColor [float \* 3] – material color for ambient lighting in non-realistic shading modes (RGB color components in 0-1 range).
* DiffuseColor [float \* 3] – material color for diffuse lighting (RGB color components in 0-1 range).
* ShadingDiffuseColor [float \* 3] – material color for diffuse lighting in non-realistic shading modes (RGB color components in 0-1 range).
* SpecularColor [float \* 3] – material color for specular highlighting (RGB color components in 0-1 range).
* ShadingSpecularColor [float \* 3] – material color for specular highlighting in non-realistic shading modes (RGB color components in 0-1 range).
* EmissionColor [float \* 3] – material emission color (RGB color components in 0-1 range).
* ShadingEmissionColor [float \* 3] – material emission color for non-realistic shading modes (RGB color components in 0-1 range).
* BlendFactor [float] – factor to blend between material diffuse color and vertex color (0 – full material color; 1 – full vertex color).
* ShadingBlendFactor [float] – factor to blend between material diffuse color and vertex color in non-realistic shading modes (0 – full material color; 1 – full vertex color).
* OpacityLevel [float] – material transparency (0 – completely transparent; 1 – opaque).
* ShadingOpacityLevel [float] – material transparency in non-realistic shading modes (0 – completely transparent; 1 – opaque).
* SpecularPower [float] – specular highlighting (gloss) factor.
* TextureBlend [float] – factor to blend between texture and material diffuse color (0 – full diffuse color; 1 – full texture color).
* TextureID [string] – optional identifier of material texture.
* UWrapType [integer] - horizontal texture wrapping mode:
* 0 – repeat texture if texture coordinates outside 0-1 range.
* 1 – clamp texture to edges if texture coordinates outside 0-1 range.
* 2 – crops texture to transparent color if texture coordinates outside 0-1 range.
* 3 – similar as repeat, but each odd texture repetition is inverted.
* VWrapType [integer] - vertical texture wrapping mode:
* 0 – repeat texture if texture coordinates outside 0-1 range.
* 1 – clamp texture to edges if texture coordinates outside 0-1 range.
* 2 – crops texture to transparent color if texture coordinates outside 0-1 range.
* 3 – similar as repeat, but each odd texture repetition is inverted.

### onMaterialDeleted

**virtual** **void** onMaterialDeleted(OdTrVisMaterialId materialId) = 0;

#### C++

This method will be called by vectorizer in case if exist material was removed from graphics scene.

#### Xml

Example:

<MaterialDeleted>

<MaterialID>51697328</MaterialID>

</MaterialDeleted>

* MaterialID [string] – identifier of material to remove.

### onVisualStyleAdded

**virtual** **void** onVisualStyleAdded(OdTrVisVisualStyleId visualStyleId, **const** OdTrVisVisualStyle &pDef) = 0;

#### C++

This method will be called by vectorizer in case if new visual style is added into graphics scene. ODTrVisVisualStyle class represents set of properties which modifies rendering behavior of different graphic primitives.

#### Xml

Example:

<VisualStyleAdded>

<VisualStyleID>65734937</VisualStyleID>

<Type>8</Type>

<Name>2dWireframe</Name>

<Internal>0</Internal>

<Temporary>0</Temporary>

<NProps>58</NProps>

<NPropsModified>58</NPropsModified>

<Prop>

<NProp>0</NProp>

<PropType>1</PropType>

<Set>1</Set>

<BVal>0</BVal>

</Prop>

<Prop>

<NProp>1</NProp>

<PropType>2</PropType>

<Set>1</Set>

<IVal>32</IVal>

</Prop>

<Prop>

<NProp>2</NProp>

<PropType>3</PropType>

<Set>1</Set>

<DVal>0.74</DVal>

</Prop>

<Prop>

<NProp>3</NProp>

<PropType>4</PropType>

<Set>1</Set>

<CVal>874363927364</CVal>

</Prop>

<Prop>

<NProp>4</NProp>

<PropType>5</PropType>

<Set>1</Set>

<SVal>linetype.ltp</SVal>

</Prop>

. . .

</VisualStyleAdded>

* VisualStyleID [string] – identifier of visual style to add.
* Type [integer] – generic type of visual style.
* Name [string] – visual style name.
* Internal [boolean] – marks internal (not accessible through UI) visual styles.
* Temporary [boolean] – marks temporary (non-database resident) visual styles.
* NProps [integer] – number of properties into visual style.
* NPropsModified [integer] – number of properties to read from file.
* Prop [tag] – set of single property settings into visual style.
* NProp [integer] – number of property inside visual style.
* PropType [integer] – type of property (1 – boolean, 2 – integer, 3 – double, 4 – color, 5 – text).
* Set [boolean] – set to 1 in case if property setting applicable, elsewhere property settings must be inherited from parent visual style.
* BVal [boolean] – boolean property type data setting.
* IVal [integer] – integer property type data setting.
* DVal [float] – floating point property type data setting.
* CVal [integer] – color property type data setting.
* SVal [string] – text string property type data setting.

### onVisualStyleModified

**virtual** **void** onVisualStyleModified(OdTrVisVisualStyleId visualStyleId, **const** OdTrVisVisualStyle &pDef) = 0;

#### C++

This method will be called by vectorizer in case if exist visual style is modified into graphics scene. ODTrVisVisualStyle class represents set of properties which modifies rendering behavior of different graphic primitives.

#### Xml

Example:

<VisualStyleModified>

<VisualStyleID>65734937</VisualStyleID>

<Type>8</Type>

<Name>2dWireframe</Name>

<Internal>0</Internal>

<Temporary>0</Temporary>

<NProps>58</NProps>

<NPropsModified>5</NPropsModified>

<Prop>

<NProp>0</NProp>

<PropType>1</PropType>

<Set>1</Set>

<BVal>0</BVal>

</Prop>

<Prop>

<NProp>1</NProp>

<PropType>2</PropType>

<Set>1</Set>

<IVal>32</IVal>

</Prop>

<Prop>

<NProp>2</NProp>

<PropType>3</PropType>

<Set>1</Set>

<DVal>0.74</DVal>

</Prop>

<Prop>

<NProp>3</NProp>

<PropType>4</PropType>

<Set>1</Set>

<CVal>874363927364</CVal>

</Prop>

<Prop>

<NProp>4</NProp>

<PropType>5</PropType>

<Set>1</Set>

<SVal>linetype.ltp</SVal>

</Prop>

</VisualStyleModified>

* VisualStyleID [string] – identifier of visual style to add.
* Type [integer] – generic type of visual style.
* Name [string] – visual style name.
* Internal [boolean] – marks internal (not accessible through UI) visual styles.
* Temporary [boolean] – marks temporary (non-database resident) visual styles.
* NProps [integer] – number of properties into visual style.
* NPropsModified [integer] – number of properties to read from file.
* Prop [tag] – set of single property settings into visual style.
* NProp [integer] – number of property inside visual style.
* PropType [integer] – type of property (1 – boolean, 2 – integer, 3 – double, 4 – color, 5 – text).
* Set [boolean] – set to 1 in case if property setting applicable, elsewhere property settings must be inherited from parent visual style.
* BVal [boolean] – boolean property type data setting.
* IVal [integer] – integer property type data setting.
* DVal [float] – floating point property type data setting.
* CVal [integer] – color property type data setting.
* SVal [string] – text string property type data setting.

### onVisualStyleDeleted

**virtual** **void** onVisualStyleDeleted(OdTrVisVisualStyleId visualStyleId) = 0;

#### C++

This method will be called by vectorizer in case if exist visual style was removed from graphics scene.

#### Xml

Example:

<VisualStyleDeleted>

<VisualStyleID>65734937</VisualStyleID>

</VisualStyleDeleted>

* VisualStyleID [string] – identifier of visual style to remove.

### onLayerAdded

**virtual** **void** onLayerAdded(OdTrVisLayerId layerId, **const** OdTrVisLayerDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if new layer is added into graphics scene.

OdTrVisLayerDef member’s description:

* m\_name [OdString] – layer name.
* m\_pOwner [OdDbStub\*] – owner database object ID.
* m\_props [OdTrVisLayerDef::LayerProps] – structure represents set of generic layer properties and flags:
  + OdTrVisLayerDef::kPersistent – layer cannot be deleted during editing.
  + OdTrVisLayerDef::kTemporary – layer haven’t linked owner database object ID.
  + OdTrVisLayerDef::kFrozen – layer is frozen.
  + OdTrVisLayerDef::kOff – layer is in disabled state.
  + OdTrVisLayerDef::kVisible – layer is visible.
  + OdTrVisLayerDef::kPlottable – layer can be printed.
  + OdTrVisLayerDef::kLocked – layer is locked.

#### Xml

Example:

<LayerAdded>

<LayerID>443253466</LayerID>

<Name>0</Name>

<OwnerID>77BA</OwnerId>

<Props>177</Props>

</LayerAdded>

* LayerID [string] – identifier of layer to add.
* Name [string] – layer name.
* OwnerID [string] – owner database object handle.
* Props [integer] – set of bit flags represent generic layer properties:
  + 1 – layer cannot be deleted during editing.
  + 2 – layer haven’t linked owner database object ID.
  + 3 – layer is frozen.
  + 4 – layer is in disabled state.
  + 5 – layer is visible.
  + 6 – layer can be printed.
  + 7 – layer is locked.

### onLayerModified

**virtual** **void** onLayerModified(OdTrVisLayerId layerId, **const** OdTrVisLayerDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if exist layer is modified into graphics scene. For OdTrVisLayerDef structure member’s description look into “onLayerAdded” method reference.

#### Xml

Example:

<LayerModified>

<LayerID>443253466</LayerID>

<Name>0</Name>

<Props>177</Props>

</LayerModified>

* LayerID [string] – identifier of layer to modify.
* Name [string] – layer name.
* Props [integer] – set of bit flags represent generic layer properties:
  + 1 – layer cannot be deleted during editing.
  + 2 – layer haven’t linked owner database object ID.
  + 3 – layer is frozen.
  + 4 – layer is in disabled state.
  + 5 – layer is visible.
  + 6 – layer can be printed.
  + 7 – layer is locked.

### onLayerDeleted

**virtual** **void** onLayerDeleted(OdTrVisLayerId layerId) = 0;

#### C++

This method will be called by vectorizer in case if exist layer was removed from graphics scene.

#### Xml

Example:

<LayerDeleted>

<LayerID>443253466</LayerID>

</LayerDeleted>

* LayerID [string] – identifier of layer to remove.

### onLayerViewportPropsOverride

**virtual** **void** onLayerViewportPropsOverride(OdTrVisViewportId viewportId, OdTrVisLayerId layerId, **bool** bOverride, **const** OdTrVisLayerDef::LayerProps &pProps = OdTrVisLayerDef::LayerProps()) = 0;

#### C++

This method will be called by vectorizer in case if viewport-dependent layer properties were added, modified or removed.

#### Xml

Example:

<LayerViewportPropsOverride>

<ViewportID>218</ViewportID>

<LayerID>1FA</LayerID>

<Override>1</Override>

<VpDepSpec>

<Props>164</Props>

</VpDepSpec>

</LayerViewportPropsOverride>

<LayerViewportPropsOverride>

<ViewportID>219</ViewportID>

<LayerID>1FA</LayerID>

<Override>0</Override>

</LayerViewportPropsOverride>

* ViewportID [string] – viewport ID with which layer contains specific properties.
* LayerID [string] – layer ID that contains specific properties.
* Override [boolean] – set to 1 if layer properties must be overrides for specified viewport or 0 elsewhere.
* Props [integer] – set of bit flags represent viewport-dependent layer properties:
  + 1 – layer cannot be deleted during editing.
  + 2 – layer haven’t linked owner database object ID.
  + 3 – layer is frozen.
  + 4 – layer is in disabled state.
  + 5 – layer is visible.
  + 6 – layer can be printed.
  + 7 – layer is locked.

### Obsolete: onVertexShader

**virtual** **void** onVertexShader(OdGLES2VertexShaderId shaderId, **const** **char** \*pProgram) = 0;

#### C++

This method was excluded from rendition interface. Previously this method is called by OpenGL ES2 vectorizer in case if previously unused vertex shading GLSL program become required for rendering.

GLSL program is compatible with OpenGL ES2 and OpenGL 2.0v GLSL specifications.

#### Xml

Example:

<VertexShader>

<ShaderID>3</ShaderID>

<Program>#ifdef GL\_ES

precision mediump float;

precision mediump int;

#endif

…

</Program>

</VertexShader>

* ShaderID [string] – identifier of vertex shader.
* Program [string] – GLSL shading program in ASCII codepage.

### Obsolete: onFragmentShader

**virtual** **void** onFragmentShader(OdGLES2FragmentShaderId shaderId, **const** **char** \*pProgram) = 0;

#### C++

This method was excluded from rendition interface. Previously this method is called by OpenGL ES2 vectorizer in case if previously unused fragment shading GLSL program become required for rendering.

GLSL program is compatible with OpenGL ES2 and OpenGL 2.0v GLSL specifications.

#### Xml

Example:

<FragmentShader>

<ShaderID>4</ShaderID>

<Program>#ifdef GL\_ES

precision mediump float;

precision mediump int;

#endif

…

</Program>

</VertexShader>

* ShaderID [string] – identifier of fragment shader.
* Program [string] – GLSL shading program in ASCII codepage.

### Obsolete: onShaderProgram

**virtual** **void** onShaderProgram(OdGLES2ProgramId programId, **const** OdGLES2ProgramDef &pDef) = 0;

#### C++

This method was excluded from rendition interface. Previously this method is called by OpenGL ES2 vectorizer in case if previously unused shading program become required for rendering.

OdGLES2ProgramDef member’s description:

* m\_vertexShaderId [OdGLES2VertexShaderId] – identifier of vertex shading program component for this shading program.
* m\_fragmentShaderId [OdGLES2FragmentShaderId] – identifier of fragment shading program component for this shading program.
* m\_nGeomMarkBase [OdUInt32] – set of bit flags which specifies render modes for which this shading program can be treated as base. Bit flags is compatible with geometry marker flags which is described in “36 – GeomMarker record” documentation section.
* m\_nGeomMarkSpec [OdUInt32] – set of bit flags which specify compatibility flags for this shading program. Bit flags is compatible with geometry marker flags which is described in “36 – GeomMarker record” documentation section.
* m\_pAttribsList [const char\*] – list of program attributes names separated by new line character.
* m\_pUniformsList [const char\*] – list of program uniforms names separated by new line character.
* m\_pAttribsMapping [const OdUInt32\*] – list of mappings between attributes enumeration and program attribute name.
* m\_nAttribsMapping [OdUInt32] – number of mappings in m\_pAttribsMapping list.
* m\_pUniformsMapping [const OdUInt32\*] – list of mappings between attributes enumeration and program uniform name.
* m\_nUniformsMapping [OdUInt32] – number of mappings in m\_pUniformsMapping list.

#### Xml

Example:

<ShaderProgram>

<ProgramID>4</ProgramID>

<VertexShaderID>3</VertexShaderID>

<FragmentShaderID>4</FragmentShaderID>

<GeomMarkerBase>1</GeomMarkerBase>

<GeomMarkerSpec>77, 0, 0, 0</GeomMarkerSpec>

<AttribsList>a\_VertPosition

a\_TexCoord0

a\_Normal</AttribsList>

<UniformsList>u\_XformMatrix

u\_ColorVec

u\_Highlighted

u\_TextureUnit0

u\_TexLut

u\_TextureUnit1

u\_Lighting

u\_ViewPosition

u\_AmbientLight

u\_Material

u\_Lights</UniformsList>

<NAttribMappings>4</NAttribMappings>

<NUniformMappings>13</NUniformMappings>

<AttribMappings>

<Mapping>0</Mapping>

<Mapping>4294967295</Mapping>

<Mapping>1</Mapping>

<Mapping>2</Mapping>

</AttribMappings>

<UniformMappings>

<Mapping>0</Mapping>

<Mapping>1</Mapping>

<Mapping>2</Mapping>

<Mapping>3</Mapping>

<Mapping>4</Mapping>

<Mapping>5</Mapping>

<Mapping>4294967295</Mapping>

<Mapping>4294967295</Mapping>

<Mapping>6</Mapping>

<Mapping>7</Mapping>

<Mapping>8</Mapping>

<Mapping>9</Mapping>

<Mapping>10</Mapping>

</UniformMappings>

</ShaderProgram>

* ProgramID [string] – identifier of program.
* VertexShaderID [string] - identifier of vertex shading program component for this shading program.
* FragmentShaderID [string] - identifier of fragment shading program component for this shading program.
* GeomMarkerBase [integer] – set of bit flags which specifies render modes for which this shading program can be treated as base. Bit flags is compatible with geometry marker flags which is described in “GeomMarker record” documentation section.
* GeomMarkerSpec [integer \* 4] – set of bit flags which specify compatibility flags for this shading program. Bit flags is compatible with geometry marker flags which is described in “GeomMarker record” documentation section.
* AttribsList [string] - list of program attributes names separated by new line character.
* UniformsList [string] - list of program uniforms names separated by new line character.
* NAttribMappings [integer] – number of attribute mappings.
* NUniformMappings [integer] – number of uniform mappings.
* AttribMappings [tag] – set of <Mapping> tags [integer] each of which specifies mapping between attribute index and program attribute name.
* UniformMappings [tag] – set of <Mapping> tags [integer] each of which specifies mapping between uniform index and program attribute name.

## Metafile graphics stream

Contents:

* C++ - binary graphics stream representation.
* Xml graphics stream representation.

### C++ - binary graphics stream representation

Binary graphics stream is accessible through OdTrVisFlatMetafileContainer interface:

**inline** **const** OdUInt8 \*memoryPtr() **const;**

Stream haven’t termination symbol, so client renderer could use stream size to check for binary stream completion:

OdUInt32 size() **const**

Binary stream could refer to binary data arrays which is also accessible through OdTrVisFlatMetafileContainer interface:

OdArray<OdTrVisArrayWrapper> m\_ArrayElements;

For list of array types refer to “Binary data array types” chapter.

Binary stream consists from 1-byte chunks and optional record data:

chunk

[record data]

chunk

[record data]

…

Example of simple binary stream data parser:

**void** ParseMetafileStream(**const** OdTrVisFlatMetafileContainer \*pMetafile)

{

OdUInt32 uSize = pMetafile->size();

**if** (uSize == 0) **return**;

**const** OdUInt8 \*pMemPtr = pMetafile->memoryPtr();

**const** OdUInt8 \*pMemPtrReadFor = pMemPtr + uSize;

**while** (pMemPtr < pMemPtrReadFor)

{

OdUInt8 chunk = \*pMemPtr;

pMemPtr++;

**switch** (chunk)

{

**case** 0: **break**; // skip empty record

**case** 1: // EnableOpt

{

OdUInt8 mode = \*pMemPtr;

pMemPtr++;

// use mode . . .

}

**break**;

**case** 2: // DisableOpt

{

OdUInt8 mode = \*pMemPtr;

pMemPtr++;

// use mode . . .

}

**break**;

// . . . process other chunks

**default**: // error

}

}

}

Subsequent chapters describe the types of records allowed into binary stream.

#### Binary data array types

##### Vertex array

Data type: float

Count: 3 (XYZ) per vertex

##### Normals array

Data type: float

Count: 3 (XYZ) per normal

##### Colors array

Data type: float

Count: 4 (RGBA) per color

##### Texture coordinates array

Data type: float

Count: 2 (UV) per texture coordinate

##### Depths array

Data type: float

Count: 1 (single floating point variable per depth)

##### Indexes array

Data type: OdUInt16

Count: 1

##### Markers array

Data type: OdTrVisDefProcMark

Count: 1

#### 0 – Empty record

Chunk: 0

Data size: 0

Description:

Empty records don’t cause any actions on renderer side. They could be available in binary stream to fix strict alignment issues during cast stream memory pointer into different data types.

#### 1 – EnableOpt record

Chunk: 1

Data size: 1 byte

Description:

Single byte value in this record represents rendering option code to enable during rendering process. Following codes can be present for this record:

* 0 – Depth – enable depth buffering.
* 1 – Blend – enable blending mode.

#### 2 – DisableOpt record

Chunk: 2

Data size: 1 byte

Description:

Single byte value in this record represents rendering option code to disable during rendering process. Following codes can be present for this record:

* 0 – Depth – disable depth buffering.
* 1 – Blend – disable blending mode.

#### 3 – Color record

Chunk: 3

Data size: 4 bytes

Description:

4 bytes in this record represents color RGBA components to be set for geometry rendering.

Data structure:

* OdUInt8 – red color component.
* OdUInt8 – green color component.
* OdUInt8 – blue color component.
* OdUInt8 – alpha color component.

#### 4 – EnableArray record

Chunk: 4

Data size: 5 bytes.

Description:

This record setup array for rendering.

Data structure:

* OdUInt8 – array type. Possible values:
* 0 – vertex array.
* 1 – colors array.
* 2 – texture coordinates array.
* 3 – normals array.
* 4 – secondary normals array.
* 5 – depths array.
* OdUInt32 – index of array in arrays list attached to OdTrVisFlatMetafileContainer.

#### 5 – DisableArray record

Chunk: 5

Data size: 1 byte.

Description:

This record disable array after rendering.

Data structure:

* OdUInt8 – array type. Possible values:
* 0 – vertex array.
* 1 – colors array.
* 2 – texture coordinates array.
* 3 – normals array.
* 4 – secondary normals array.
* 5 – depths array.

#### 6 – DrawArrays record

Chunk: 6

Data size: 9 bytes

Description:

This record is used to finally draw arrays which are set using EnableArray records.

Data structure:

* OdUInt8 – rendering primitive type. Possible values:
* 0 – render points (1 vertex per point).
* 1 – render lines (2 vertexes per line).
* 2 – render lines loop.
* 3 – render lines strip.
* 4 – render filled triangles (3 vertexes per triangle).
* 5 – render filled triangles strip.
* 6 – render filled triangles fan.
* OdInt32 – index of first vertex in array from which primitive rendering must be started.
* OdInt32 – number of vertexes to be rendered.

#### 7 – DrawElements record

Chunk: 7

Data size: 9 bytes

Description:

This record is used to draw arrays using indexes array. Each index in indexes array represent index of vertex in vertexes array which is previously set using EnableArray record.

Data structure:

* OdUInt8 – rendering primitive type. Possible values:
* 0 – render points (1 vertex per point).
* 1 – render lines (2 vertexes per line).
* 2 – render lines loop.
* 3 – render lines strip.
* 4 – render filled triangles (3 vertexes per triangle).
* 5 – render filled triangles strip.
* 6 – render filled triangles fan.
* OdInt32 – number of vertexes to be rendered.
* OdUInt32 - index of indexes array in arrays list attached to OdTrVisFlatMetafileContainer.

#### 8 – CullFace record

Chunk: 8

Data size: 1 byte

Description:

Set mode for faces culling.

Data structure:

* OdUInt8 – culling mode. Possible values:
* 0 – disable faces culling.
* 1 – enable back faces culling.
* 2 – enable front faces culling.
* 3 – faces culling mode can be selected by renderer.

#### 9 – LStipple record

Chunk: 9

Data size: 1 byte

Description:

Sets line stippling pattern.

Data structure:

* OdUInt8 – stippling pattern index (look for TD PlotStyle linetypes specification). 0 – disable stippling pattern (solid pattern).

#### 10 – PStipple record

Chunk: 10

Data size: 1 byte

Description:

Sets triangle stippling pattern.

Data structure:

* OdUInt8 – stippling pattern index (look for TD PlotStyle fillstyles specification; add 64 to convert into PlotStyle representation). 0 – disable stippling pattern (solid pattern).

#### 11 – VPoint record

Chunk: 11

Data size: 12 bytes

Description:

Draw single point.

Data structure:

* float – point X coordinate.
* float – point Y coordinate.
* float – point Z coordinate.

#### 12 – VLine record

Chunk: 12

Data size: 24 bytes

Description:

Draw single line.

Data structure:

* float – line start X coordinate.
* float – line start Y coordinate.
* float – line start Z coordinate.
* float – line end X coordinate.
* float – line end Y coordinate.
* float – line end Z coordinate.

#### 13 – IPoint record

Chunk: 13

Data size: 4 bytes

Description:

Draw single point using index in vertexes array.

Data structure:

* OdInt32 – index in currently set vertexes array.

#### 14 – ILine record

Chunk: 14

Data size: 8 bytes

Description:

Draw single line using indexes in vertex array.

Data structure:

* OdInt32 – line start vertex index in currently set vertexes array.
* OdInt32 – line end vertex index in currently set vertexes array.

#### 15 – HLRStencil record

Chunk: 15

Data size: 1 byte

Description:

Enable/disable Hidden Line shading for rendered triangles. If Hidden Line shading enabled, renderer can use faces filling color from current viewport to fill subsequent triangles.

Data structure:

* OdUInt8 – Hidden Line shading state (0 – disable, 1 – enable).

#### 16 – EnableShading record

Chunk: 16

Data size: 1 byte

Description:

Single byte value in this record represents geometry shading option code to enable during rendering process. Following codes can be present for this record:

* 0 – Lighting – enable faces lighting for shaded mode.
* 1 – Highlighting – enable geometry highlighting.
* 2 – Gouraud – enable per-vertex color buffer.
* 3 – MultiNormals – marks facets geometry in case if vertex normal is specified (separate secondary array of normals can be used for FlatShaded render modes).
* 4 – Disable2dLineweights – disable lineweights in 2dOptimized render mode.
* 5 – NoColorOverride – marks geometry which shouldn’t be influenced by color modifiers.

#### 17 – DisableShading record

Chunk: 17

Data size: 1 byte

Description:

Single byte value in this record represents geometry shading option code to disable during rendering process. Following codes can be present for this record:

* 0 – Lighting – disable faces lighting for shaded mode.
* 1 – Highlighting – disable geometry highlighting.
* 2 – Gouraud – disable per-vertex color buffer.
* 3 – MultiNormals – disable multiple normal arrays.
* 4 – Disable2dLineweights – enables lineweights in 2dOptimized render mode.
* 5 – NoColorOverride – ends geometry which shouldn’t be influenced by color modifiers.

#### 18 – Material record

Chunk: 18

Data size: 8 bytes

Description:

Setup material for shaded rendering mode.

Data structure:

* OdUInt64 – material identifier.

#### 19 – UserEntry record

Chunk: 19

Data size: 4 bytes + variable length

Description:

User-defined data entry. Typically represents debug marker strings in ASCII form.

Data structure:

* OdUInt32 – data size in bytes.
* OdUInt8 \* size of data in bytes – binary data stream.

#### 20 – InitTexture record

Chunk: 20

Data size: 9 bytes

Description:

Enable texture for triangles rendering (supported for all rendering modes).

Data structure:

* OdUInt64 – texture identifier.
* OdUInt8 – texture wrapping mode (0 – clamp to edges; 1 – repeat).

#### 21 – UninitTexture record

Chunk: 21

Data size: 0

Description:

Disable triangles texturization which is previously enabled using InitTexture record.

#### 22 – SelectionMarker record

Chunk: 22

Data size: 8 bytes.

Description:

Sets selection marker which will be used for next geometry selection.

Data structure:

* OdUInt64 – selection marker.

#### 23 – EnableMarkerArray record

Chunk: 23

Data size: 5 bytes

Description:

Setup selection or metafile markers array for underlying geometry.

Data structure:

* OdUInt8 – array type. 0 – selection markers; 1 – metafile markers.
* OdUInt32 – index of markers array in arrays list attached to OdTrVisFlatMetafileContainer.

#### 24 – DisableMarkerArray record

Chunk: 24

Data size: 1

Description:

Disable usage of previously set selection or metafile markers array for underlying geometry.

Data structure:

* OdUInt8 – array type. 0 – selection markers; 1 – metafile markers.

#### 25 – SelectionFlags record

Chunk: 25

Data size: 1

Description:

Provide set of flags which modify behavior of geometry selection and highlighting.

Data structure:

* OdUInt8 – set of bit flags (as in OdTrVisSelectionFlags enumeration: bit 1 – don’t display geometry if it is not highlighted; bit 2 – don’t select geometry if it is not highlighted; bit 3 – don’t display geometry if it is highlighted; bit 4 – don’t select geometry if it is highlighted).

#### 26 – Lineweight record

Chunk: 26

Data size: 3-9 bytes

Description:

Setup current lineweight for points and lines geometry rendering.

Data structure:

* OdUInt8 – lineweight type (0 – lineweight from enum attached to viewport; 1 – lineweight as floating point variable; 2 – lineweight directly set in pixels; 3 – lineweight directly set in geometry coordinates system).
* double [if type is set to 1 or 3] – lineweight (if type set to 1 it is must be multiplied by coefficient attached to viewport, elsewhere it is can be drawn in geometry coordinates).
* OdInt16 [if type is set to 0 or 2] – index of lineweight in lineweights array attached to viewport (if type set to 0), or lineweight in pixels (if type set to 2).

#### 27 – Linestyle record

Chunk: 27

Data size: 1-3 bytes

Description:

Setup style for linewight caps and joins displaying.

Data structure:

* OdUInt8 – set to 1 for default style, 0 – for non-default.
* OdUInt8 [if type is set to 0] – lineweight caps style.
* OdUInt8 [if type is set to 0] – lineweight joins style.

#### 28 – Program record

Chunk: 28

Data size: 8 bytes

Description:

Change currently set shading program. These records will be available in metafile streams only in case if renderer implements “queryProgramId” method.

Data structure:

* OdUInt64 – shading program identifier.

#### 29 – TtfText record

Chunk: 29

Data size: variable

Description:

Render string of cached True Type font characters.

Data structure:

* OdUInt64 – first part of font key.
* OdUInt64 – second part of font key.
* OdGeMatrix3d – initial transformation matrix for characters chain rendering.
* OdGeVector3d – characters offset vector.
* OdUInt32 – number of characters in text string.
* OdUInt32 \* number of characters – single OdUInt32 value per text string character represent character code in UTF-32 encoding.

#### 30 – PushMatrix record

Chunk: 30

Data size: 1-129 bytes

Description:

Informs renderer that it must apply additional transformation matrix for all metafiles which will come after this record. Matrices can be nested, so renderer must multiply this matrix with previous matrices if them available. Best practice is to use matrix stack for manage transformation matrices.

Data structure:

* OdUInt8 – matrix type (0 – identity matrix must be applied; 1 – matrix is attached to this record).
* OdGeMatrix3d [if type set to 1] – transformation matrix.

#### 31 – PopMatrix record

Chunk: 31

Data size: 0

Description:

Informs renderer that previously applied transformation matrix must be removed from a matrices stack.

#### 32 – Metafile record

Chunk: 32

Data size: 8 bytes

Description:

Draw nested metafile.

Data structure:

* OdUInt64 – metafile identifier.

#### 33 – GeomMarker record

Chunk: 33

Data size: 1 byte

Description:

Sends current geometry type to renderer (typically used if composite metafiles mode enabled).

Data structure:

* OdUInt8 – geometry type (as in OdTrVisGeomType enumeration).

Geometry types:

* 0 – Default – default geometry type which will be rendered in all render modes.
* 1 – 2dFacets – marks non-shaded facets.
* 2 – 2dFacetsNoFill – marks non-shaded facets which aren’t rendered in 2d render mode.
* 3 – 3dFacets – marks shaded facets.
* 4 – 3dFacetsNoFill – marks shaded facets which aren’t rendered in 2d render mode.
* 5 – RasterImageFacets – marks raster image facets.
* 6 – 2dFacetEdges – marks non-shaded facet edges.
* 7 – 3dFacetEdges – marks shaded facet edges.
* 8 – Isolines – marks isolines geometry.
* 9 – EdgesWithIsolines – marks edges which represent isolines geometry too.
* 10 – HatchIsolineEdges – isolines renderable in all render modes except HiddenLine.
* 11 – Silhouettes – marks silhouettes geometry.

#### 34 – VisualStyle record

Chunk: 34

Data size: 8 bytes

Description:

Setups visual style override for subsequent geometry rendering.

Data structure:

* OdUInt64 – visual style identifier.

#### 35 – MetafileMarker record

Chunk: 35

Data size: 8 bytes.

Description:

Sets metafile marker which will be used for underlying geometry.

Data structure:

* OdUInt64 – metafile identifier.

### Xml graphics stream representation

Graphics metafile stream example:

<MetafileData>

<NArrays>3</NArrays>

<Array>

<Type>0</Type>

<NData>14</NData>

<ArrayData>-62.838,224.47,0,-62.838,-6.62359,0,177.757,-6.62359,0,177.757,224.47,0,-62.838,224.47,0,177.757,-6.62359,0,-62.838,224.47,0,-62.838,-6.62359,0,-62.838,-6.62359,0,177.757,-6.62359,0,177.757,224.47,0,-62.838,224.47,0,177.757,-6.62359,0,177.757,224.47,0</ArrayData>

</Array>

<Array>

<Type>1</Type>

<NData>6</NData>

<ArrayData>0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1</ArrayData>

</Array>

<Array>

<Type>3</Type>

<NData>6</NData>

<ArrayData>-7.10543e-017,2.31093,-7.10543e-017,0,2.40595,0,2.40595,2.31093,-7.10543e-017,2.31093,2.40595,0</ArrayData>

</Array>

<MetafileStream>

<Material>

<MaterialID>51697328</MaterialID>

</Material>

<Color>

<Color>255, 255, 255, 255</Color>

</Color>

<EnableOpt>

<Mode>0</Mode>

</EnableOpt>

<EnableArray>

<Type>0</Type>

<NArray>0</NArray>

</EnableArray>

<EnableArray>

<Type>3</Type>

<NArray>1</NArray>

</EnableArray>

<EnableArray>

<Type>2</Type>

<NArray>2</NArray>

</EnableArray>

<DrawArrays>

<Mode>4</Mode>

<First>0</First>

<Count>6</Count>

</DrawArrays>

<DisableArray>

<Type>2</Type>

</DisableArray>

<DisableArray>

<Type>3</Type>

</DisableArray>

<DisableOpt>

<Mode>0</Mode>

</DisableOpt>

<DrawArrays>

<Mode>1</Mode>

<First>6</First>

<Count>8</Count>

</DrawArrays>

<DisableArray>

<Type>0</Type>

</DisableArray>

</MetafileStream>

</MetafileData>

Metafile data consists from arrays list and metafile stream. <NArrays> tag specifies how many arrays available in arrays list. Array types are described in “Metafile data array types” section.

Subsequent chapters describe the types of records allowed into metafile stream.

#### Metafile data array types

##### Vertex array

Example:

<Array>

<Type>0</Type>

<NData>14</NData>

<ArrayData>-62.838,224.47,0,-62.838,-6.62359,0,177.757,-6.62359,0,177.757,224.47,0,-62.838,224.47,0,177.757,-6.62359,0,-62.838,224.47,0,-62.838,-6.62359,0,-62.838,-6.62359,0,177.757,-6.62359,0,177.757,224.47,0,-62.838,224.47,0,177.757,-6.62359,0,177.757,224.47,0</ArrayData>

</Array>

* Type [integer] – 0 for vertex arrays.
* NData [integer] – number of vertexes.
* ArrayData [float \* 3 \* NData] – vertexes array; 3 float variables (XYZ) per vertex.

##### Normals array

Example:

<Array>

<Type>1</Type>

<NData>6</NData>

<ArrayData>0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1</ArrayData>

</Array>

* Type [integer] – 1 for normals array.
* NData [integer] – number of normals.
* ArrayData [float \* 3 \* NData] – normals array; 3 float variables (XYZ) per normal.

##### Colors array

Example:

<Array>

<Type>2</Type>

<NData>2</NData>

<ArrayData>1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,1</ArrayData>

</Array>

* Type [integer] – 2 for colors array.
* NData [integer] – number of colors in array.
* ArrayData [float \* 4 \* NData] – colors array; 4 float variables (RGBA) per color; colors in 0-1 range.

##### Texture coordinates array

Example:

<Array>

<Type>3</Type>

<NData>6</NData>

<ArrayData>-7.10543e-017,2.31093,-7.10543e-017,0,2.40595,0,2.40595,2.31093,-7.10543e-017,2.31093,2.40595,0</ArrayData>

</Array>

* Type [integer] – 3 for texture coordinates array.
* NData [integer] – number of texture coordinates in array.
* ArrayData [float \* 2 \* NData] – texture coordinates array; 2 float variables (UV) per texture coordinate.

##### Depths array

Example:

<Array>

<Type>4</Type>

<NData>4</NData>

<ArrayData>0.1,0.2,0.3,0.5</ArrayData>

</Array>

* Type [integer] – 4 for depths array.
* NData [integer] – number of depths in array.
* ArrayData [float \* NData] – depths array; single float variable per depth.

##### Indexes array

Example:

<Array>

<Type>5</Type>

<NData>6</NData>

<ArrayData>0,1,2,1,2,3</ArrayData>

</Array>

* Type [integer] – 5 for indexes array.
* NData [integer] – number of indexes in array.
* ArrayData [integer \* NData] – indexes array.

##### Markers array

Example:

<Array>

<Type>6</Type>

<NData>2</NData>

<Data>

<Entry>

<Marker>1</Marker>

<From>0</Marker>

</Entry>

<Entry>

<Marker>2</Marker>

<From>4</Marker>

</Entry>

</Data>

</Array>

* Type [integer] – 6 for selection or metafile markers array.
* NData [integer] – number of markers in array.
* Data [tag] – array data:
* Entry [tag] – entry for each marker in array:
* Marker [64 bit integer] – selection or metafile marker.
* From [integer] – vertex index from which marker must be applied for geometry.

#### EnableOpt record

Example:

<EnableOpt>

<Mode>0</Mode>

</EnableOpt>

Single <Mode> value [integer] in this record represents rendering option code to enable during rendering process. Following codes can be present for this record:

* 0 – Depth – enable depth buffering.
* 1 – Blend – enable blending mode.

#### DisableOpt record

Example:

<DisableOpt>

<Mode>0</Mode>

</DisableOpt>

Single <Mode> value [integer] in this record represents rendering option code to disable during rendering process. Following codes can be present for this record:

* 0 – Depth – disable depth buffering.
* 1 – Blend – disable blending mode.

#### Color record

Example:

<Color>

<Color>255, 255, 255, 255</Color>

</Color>

4 integers in this record represents color RGBA components to be set for geometry rendering. Color components specified in 0-255 range.

#### EnableArray record

Example:

<EnableArray>

<Type>2</Type>

<NArray>2</NArray>

</EnableArray>

This record setup array for rendering.

* Type [integer] – array type. Possible values:
* 0 – vertex array.
* 1 – colors array
* 2 – texture coordinates array.
* 3 – normals array.
* 4 – secondary normals array.
* 5 – depths array.
* NArray [integer] – index of array in metafile arrays list.

#### DisableArray record

Example:

<DisableArray>

<Type>2</Type>

</DisableArray>

This record disable array after rendering.

* Type [integer] – array type. Possible values:
* 0 – vertex array.
* 1 – colors array
* 2 – texture coordinates array.
* 3 – normals array.
* 4 – secondary normals array.
* 5 – depths array.

#### DrawArrays record

Example:

<DrawArrays>

<Mode>1</Mode>

<First>6</First>

<Count>8</Count>

</DrawArrays>

This record is used to finally draw arrays which are set using EnableArray records.

* Mode [integer] – rendering primitive type. Possible values:
* 0 – render points (1 vertex per point).
* 1 – render lines (2 vertexes per line).
* 2 – render lines loop.
* 3 – render lines strip.
* 4 – render filled triangles (3 vertexes per triangle).
* 5 – render filled triangles strip.
* 6 – render filled triangles fan.
* First [integer] – index of first vertex in array from which primitive rendering must be started.
* Count [integer] – number of vertexes to be rendered.

#### DrawElements record

Example:

<DrawElements>

<Mode>1</Mode>

<Count>6</Count>

<NArray>2</NArray>

</DrawElements>

This record is used to draw arrays using indexes array. Each index in indexes array represent index of vertex in vertexes array which is previously set using EnableArray record.

* Mode [integer] – rendering primitive type. Possible values:
* 0 – render points (1 vertex per point).
* 1 – render lines (2 vertexes per line).
* 2 – render lines loop.
* 3 – render lines strip.
* 4 – render filled triangles (3 vertexes per triangle).
* 5 – render filled triangles strip.
* 6 – render filled triangles fan.
* Count [integer] – number of vertexes to be rendered.
* NArray [integer] - index of indexes array in metafile arrays list.

#### CullFace record

Example:

<CullFace>

<Mode>1</Mode>

</CullFace>

Set mode for faces culling.

* Mode [integer] – culling mode. Possible values:
* 0 – disable faces culling.
* 1 – enable back faces culling.
* 2 – enable front faces culling.
* 3 – faces culling mode can be selected by renderer.

#### LStipple record

Example:

<LStipple>

<Pattern>1</Pattern>

</LStipple>

Sets line stippling pattern.

* Pattern [integer] – stippling pattern index (look for TD PlotStyle linetypes specification), 0 – disable stippling pattern (solid pattern).

#### PStipple record

Example:

<PStipple>

<Pattern>1</Pattern>

</PStipple>

Sets triangle stippling pattern.

* Pattern [integer] – stippling pattern index (look for TD PlotStyle fillstyles specification; add 64 to convert into PlotStyle representation), 0 – disable stippling pattern (solid pattern).

#### VPoint record

Example:

<VPoint>

<Point>1.057, 2.2, 0.001</Point>

</VPoint>

Draw single point.

* Point [float \* 3] – XYZ point coordinates.

#### VLine record

Example:

<VLine>

<Point1>1.057, 2.2, 0.001</Point1>

<Point2>2.066, 3.3, 0.001</Point2>

</VLine>

Draw single line.

* Point1 [float \* 3] – XYZ line start coordinates.
* Point2 [float \* 3] – XYZ line end coordinates.

#### IPoint record

Example:

<IPoint>

<Index>77</Index>

</IPoint>

Draw single point using index in vertexes array.

* Index [integer] – index in currently set vertexes array.

#### ILine record

Example:

<ILine>

<Index1>77</Index1>

<Index2>78</Index2>

</ILine>

Draw single line using indexes in vertex array.

* Index1 [integer] – line start vertex index in currently set vertexes array.
* Index2 [integer] – line end vertex index in currently set vertexes array.

#### HLRStencil record

Example:

<HLRStencil>

<State>1</State>

</HLRStencil>

Enable/disable Hidden Line shading for rendered triangles. If Hidden Line shading enabled, renderer can use faces filling color from current viewport to fill subsequent triangles.

* State [boolean] – Hidden Line shading state (0 – disable, 1 – enable).

#### EnableShading record

Example:

<EnableShading>

<Mode>2</Mode>

</EnableShading>

Single <Mode> value [integer] in this record represents geometry shading option code to enable during rendering process. Following codes can be present for this record:

* 0 – Lighting – enable faces lighting for shaded mode.
* 1 – Highlighting – enable geometry highlighting.
* 2 – Gouraud – enable per-vertex color buffer.
* 3 – MultiNormals – marks facets geometry in case if vertex normal is specified (separate secondary array of normals can be used for FlatShaded render modes).
* 4 – Disable2dLineweights – disable lineweights in 2dOptimized render mode.
* 5 – NoColorOverride – marks geometry which shouldn’t be influenced by color modifiers.

#### DisableShading record

Example:

<DisableShading>

<Mode>2</Mode>

</DisableShading>

Single <Mode> value [integer] in this record represents geometry shading option code to disable during rendering process. Following codes can be present for this record:

* 0 – Lighting – disable faces lighting for shaded mode.
* 1 – Highlighting – disable geometry highlighting.
* 2 – Gouraud – enable per-vertex color buffer.
* 3 – MultiNormals – disable multiple normal arrays.
* 4 – Disable2dLineweights – enables lineweights in 2dOptimized render mode.
* 5 – NoColorOverride – ends geometry which shouldn’t be influenced by color modifiers.

#### Material record

Example:

<Material>

<MaterialID>51697328</MaterialID>

</Material>

Setup material for shaded rendering mode.

* MaterialID [string] – material identifier.

#### UserEntry record

Example:

<UserEntry>

<NumData>13</NumData>

<Data>1C2FEA115D3A2EAA12590DA000</Data>

</UserEntry>

User-defined data entry. Typically represents debug marker strings in ASCII form.

* NumData [integer] – data size in bytes.
* Data [binary] – encoded binary data.

#### InitTexture record

Example:

<InitTexture>

<TextureID>34697328</TextureID>

<Repeat>0</Repeat>

</InitTexture>

Enable texture for triangles rendering (supported in all rendering modes).

* TextureID [string] – texture identifier.
* Repeat [boolean] – texture wrapping mode (0 – clamp to edges; 1 – repeat).

#### UninitTexture record

Example:

<UninitTexture>

</UninitTexture>

Disable triangles texturization which is previously enabled using InitTexture record.

#### SelectionMarker record

Example:

<SelectionMarker>

<Marker>2</Marker>

</SelectionMarker>

Sets selection marker which will be used for next geometry selection.

* Marker [64 bit integer] – selection marker.

#### EnableMarkerArray record

Example:

<EnableMarkerArray>

<Type>0</Type>

<NArray>4</NArray>

</EnableMarkerArray>

Setup selection or metafile markers array for underlying geometry.

* Type [integer] – array type. 0 – selection markers; 1 – metafile markers.
* NArray [integer] – index of selection markers array in metafile arrays list.

#### DisableMarkerArray record

Example:

<DisableMarkerArray>

<Type>0</Type>

</DisableMarkerArray>

Disable usage of previously set selection or metafile markers array for underlying geometry.

* Type [integer] – array type. 0 – selection markers; 1 – metafile markers.

#### SelectionFlags record

Example:

<SelectionFlags>

<Flags>10</Flags>

</SelectionFlags>

Provide set of flags which modify behavior of geometry selection and highlighting.

* Flags [integer] – set of bit flags with information about following geometry selection and highlighting specifics (bit 1 – don’t display geometry if it is not highlighted; bit 2 – don’t select geometry if it is not highlighted; bit 3 – don’t display geometry if it is highlighted; bit 4 – don’t select geometry if it is highlighted).

#### Lineweight record

Example of pixel lineweight:

<Lineweight>

<Type>0</Type>

<Lw>6</Lw>

</Lineweight>

Example of plot style lineweight:

<Lineweight>

<Type>1</Type>

<Ps>1.25</Ps>

</Lineweight>

Setup current lineweight for points and lines geometry rendering.

* Type [boolean] – lineweight type (0 – lineweight from lineweights enum attached to viewport; 1 – lineweight as floating point variable; 2 – lineweight directly set in pixels; 3 – lineweight directly set in geometry coordinates system).
* Lw [if Type is set to 0 or 2] – index of lineweight in lineweights array attached to viewport (if Type set to 0), or lineweight in pixels (if Type set to 2).
* Ps [if Type is set to 1 or 3] – lineweight (if Type set to 1 it is must be multiplied by coefficient attached to viewport, elsewhere it is can be drawn in geometry coordinates).

#### Linestyle record

Example of default line style:

<Linestyle>

<Default>1</Default>

</Linestyle>

Example of non-default line style:

<Linestyle>

<Default>0</Default>

<CapsType>1</CapsType>

<JoinType>3</JoinType>

</Linestyle>

Setup style for linewight caps and joins displaying.

* Default [boolean] set to 1 for default style, 0 – for non-default.
* CapsType [if Default is set to 0] – type of lineweight caps (0 – butt, 1 – square, 2 – round, 3 – diamond, 4 – default).
* JoinType [if Default is set to 0] – type of lineweight joins (0 – miter, 1 – bevel, 2 – round, 3 – diamond, 5 – default).

#### Program record

Example:

<Program>

<ProgramID>3</ProgramID>

</Program>

Change currently set shading program. These records will be available in metafile streams only in case if renderer implements “queryProgramId” method.

* ProgramID [string] – shading program identifier.

#### TtfText record

Example:

<TtfText>

<NText>4</NText>

<Transform>1, -0, -0, -57.791, -0, 0, 1, -81.423, 0, -1, 0, 109.087, 0, 0, 0, 1</Transform>

<Direction>1, 0, 0</Direction>

<Text>

<Char>

<MetafileID>54472956</MetafileID>

<Multiplier>0.75</Multiplier>

</Char>

<Char>

<MetafileID>56799037</MetafileID>

<Multiplier>0.76</Multiplier>

</Char>

<Char>

<MetafileID>59874637</MetafileID>

<Multiplier>0.75</Multiplier>

</Char>

<Char>

<MetafileID>65734028</MetafileID>

<Multiplier>0.5</Multiplier>

</Char>

</Text>

</TtfText>

Render string of cached True Type font characters.

* NText [integer] – number of characters in text string.
* Transform [float \* 16] – initial transformation matrix for characters chain rendering.
* Direction [float \* 3] – characters offset vector.
* Text [tag] – open characters array:
* Char [tag] – represents single character in characters chain:
* MetafileID [string] – nested metafile identifier for character rendering.
* Multiplier [float] – multiplier on which direction vector must be multiplied for offset onto next character.

#### PushMatrix record

Example of identity matrix:

<PushMatrix>

<Identity>1</Identity>

</PushMatrix>

Example of normal matrix:

<PushMatrix>

<Matrix>1, -0, -0, -57.791, -0, 0, 1, -81.423, 0, -1, 0, 109.087, 0, 0, 0, 1</Matrix>

</PushMatrix>

Informs renderer that it must apply additional transformation matrix for all metafiles which will come after this record. Matrices can be nested, so renderer must multiply this matrix with previous matrices if them available. Best practice is to use matrix stack for manage transformation matrices.

* Identity [boolean] – informs that identity matrix must be applied.
* Matrix [if Identity field doesn’t available] – transformation matrix.

#### PopMatrix record

Example:

<PopMatrix>

</PopMatrix>

Informs renderer that previously applied transformation matrix must be removed from a matrices stack.

#### Metafile record

Example:

<Metafile>

<MetafileID>34355245672</MetafileID>

</Metafile>

Draw nested metafile.

* MetafileID [string] – metafile identifier.

#### GeomMarker record

Example:

<GeomMarker>

<Type>1</Type>

</GeomMarker>

Sends current geometry type to renderer (typically used if composite metafiles mode enabled).

* Type [integer] – represent geometry type.

Geometry types:

* 0 – Default – default geometry type which will be rendered in all render modes.
* 1 – 2dFacets – marks non-shaded facets.
* 2 – 2dFacetsNoFill – marks non-shaded facets which aren’t rendered in 2d render mode.
* 3 – 3dFacets – marks shaded facets.
* 4 – 3dFacetsNoFill – marks shaded facets which aren’t rendered in 2d render mode.
* 5 – RasterImageFacets – marks raster image facets.
* 6 – 2dFacetEdges – marks non-shaded facet edges.
* 7 – 3dFacetEdges – marks shaded facet edges.
* 8 – Isolines – marks isolines geometry.
* 9 – EdgesWithIsolines – marks edges which represent isolines geometry too.
* 10 – HatchIsolineEdges – isolines renderable in all render modes except HiddenLine.
* 11 – Silhouettes – marks silhouettes geometry.

#### VisualStyle record

Example:

<VisualStyle>

<VisualStyleID>82352949</VisualStyleID>

</VisualStyle>

Setups visual style override for subsequent geometry rendering.

* VisualStyleID [string] – visual style identifier.

#### MetafileMarker record

Example:

<MetafileMarker>

<Marker>2343876453</Marker>

</MetafileMarker>

Sets metafile marker which will be used for underlying geometry.

* Marker [string] – metafile identifier.

## Remarks

Last edits: 19.06.2018