# OdTrVisRendition interface and GsUpdate Xml protocol description

## Contents

[OdTrVisRendition interface and GsUpdate Xml protocol description 1](#_Toc90982887)

[Contents 1](#_Toc90982888)

[Introduction 11](#_Toc90982889)

[Basic data elements 11](#_Toc90982890)

[OdTrVisRendition interface methods 11](#_Toc90982891)

[onDeviceUpdateBegin/onDeviceUpdateEnd 11](#_Toc90982892)

[C++ 11](#_Toc90982893)

[Xml 11](#_Toc90982894)

[isWithinDeviceUpdate 12](#_Toc90982895)

[C++ 12](#_Toc90982896)

[Xml 12](#_Toc90982897)

[onFrameSwitchInfo 12](#_Toc90982898)

[C++ 12](#_Toc90982899)

[Xml 12](#_Toc90982900)

[queryRenditionCaps 12](#_Toc90982901)

[C++ 12](#_Toc90982902)

[Xml 13](#_Toc90982903)

[onPartialUpdateCheckLimit 13](#_Toc90982904)

[C++ 13](#_Toc90982905)

[Xml 13](#_Toc90982906)

[onPartialUpdateInvalidRects 13](#_Toc90982907)

[C++ 13](#_Toc90982908)

[Xml 13](#_Toc90982909)

[onPartialUpdateList 13](#_Toc90982910)

[C++ 13](#_Toc90982911)

[Xml 14](#_Toc90982912)

[queryPartialUpdateDirectInvocation 14](#_Toc90982913)

[C++ 15](#_Toc90982914)

[Xml 15](#_Toc90982915)

[onPartialUpdateDirectInvocation 15](#_Toc90982916)

[C++ 15](#_Toc90982917)

[Xml 15](#_Toc90982918)

[querySnapshot 15](#_Toc90982919)

[C++ 15](#_Toc90982920)

[Xml 15](#_Toc90982921)

[getSelectedObject 15](#_Toc90982922)

[C++ 15](#_Toc90982923)

[Xml 15](#_Toc90982924)

[createSharingProvider 15](#_Toc90982925)

[C++ 15](#_Toc90982926)

[Xml 16](#_Toc90982927)

[mtDataTypeProcessing 16](#_Toc90982928)

[C++ 16](#_Toc90982929)

[Xml 16](#_Toc90982930)

[idRegistrator 16](#_Toc90982931)

[C++ 16](#_Toc90982932)

[Xml 16](#_Toc90982933)

[queryMetafileStream 16](#_Toc90982934)

[C++ 16](#_Toc90982935)

[Xml 17](#_Toc90982936)

[unlockMetafileStream 17](#_Toc90982937)

[C++ 17](#_Toc90982938)

[Xml 17](#_Toc90982939)

[queryBlockMetafilesList 17](#_Toc90982940)

[C++ 17](#_Toc90982941)

[Xml 17](#_Toc90982942)

[queryExtensionObject 17](#_Toc90982943)

[C++ 17](#_Toc90982944)

[Xml 17](#_Toc90982945)

[queryProgramId 17](#_Toc90982946)

[C++ 17](#_Toc90982947)

[Xml 18](#_Toc90982948)

[saveRenditionState 18](#_Toc90982949)

[C++ 18](#_Toc90982950)

[Xml 18](#_Toc90982951)

[loadRenditionState 18](#_Toc90982952)

[C++ 18](#_Toc90982953)

[Xml 18](#_Toc90982954)

[obtainClientSettings 18](#_Toc90982955)

[C++ 18](#_Toc90982956)

[Xml 19](#_Toc90982957)

[onSurfaceChanged 19](#_Toc90982958)

[C++ 19](#_Toc90982959)

[Xml 19](#_Toc90982960)

[onSurfaceInvalidated 20](#_Toc90982961)

[C++ 20](#_Toc90982962)

[Xml 20](#_Toc90982963)

[onOverlayAdded 20](#_Toc90982964)

[C++ 20](#_Toc90982965)

[Xml 21](#_Toc90982966)

[onOverlayDeleted 21](#_Toc90982967)

[C++ 21](#_Toc90982968)

[Xml 21](#_Toc90982969)

[onOverlayVisibilityChanged 21](#_Toc90982970)

[C++ 21](#_Toc90982971)

[Xml 22](#_Toc90982972)

[onOverlayRenderOrderChanged 22](#_Toc90982973)

[C++ 22](#_Toc90982974)

[Xml 22](#_Toc90982975)

[onOverlayPropertyAttached 22](#_Toc90982976)

[C++ 22](#_Toc90982977)

[Xml 23](#_Toc90982978)

[onOverlayPropertyDetached 23](#_Toc90982979)

[C++ 23](#_Toc90982980)

[Xml 24](#_Toc90982981)

[onViewportAdded 24](#_Toc90982982)

[C++ 24](#_Toc90982983)

[Xml 24](#_Toc90982984)

[onViewportInserted 25](#_Toc90982985)

[C++ 25](#_Toc90982986)

[Xml 25](#_Toc90982987)

[onViewportDeleted 26](#_Toc90982988)

[C++ 26](#_Toc90982989)

[Xml 26](#_Toc90982990)

[onViewportVisibilityChanged 26](#_Toc90982991)

[C++ 26](#_Toc90982992)

[Xml 26](#_Toc90982993)

[onViewportViewParamsModified 26](#_Toc90982994)

[C++ 26](#_Toc90982995)

[Xml 27](#_Toc90982996)

[onViewportLineStylesModified 28](#_Toc90982997)

[C++ 28](#_Toc90982998)

[Xml 28](#_Toc90982999)

[onViewportFadingModified 29](#_Toc90983000)

[C++ 29](#_Toc90983001)

[Xml 29](#_Toc90983002)

[onViewportLineweightsModified 29](#_Toc90983003)

[C++ 29](#_Toc90983004)

[Xml 30](#_Toc90983005)

[onViewportShadingModified 30](#_Toc90983006)

[C++ 30](#_Toc90983007)

[Xml 30](#_Toc90983008)

[onViewportClipModified 31](#_Toc90983009)

[C++ 31](#_Toc90983010)

[Xml 31](#_Toc90983011)

[onViewportPropertyAttached 33](#_Toc90983012)

[C++ 33](#_Toc90983013)

[Xml 33](#_Toc90983014)

[onViewportPropertyDetached 34](#_Toc90983015)

[C++ 34](#_Toc90983016)

[Xml 34](#_Toc90983017)

[onViewportBackgroundChanged 34](#_Toc90983018)

[C++ 34](#_Toc90983019)

[Xml 35](#_Toc90983020)

[onViewportFogChanged 36](#_Toc90983021)

[C++ 36](#_Toc90983022)

[Xml 36](#_Toc90983023)

[onViewportSectioningChanged 37](#_Toc90983024)

[C++ 37](#_Toc90983025)

[Xml 37](#_Toc90983026)

[onViewportAntiAliasingChanged 38](#_Toc90983027)

[C++ 38](#_Toc90983028)

[Xml 38](#_Toc90983029)

[onViewportSSAOChanged 38](#_Toc90983030)

[C++ 38](#_Toc90983031)

[Xml 38](#_Toc90983032)

[onViewportAmbientLightChanged 38](#_Toc90983033)

[C++ 38](#_Toc90983034)

[Xml 39](#_Toc90983035)

[onViewportDefaultLightsChanged 39](#_Toc90983036)

[C++ 39](#_Toc90983037)

[Xml 40](#_Toc90983038)

[onViewportLightAttached 41](#_Toc90983039)

[C++ 41](#_Toc90983040)

[Xml 42](#_Toc90983041)

[onViewportLightDetached 43](#_Toc90983042)

[C++ 43](#_Toc90983043)

[Xml 43](#_Toc90983044)

[onOverlayViewParamsOverride 43](#_Toc90983045)

[C++ 44](#_Toc90983046)

[Xml 44](#_Toc90983047)

[onOverlayExtentsOverride 45](#_Toc90983048)

[C++ 46](#_Toc90983049)

[Xml 46](#_Toc90983050)

[onMetafileOrderChanged 46](#_Toc90983051)

[C++ 46](#_Toc90983052)

[Xml 46](#_Toc90983053)

[onMetafileOrderInheritance 47](#_Toc90983054)

[C++ 47](#_Toc90983055)

[Xml 47](#_Toc90983056)

[onMetafileAdded 48](#_Toc90983057)

[C++ 48](#_Toc90983058)

[Xml 49](#_Toc90983059)

[onMetafileDeleted 51](#_Toc90983060)

[C++ 51](#_Toc90983061)

[Xml 51](#_Toc90983062)

[onMetafileVisibilityChanged 51](#_Toc90983063)

[C++ 51](#_Toc90983064)

[Xml 51](#_Toc90983065)

[onMetafileHighlightingChanged 51](#_Toc90983066)

[C++ 51](#_Toc90983067)

[Xml 51](#_Toc90983068)

[onMetafileFadingChanged 52](#_Toc90983069)

[C++ 52](#_Toc90983070)

[Xml 52](#_Toc90983071)

[onMetafilePropertyAttached 52](#_Toc90983072)

[C++ 52](#_Toc90983073)

[Xml 52](#_Toc90983074)

[onMetafilePropertyDetached 53](#_Toc90983075)

[C++ 53](#_Toc90983076)

[Xml 53](#_Toc90983077)

[onLightAdded 54](#_Toc90983078)

[C++ 54](#_Toc90983079)

[Xml 54](#_Toc90983080)

[onLightModified 55](#_Toc90983081)

[C++ 56](#_Toc90983082)

[Xml 57](#_Toc90983083)

[onLightDeleted 58](#_Toc90983084)

[C++ 58](#_Toc90983085)

[Xml 58](#_Toc90983086)

[onLightPropertyAttached 58](#_Toc90983087)

[C++ 58](#_Toc90983088)

[Xml 58](#_Toc90983089)

[onLightPropertyDetached 59](#_Toc90983090)

[C++ 59](#_Toc90983091)

[Xml 59](#_Toc90983092)

[onStateBranchAdded 60](#_Toc90983093)

[C++ 60](#_Toc90983094)

[Xml 60](#_Toc90983095)

[onStateBranchModified 60](#_Toc90983096)

[C++ 60](#_Toc90983097)

[Xml 60](#_Toc90983098)

[onStateBranchDeleted 61](#_Toc90983099)

[C++ 61](#_Toc90983100)

[Xml 61](#_Toc90983101)

[onStateBranchPropertyAttached 61](#_Toc90983102)

[C++ 61](#_Toc90983103)

[Xml 62](#_Toc90983104)

[onStateBranchPropertyDetached 62](#_Toc90983105)

[C++ 62](#_Toc90983106)

[Xml 63](#_Toc90983107)

[onTextureAdded 63](#_Toc90983108)

[C++ 63](#_Toc90983109)

[Xml 64](#_Toc90983110)

[onTextureDeleted 65](#_Toc90983111)

[C++ 65](#_Toc90983112)

[Xml 65](#_Toc90983113)

[onTexturePropertyAttached 65](#_Toc90983114)

[C++ 66](#_Toc90983115)

[Xml 66](#_Toc90983116)

[onTexturePropertyDetached 67](#_Toc90983117)

[C++ 67](#_Toc90983118)

[Xml 67](#_Toc90983119)

[onMaterialAdded 67](#_Toc90983120)

[C++ 67](#_Toc90983121)

[Xml 69](#_Toc90983122)

[onMaterialDeleted 71](#_Toc90983123)

[C++ 71](#_Toc90983124)

[Xml 71](#_Toc90983125)

[onMaterialPropertyAttached 71](#_Toc90983126)

[C++ 71](#_Toc90983127)

[Xml 72](#_Toc90983128)

[onMaterialPropertyDetached 72](#_Toc90983129)

[C++ 72](#_Toc90983130)

[Xml 73](#_Toc90983131)

[onVisualStyleAdded 73](#_Toc90983132)

[C++ 73](#_Toc90983133)

[Xml 73](#_Toc90983134)

[onVisualStyleModified 74](#_Toc90983135)

[C++ 74](#_Toc90983136)

[Xml 74](#_Toc90983137)

[onVisualStyleDeleted 75](#_Toc90983138)

[C++ 75](#_Toc90983139)

[Xml 75](#_Toc90983140)

[onVisualStylePropertyAttached 76](#_Toc90983141)

[C++ 76](#_Toc90983142)

[Xml 76](#_Toc90983143)

[onVisualStylePropertyDetached 77](#_Toc90983144)

[C++ 77](#_Toc90983145)

[Xml 77](#_Toc90983146)

[onLayerAdded 77](#_Toc90983147)

[C++ 77](#_Toc90983148)

[Xml 78](#_Toc90983149)

[onLayerDeleted 78](#_Toc90983150)

[C++ 78](#_Toc90983151)

[Xml 78](#_Toc90983152)

[onLayerVisibilityChanged 78](#_Toc90983153)

[C++ 78](#_Toc90983154)

[Xml 78](#_Toc90983155)

[onLayerFadingChanged 78](#_Toc90983156)

[C++ 79](#_Toc90983157)

[Xml 79](#_Toc90983158)

[onLayerViewportPropsOverride 79](#_Toc90983159)

[C++ 79](#_Toc90983160)

[Xml 79](#_Toc90983161)

[onLayerPropertyAttached 79](#_Toc90983162)

[C++ 80](#_Toc90983163)

[Xml 80](#_Toc90983164)

[onLayerPropertyDetached 80](#_Toc90983165)

[C++ 81](#_Toc90983166)

[Xml 81](#_Toc90983167)

[onGroupAdded 81](#_Toc90983168)

[C++ 81](#_Toc90983169)

[Xml 81](#_Toc90983170)

[onGroupDeleted 81](#_Toc90983171)

[C++ 82](#_Toc90983172)

[Xml 82](#_Toc90983173)

[onGroupMetafilesAttached 82](#_Toc90983174)

[C++ 82](#_Toc90983175)

[Xml 82](#_Toc90983176)

[onGroupMetafilesDetached 82](#_Toc90983177)

[C++ 82](#_Toc90983178)

[Xml 82](#_Toc90983179)

[Obsolete: onGroupMetafilesDeleted 83](#_Toc90983180)

[C++ 83](#_Toc90983181)

[Xml 83](#_Toc90983182)

[onGroupMetafilesVisibilityChanged 83](#_Toc90983183)

[C++ 83](#_Toc90983184)

[Xml 83](#_Toc90983185)

[onGroupMetafilesHighlightingChanged 83](#_Toc90983186)

[C++ 84](#_Toc90983187)

[Xml 84](#_Toc90983188)

[onGroupMetafilesFadingChanged 84](#_Toc90983189)

[C++ 84](#_Toc90983190)

[Xml 84](#_Toc90983191)

[onGroupPropertyAttached 84](#_Toc90983192)

[C++ 84](#_Toc90983193)

[Xml 85](#_Toc90983194)

[onGroupPropertyDetached 85](#_Toc90983195)

[C++ 85](#_Toc90983196)

[Xml 86](#_Toc90983197)

[onExtensionObjectAdded 86](#_Toc90983198)

[C++ 86](#_Toc90983199)

[Xml 86](#_Toc90983200)

[onExtensionObjectDeleted 86](#_Toc90983201)

[C++ 87](#_Toc90983202)

[Xml 87](#_Toc90983203)

[onExtensionObjectSpecificData 87](#_Toc90983204)

[C++ 87](#_Toc90983205)

[Xml 87](#_Toc90983206)

[onExtensionObjectVisibilityChanged 88](#_Toc90983207)

[C++ 88](#_Toc90983208)

[Xml 88](#_Toc90983209)

[onExtensionObjectPropertyAttached 88](#_Toc90983210)

[C++ 88](#_Toc90983211)

[Xml 89](#_Toc90983212)

[onExtensionObjectPropertyDetached 89](#_Toc90983213)

[C++ 89](#_Toc90983214)

[Xml 90](#_Toc90983215)

[onBlockAdded 90](#_Toc90983216)

[C++ 90](#_Toc90983217)

[Xml 90](#_Toc90983218)

[onBlockModified 90](#_Toc90983219)

[C++ 91](#_Toc90983220)

[Xml 91](#_Toc90983221)

[onBlockDeleted 91](#_Toc90983222)

[C++ 91](#_Toc90983223)

[Xml 91](#_Toc90983224)

[onBlockPropertyAttached 91](#_Toc90983225)

[C++ 92](#_Toc90983226)

[Xml 92](#_Toc90983227)

[onBlockPropertyDetached 92](#_Toc90983228)

[C++ 93](#_Toc90983229)

[Xml 93](#_Toc90983230)

[Obsolete: onVertexShader 93](#_Toc90983231)

[C++ 93](#_Toc90983232)

[Xml 93](#_Toc90983233)

[Obsolete: onFragmentShader 94](#_Toc90983234)

[C++ 94](#_Toc90983235)

[Xml 94](#_Toc90983236)

[Obsolete: onShaderProgram 94](#_Toc90983237)

[C++ 94](#_Toc90983238)

[Xml 95](#_Toc90983239)

[Metafile graphics stream 96](#_Toc90983240)

[C++ - binary graphics stream representation 96](#_Toc90983241)

[Binary data array types 97](#_Toc90983242)

[Vertex array 97](#_Toc90983243)

[Normals array 97](#_Toc90983244)

[Colors array 97](#_Toc90983245)

[Texture coordinates array 97](#_Toc90983246)

[Depths array 97](#_Toc90983247)

[Indexes array 97](#_Toc90983248)

[Markers array 98](#_Toc90983249)

[0 – Empty record 98](#_Toc90983250)

[1 – EnableOpt record 98](#_Toc90983251)

[2 – DisableOpt record 98](#_Toc90983252)

[3 – Color record 98](#_Toc90983253)

[4 – EnableArray record 99](#_Toc90983254)

[5 – DisableArray record 99](#_Toc90983255)

[6 – DrawArrays record 100](#_Toc90983256)

[7 – DrawElements record 100](#_Toc90983257)

[8 – CullFace record 101](#_Toc90983258)

[9 – LStipple record 101](#_Toc90983259)

[10 – PStipple record 101](#_Toc90983260)

[11 – VPoint record 101](#_Toc90983261)

[12 – VLine record 102](#_Toc90983262)

[13 – IPoint record 102](#_Toc90983263)

[14 – ILine record 102](#_Toc90983264)

[15 – HLRStencil record 102](#_Toc90983265)

[16 – EnableShading record 103](#_Toc90983266)

[17 – DisableShading record 103](#_Toc90983267)

[18 – Material record 103](#_Toc90983268)

[19 – UserEntry record 104](#_Toc90983269)

[20 – InitTexture record 104](#_Toc90983270)

[21 – UninitTexture record 104](#_Toc90983271)

[22 – SelectionMarker record 104](#_Toc90983272)

[23 – EnableMarkerArray record 104](#_Toc90983273)

[24 – DisableMarkerArray record 105](#_Toc90983274)

[25 – VisibilityFlags record 105](#_Toc90983275)

[26 – Lineweight record 105](#_Toc90983276)

[27 – Linestyle record 106](#_Toc90983277)

[28 – Program record 106](#_Toc90983278)

[29 – TtfText record 106](#_Toc90983279)

[30 – PushMatrix record 107](#_Toc90983280)

[31 – PopMatrix record 107](#_Toc90983281)

[32 – Metafile record 107](#_Toc90983282)

[33 – GeomMarker record 107](#_Toc90983283)

[34 – VisualStyle record 108](#_Toc90983284)

[35 – MetafileMarker record 108](#_Toc90983285)

[36 – ExtensionObject record 108](#_Toc90983286)

[37 – SelectionStyle record 108](#_Toc90983287)

[38 – DPoint record 109](#_Toc90983288)

[39 – DLine record 109](#_Toc90983289)

[40 – Block record 109](#_Toc90983290)

[Xml graphics stream representation 110](#_Toc90983291)

[Metafile data array types 111](#_Toc90983292)

[Vertex array 111](#_Toc90983293)

[Normals array 112](#_Toc90983294)

[Colors array 112](#_Toc90983295)

[Texture coordinates array 112](#_Toc90983296)

[Depths array 113](#_Toc90983297)

[Indexes array 113](#_Toc90983298)

[Markers array 113](#_Toc90983299)

[EnableOpt record 114](#_Toc90983300)

[DisableOpt record 114](#_Toc90983301)

[Color record 114](#_Toc90983302)

[EnableArray record 114](#_Toc90983303)

[DisableArray record 115](#_Toc90983304)

[DrawArrays record 115](#_Toc90983305)

[DrawElements record 116](#_Toc90983306)

[CullFace record 116](#_Toc90983307)

[LStipple record 116](#_Toc90983308)

[PStipple record 117](#_Toc90983309)

[VPoint record 117](#_Toc90983310)

[VLine record 117](#_Toc90983311)

[IPoint record 117](#_Toc90983312)

[ILine record 117](#_Toc90983313)

[HLRStencil record 118](#_Toc90983314)

[EnableShading record 118](#_Toc90983315)

[DisableShading record 118](#_Toc90983316)

[Material record 118](#_Toc90983317)

[UserEntry record 119](#_Toc90983318)

[InitTexture record 119](#_Toc90983319)

[UninitTexture record 119](#_Toc90983320)

[SelectionMarker record 119](#_Toc90983321)

[EnableMarkerArray record 119](#_Toc90983322)

[DisableMarkerArray record 120](#_Toc90983323)

[VisibilityFlags record 120](#_Toc90983324)

[Lineweight record 120](#_Toc90983325)

[Linestyle record 121](#_Toc90983326)

[Program record 121](#_Toc90983327)

[TtfText record 121](#_Toc90983328)

[PushMatrix record 122](#_Toc90983329)

[PopMatrix record 122](#_Toc90983330)

[Metafile record 122](#_Toc90983331)

[GeomMarker record 123](#_Toc90983332)

[VisualStyle record 123](#_Toc90983333)

[MetafileMarker record 123](#_Toc90983334)

[ExtensionObject record 123](#_Toc90983335)

[SelectionStyle record 124](#_Toc90983336)

[DPoint record 124](#_Toc90983337)

[DLine record 124](#_Toc90983338)

[Block record 124](#_Toc90983339)

[Remarks 125](#_Toc90983340)

## 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.
* Lights – represents light sources.
* State branches – represents 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.
* Groups – represent groups of metafiles.
* Extension objects – represent objects, which is implemented by external modules.
* Blocks – represent simple nested display lists.

## OdTrVisRendition interface methods

### onDeviceUpdateBegin/onDeviceUpdateEnd

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

**virtual** **void** onDeviceUpdateEnd(OdTrVisRenderClient \*pDevice, **bool** bRender) = 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, so their behavior depends from additional bRender argument.

#### 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.

### isWithinDeviceUpdate

**virtual** **bool** isWithinDeviceUpdate() **const** = 0;

#### C++

This method should return true in case if it is called between onDeviceUpdateBegin/ onDeviceUpdateEnd scopes.

#### Xml

Not related to Xml protocol.

### onFrameSwitchInfo

**virtual** **void** onFrameSwitchInfo(OdUInt32 nInterval) = 0;

#### C++

This method will be called by vectorizer on each frame change. Interval value contains time in milliseconds which is passed between previous frame completion and next frame beginning.

#### Xml

Example:

<FrameSwitchInfo>

<Interval>1200</Interval>

</FrameSwitchInfo>

Interval tag contains integer value which represents time in milliseconds which is passed between previous frame completion and next frame beginning.

### 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::kSupportGPUSelection – set if current OdTrVisRendition supports GPU-optimized geometry selection;
* 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.

### onPartialUpdateCheckLimit

**virtual** **bool** onPartialUpdateCheckLimit(OdTrVisViewportId viewportId, OdTrVisOverlayId overlayId) = 0;

#### C++

This method called by vectorizer to check does current combination of viewport and overlay supports partial update queries.

#### Xml

Not support.

### 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::kDisplayCodeFade – informs renderer that fading must be applied for upcoming metafiles. Next DisplaId contains fading slot number: 1 – locked layer fading, 2 – external reference fading;
* OdTrVisRendition::kDisplayCodeUnhighlight – informs renderer that fading must not be applied for upcoming metafiles. Next DisplaId contains fading slot number: 1 – locked layer fading, 2 – external reference fading;
* OdTrVisRendition::kDisplayCodeSelMarkers – sets current highlighting state branch directly from a display list. Next DisplayId after this code represents pointer onto OdGsStateBranch data type;
* OdTrVisRendition::kDisplayCodeVsMarkers – sets current visibility state branch directly from a display list. Next DisplayId after this code represents pointer onto OdGsStateBranch data type;
* OdTrVisRendition::kDisplayCodeDrawOrder – provides drawing order for upcoming metafiles. Next DisplayId after this code contains drawing order value as ‘double’ data type. Special OdTrVisRendition:: kDisplayMultiDrawOrderMarker value in ‘double’ data type 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;
* OdTrVisRendition::kDisplayCodeDepthIval – sets transformation for drawing order values. Next two DisplayId’s after this code represent scale and translation transformation components. In case if both values set to zero – this is means that renderer should disable drawing order transformation to default;
* OdTrVisRendition::kDisplayCodeExploded – informs renderer that metafiles which will come after this code represent union streams which is exploded for some reason. This is means that renderer must take into account metafile markers and check that upcoming part of graphics is visible, must be highlighted and etc.;
* OdTrVisRendition::kDisplayCodeNonExploded – disables exploded union streams rendering mode, so after this code renderer can process upcoming metafiles normally;
* OdTrVisRendition::kDisplayCodeSelStyle – sets selection style for upcoming highlighted geometry. Next DisplayId after this code contains integer, which specifies selection style index;
* OdTrVisRendition::kDisplayCodeSelectionID – sets special entity ID for GPU selection process. Next DisplayId after this code represents selection ID.

#### Xml

Not support.

### queryPartialUpdateDirectInvocation

**virtual** **bool** queryPartialUpdateDirectInvocation(OdTrVisViewportId viewportId, OdTrVisOverlayId overlayId , OdTrVisDirectRenderPartialUpdateInvocationPtr &pInvocationPtr) = 0;

#### C++

This method called by vectorizer to check does current combination of viewport and overlay supports direct partial update query. In case if direct partial update queries support by this OdTrVisRendition, this method returns invocation object, which can be used by vectorizer for direct partial update queries.

#### Xml

Not support.

### onPartialUpdateDirectInvocation

**virtual** **void** onPartialUpdateDirectInvocation(**const** OdTrVisDirectRenderPartialUpdateInvocation \*pInvocation, OdTrVisRenderDataDirectAccessor \*pAccessor) = 0;

#### C++

This method can be used by vectorizer direct partial update queries. It is invoke invocation object, returned by *OdTrVisRendition::queryPartialUpdateDirectInvocation* method and OdTrVisRenderDataDirectAccessor, provided by vectorizer.

#### 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 related to Xml protocol.

### getSelectedObject

**virtual** **void** getSelectedObject(OdTrVisViewportId id, OdUInt32 x, OdUInt32 y, OdArray<OdTrVisId> &outIDs, OdTrVisGsMarker \*outMarker = NULL) = 0;

#### C++

Vectorizer can call this method to query object Id on given screen surface position.

#### Xml

Not related to Xml protocol.

### 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.
* OdTrVisRendition::kMtComplex – vectorizer must use one or more mutexes, encoded inside returned value. Each mutex encoded as DataType and stored using 6 bits.

#### 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(OdTrVisViewportId viewportId, OdTrVisOverlayId overlayId, **const** OdTrVisDisplayId \*pDisplayId, OdUInt32 nPathLength, **const** OdTrVisMetafileContainer \*\*pStream) = 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. List of DisplayID’s typically contain single metafile, but for nested block reference metafile path can contain entire set of block reference metafiles, where last ID represents final object. After completion of selection process onto queried metafile stream, vectorizer calls OdTrVisRendition::unlockMetafileStream() method (see below).

#### Xml

Not related to Xml protocol.

### unlockMetafileStream

**virtual** **bool** unlockMetafileStream(**const** OdTrVisMetafileContainer \*pStream) = 0;

#### C++

After completion of selection process onto queried metafile stream (see OdTrVisRendition::queryMetafileStream() method description), vectorizer calls unlockMetafileStream() method to free resources, which is allocated to hold this metafile stream alive.

#### Xml

Not related to Xml protocol.

### queryBlockMetafilesList

**virtual** **const** OdTrVisDisplayId \*queryBlockMetafilesList(OdTrVisBlockId blockId, OdUInt32 &numEntries) = 0;

#### C++

In case if client supports returning of exist block metafile lists, this method can be used by vectorizer to query them from internal client container. This method will return null pointer in case if it doesn’t support block metafile list queries or requested block doesn’t registered previously.

#### Xml

Not related to Xml protocol.

### queryExtensionObject

**virtual** OdTrVisExtensionObjectPtr queryExtensionObject(OdTrVisExtensionId extId) = 0;

#### C++

In case if client supports returning of exist extension objects, this method can be used by vectorizer to query them from internal client container. This method will return null smart pointer in case if it doesn’t support extension object queries or requested extension object doesn’t registered previously.

#### Xml

Not related to Xml protocol.

### queryProgramId

**virtual** OdTrVisProgramId queryProgramId(OdGsView::RenderMode mode, OdTrVisVisualStyleId vsId = kTrVisNegativeId, OdTrVisGeomType geomType = OdTrVisGeomType\_Default, OdUInt16 attrFlags = 0, OdUInt16 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 must 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.
* kCubeMapping – set to true if client support cube textures.
* 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\_nMaxTextures [OdUInt32] – specifies maximum number of textures, which can be applied simultaneously during rendering operation.
* m\_nMaxLights [OdUInt32] – specifies maximum number of lights, which is possible to be rendered.

#### 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.

### onSurfaceInvalidated

**virtual** **void** onSurfaceInvalidated() = 0;

#### C++

This method will be called by vectorizer in case if rendering surface should be completely invalidated.

#### Xml

Example:

<SurfaceInvalidated>

</SurfaceInvalidated>

### 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.
* OdTrVisOverlayDef::kOBGenShadows – enable shadows generation for this overlay graphics data.
* OdTrVisOverlayDef::kOBUseShadows – enable shadows applying for this overlay graphics data.
* 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.
* 7 – enable shadows generation for this overlay graphics data.
* 8 – enable shadows applying for this overlay graphics data.
* 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.

### onOverlayPropertyAttached

**virtual** **void** onOverlayPropertyAttached(OdTrVisOverlayId overlayId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional overlay data, like overlay name, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this overlay object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this overlay object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this overlay object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this overlay object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this overlay object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this overlay object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this overlay object;
* OdTrVisPropertyDef::kNameProperty – optional overlay object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this overlay is a persistent object, which couldn’t be erased.

#### Xml

Example:

<OverlayPropertyAttached>

<OverlayID>3</OverlayID>

<PropType>7</PropType>

<PropVal>

<Text>Overlay Name</Text>

</PropVal>

</OverlayPropertyAttached>

* OverlayID [string] – identifier of overlay buffer to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional state branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onOverlayPropertyDetached

**virtual** **void** onOverlayPropertyDetached(OdTrVisOverlayId overlayId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached overlay data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this overlay object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this overlay object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this overlay object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this overlay object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this overlay object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this overlay object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this overlay object;
* OdTrVisPropertyDef::kNameProperty – optional overlay object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this overlay is a persistent object, which couldn’t be erased.

#### Xml

Example:

<OverlayPropertyDetached>

<OverlayID>3</OverlayID>

<PropType>7</PropType>

</OverlayPropertyDetached>

* OverlayID [string] – identifier of overlay buffer to detach additional data.
* PropType [integer] – identifies property type to detach.

### onViewportAdded

**virtual** **void** onViewportAdded(OdTrVisViewportId viewportId, **const** OdTrVisViewportDef &pDef) = 0;

#### C++

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

OdTrVisViewportDef member’s description:

* m\_vpFlags [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.
* OdTrVisViewportDef::kVpVisible – viewport is visible onto screen.
* OdTrVisViewportDef::kVpAntiAliasing – viewport anti-aliasing supported.
* m\_overallViewportId [OdTrVisViewportId] – identifier of overall paper space viewport in case if this is paper space viewport which shows model space contents.

#### Xml

Example:

<ViewportAdded>

<ViewportID>145425480</ViewportID>

<PSModelViewport>1</PSModelViewport>

<OverallVpId>143255432</OverallVpId>

<PSOverallViewport>0</PSOverallViewport>

<PSHelperViewport>0</PSHelperViewport>

<ViewportVisible>1</ViewportVisible>

<ViewportAntiAliasing>1</ViewportAntiAliasing>

</ViewportAdded>

* ViewportID [string] – identifier of newly created viewport.
* 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.
* ViewportVisible [boolean] – set to 1 if this viewport is visible onto screen.
* ViewportAntiAliasing [boolean] – set to 1 if this viewport supports anti-aliasing.

### onViewportInserted

**virtual** **void** onViewportInserted(OdTrVisViewportId viewportId, **int** nIndex, **const** OdTrVisViewportDef &pDef) = 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.

OdTrVisViewportDef member’s description:

* m\_vpFlags [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.
* OdTrVisViewportDef::kVpVisible – viewport is visible onto screen.
* OdTrVisViewportDef::kVpAntiAliasing – viewport anti-aliasing supported.
* m\_overallViewportId [OdTrVisViewportId] – identifier of overall paper space viewport in case if this is paper space viewport which shows model space contents.

#### Xml

Example:

<ViewportInserted>

<ViewportID>145425480</ViewportID>

<Index>1</Index>

<PSModelViewport>1</PSModelViewport>

<OverallVpId>143255432</OverallVpId>

<PSOverallViewport>0</PSOverallViewport>

<PSHelperViewport>0</PSHelperViewport>

<ViewportVisible>1</ViewportVisible>

<ViewportAntiAliasing>1</ViewportAntiAliasing>

</ViewportInserted>

* ViewportID [string] – identifier of newly created viewport.
* Index [integer] – position at which newly created viewport must be added into viewports list.
* 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.
* ViewportVisible [boolean] – set to 1 if this viewport is visible onto screen.
* ViewportAntiAliasing [boolean] – set to 1 if this viewport supports anti-aliasing.

### 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.

### onViewportVisibilityChanged

**virtual** **void** onViewportVisibilityChanged(OdTrVisViewportId viewportId, **bool** bVisibility) = 0;

#### C++

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

#### Xml

Example:

<ViewportVisibilityChanged>

<ViewportID>24A</ViewportID>

<Visibility>0</Visibility>

</ViewportVisibilityChanged>

* ViewportID [string] – identifier of viewport to change visibility state.
* Visibility [boolean] – new visibility state.

### onViewportViewParamsModified

**virtual** **void** onViewportViewParamsModified(OdTrVisViewportId viewportId, **const** OdTrVisViewParamsDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if viewport viewing parameters (positioning, zooming, clipping and etc.) are to be changed.

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:

<ViewportViewParamsModified>

<ViewportID>24A</ViewportID>

<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>

</ViewportViewParamsModified>

* ViewportID [string] – identifier of viewport to modify viewing paramaters.
* 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.

### onViewportLineStylesModified

**virtual** **void** onViewportLineStylesModified(OdTrVisViewportId viewportId, **const** OdTrVisViewportLineStyleDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if viewport line styles configuration parameters are to be changed.

OdTrVisViewportLineStyleDef member’s description:

* 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).

#### Xml

Example:

<ViewportLineStylesModified>

<ViewportID>24A</ViewportID>

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

</ViewportLineStylesModified>

* ViewportID [string] – identifier of viewport to change line styles configuration parameters.
* 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).

### onViewportFadingModified

**virtual** **void** onViewportFadingModified(OdTrVisViewportId viewportId, **const** OdTrVisViewportFadingDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if viewport fading configuration parameters are to be changed.

OdTrVisViewportFadingDef member’s description:

* 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).

#### Xml

Example:

<ViewportFadingModified>

<ViewportID>241</ViewportID>

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

</ViewportFadingModified>

* ViewportID [string] – identifier of viewport to change fading configuration parameters.
* 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).

### onViewportLineweightsModified

**virtual** **void** onViewportLineweightsModified(OdTrVisViewportId viewportId, **const** OdTrVisViewportLineweightsDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if viewport lineweight parameters are to be changed.

OdTrVisViewportLineweightsDef member’s description:

* m\_lineweightsEnum [OdUInt8Array] – array of lineweights in pixel for lineweight indexes in model space.
* m\_lineweightsCoef [double] – lineweights multiplier.

#### Xml

Example:

<ViewportLineweightsModified>

<ViewportID>241</ViewportID>

<NLineweightEnum>2</NLineweightEnum>

<LineweightCoef>0</LineweightCoef>

<LineweightEnum>

<NPixels>1<NPixels>

<NPixels>2<NPixels>

</LineweightEnum>

</ViewportLineweightsModified>

* ViewportID [string] – identifier of viewport to change lineweight parameters.
* 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.

### onViewportShadingModified

**virtual** **void** onViewportShadingModified(OdTrVisViewportId viewportId, **const** OdTrVisViewportShadingDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if viewport shading parameters (rendering mode, visual style and etc.) are to be changed.

OdTrVisViewportShadingDef member’s description:

* m\_rmShader [OdTrVisProgramId] – identifier of viewport basic shader program.
* m\_visualStyle [OdTrVisVisualStyleId] – identifier of assigned viewport visual style.
* m\_renderMode [OdTrVisRenderMode] – represents viewport rendering mode in visualizer renderer format.
* m\_faceFillColor [ODCOLORREF] – specifies face filling color for Hidden Line render mode.

#### Xml

Example:

<ViewportShadingModified>

<ViewportID>241</ViewportID>

<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>

</ViewportShadingModified>

* ViewportID [string] – identifier of viewport to change shading parameters.
* 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).

### onViewportClipModified

**virtual** **void** onViewportClipModified(OdTrVisViewportId viewportId, **const** OdTrVisViewportClipDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if viewport clipping parameters (positioning onto screen, clipping contour and viewport border) are to be changed.

OdTrVisViewportClipDef member’s description:

* m\_flags [OdUInt16] – set of bit flags which enables some parts of viewport clipping parameters:
* OdTrVisViewportClipDef::kEnableNrcClip – enabled non-rectangular viewport clipping contour.
* OdTrVisViewportClipDef::kEnableBorder – enables viewport border visibility.
* m\_screenRect [OdGsDCRectDouble] – specifies viewport rectangle on output surface.
* m\_nrcClip [OdTrVisViewportRegionDef] – non-rectangular viewport clipping contour. OdTrVisViewportRegionDef structure member’s description:
  + m\_counts [OdIntArray] – in case if non-rectangular viewport clipping enabled this array contains number of vertices for each clipping boundary loop.
  + m\_points [OdFloatArray] – array of float’s which represents non-rectangular viewport clipping vertices. Pair of float’s per each vertex.
* m\_border [OdTrVisViewportBorderDef] – viewport border parameters. OdTrVisViewportBorderDef structure member’s desction:
  + m\_color [ODCOLORREF] – specifies viewport border color.
  + m\_width [int] – specified viewport border lines width in pixels.
  + m\_region [OdTrVisViewportRegionDef] – viewport border region. OdTrVisViewportRegionDef structure member’s description:
    - m\_counts [OdIntArray] – in case if viewport border enabled this array contains number of vertices for each border loop.
    - m\_points [OdFloatArray] – array of float’s which represents viewport border vertices. Pair of float’s per each vertex.

#### Xml

Example:

<ViewportClipModified>

<ViewportID>241</ViewportID>

<ScreenRect>0, 1.000000, 1.000000, 0</ScreenRect>

<NrcClip>1</NrcClip>

<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>

<Border>0</Border>

<BorderColor>0, 0, 0, 255</BorderColor>

<BorderWidth>0</BorderWidth>

<NBorderCounts>1</NBorderCounts>

<NBorderPoints>5</NBorderPoints>

<BorderCounts>

<Count>5</Count>

</BorderCounts>

<BorderPoints>

<Point>5.68434e-17, 1.000000</Point>

<Point>1.000000, 1.000000</Point>

<Point>1.000000, 5.68434e-17</Point>

<Point>5.68434e-17, 5.68434e-17</Point>

<Point>5.68434e-17, 1.000000</Point>

</BorderPoints>

</ViewportClipModified>

* ViewportID [string] – identifier of viewport to change clipping parameters.
* ScreenRect [float \* 4] – viewport rectangle on output surface.
* NrcClip [boolean] – set to 1 in case if non-rectangular viewport clipping enabled or set to 0 elsewhere.
* 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.
* 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.

### onViewportPropertyAttached

**virtual** **void** onViewportPropertyAttached(OdTrVisViewportId viewportId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional viewport data, like viewport name, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this viewport object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this viewport object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this viewport object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this viewport object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this viewport object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this viewport object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this viewport object;
* OdTrVisPropertyDef::kNameProperty – optional viewport object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this viewport is a persistent object, which couldn’t be erased.

#### Xml

Example:

<ViewportPropertyAttached>

<ViewportID>34365</ViewportID>

<PropType>7</PropType>

<PropVal>

<Text>Viewport Name</Text>

</PropVal>

</ViewportPropertyAttached>

* ViewportID [string] – identifier of viewport object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional highlighting branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onViewportPropertyDetached

**virtual** **void** onViewportPropertyDetached(OdTrVisViewportId viewportId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached viewport data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this viewport object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this viewport object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this viewport object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this viewport object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this viewport object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this viewport object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this viewport object;
* OdTrVisPropertyDef::kNameProperty – optional viewport object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this viewport is a persistent object, which couldn’t be erased.

#### Xml

Example:

<ViewportPropertyDetached>

<ViewportID>34656</ViewportID>

<PropType>7</PropType>

</ViewportPropertyDetached>

* ViewportID [string] – identifier of viewport object to detach additional data.
* PropType [integer] – identifies property type to detach.

### onViewportBackgroundChanged

**virtual** **void** onViewportBackgroundChanged(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:

<ViewportBackgroundChanged>

<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>

</ViewportBackgroundChanged>

* 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.

### onViewportFogChanged

**virtual** **void** onViewportFogChanged(OdTrVisViewportId viewportId, **const** OdTrVisFogDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if settings of fog effect inside viewport are changed. Fog inside renderer presented as near and far distances (in world coordinates system) and near/far densities.

OdTrVisFogDef member’s description:

* m\_flags [OdUInt16] – set of binary flags, described by Flags enumeration:
  + kEnableFog – enable fog effect rendering.
  + kFogBackground – apply fog effect onto viewport background.
* m\_color [ODCOLORREF] – fog color, represented as RGBA color components.
* m\_fDistNear [double] – distance at which fog effect will be started. All geometry, which will come before this distance will be drawn without fog effect.
* m\_fDistFar [double] – distance at which fog will have maximum effect. All geometry, which will come after this distance will be also drawn with maximum fog effect.
* m\_fDensNear [double] – value in [0-1] range which represent fog density at nearest for density.
* m\_fDensFar [double] – valine in [0-1] range which represent maximal fog density.

#### Xml

Example:

<ViewportFogChanged>

<ViewportID>145423481</ViewportID>

<Enable>1</Enable>

<Background>1</Background>

<Color>224, 224, 224, 255</Color>

<Distance>100.0, 1000.0</Distance>

<Density>0.1, 0.9</Density>

</ViewportFogChanged>

* ViewportID [string] – identifier of viewport to update fog effect settings.
* Enable [boolean] – enables or disables fog effect.
* Background [boolean] – enables or disables fog effect rendering onto viewport background.
* Color [integer \* 4] – fog color as four R, G, B and A (0-255 range) integers.
* Distance [float \* 2] – near and far distances of fog effect. All geometry which is come before nearest distance will be drawn without fog effect applying. At nearest distance fog effect will have minimal density which is specified by nearest density. At far distance fog effect will have maximal density. All geometry which is come after far distance will be drawn with maximal fog density.
* Density [float \* 3] – near (minimal) and far (maximal) fog densities which will be applied onto near and far fog distances.

### onViewportSectioningChanged

**virtual** **void** onViewportSectioningChanged(OdTrVisViewportId viewportId, **const** OdTrVisSectionDef &pDef = OdTrVisSectionDef()) = 0;

#### C++

This method will be called by vectorizer to modify set of sectioning planes into viewport.

OdTrVisSectionDef member’s description:

* m\_flags [OdUInt16] – set of bit flags:
* OdTrVisSectionDef::kEnableFilling – enable filling of geometry sections.
* OdTrVisSectionDef::kEnableStippling – enable stippling pattern in case if geometry sections filling enabled.
* m\_planes [OdTrVisSectionPlaneArray] – array of section planes.
* m\_color [ODCOLORREF] – color of section filling.
* m\_stippleColor [ODCOLORREF] – color of stippling pattern.
* m\_stipplePattern [OdUInt8] – stippling pattern index (look for TD PlotStyle linetypes specification).

#### Xml

Example:

<ViewportSectioningChanged>

<ViewportID>2787365</ViewportID>

<NPlanes>1</NPlanes>

<PlanesList>

<Plane>

<Row0>0.0, 0.0, 0.0</Row0>

<Row1>0.0, 0.0, 1.0</Row1>

<Row2>0.0, 0.0, 0.0</Row2>

</Plane>

</PlanesList>

<EnableFilling>1</EnableFilling>

<Color>255, 0, 0, 255</Color>

<EnableStippling>1</EnableStippling>

<StipplePattern>2</StipplePattern>

<StipplingColor>0, 0, 255, 255</StipplingColor>

</ViewportSectioningChanged>

* ViewportID [string] – identifier of viewport to modify sectioning planes setting.
* NPlanes [integer] – number of section planes in planes list.
* PlanesList [tag] – set of <Plane> tags each of which specifies section plane setting to be added in viewport:
  + Row0 [float \* 3] – plane origin in world space.
  + Row1 [float \* 3] – plane normal in world space.
  + Row2 [float \* 3] – reserved vector.
* EnableFilling [boolean] – set to 1 if sectioned geometry caps filling enabled.
* Color [integer \* 4] – section filling color as four R, G, B and A (0-255 range) integers.
* EnableStippling [boolean] – enable stippling pattern in case if section plane filling enabled.
* StipplePattern [integer] – stippling pattern index (look for TD PlotStyle linetypes specification).
* StipplingColor [integer \* 4] – stippling pattern color.

### onViewportAntiAliasingChanged

**virtual** **void** onViewportAntiAliasingChanged(OdTrVisViewportId viewportId, OdUInt32 nAntiAliasing) = 0;

#### C++

This method will be called by vectorizer in case if viewport anti-aliasing state is to be changed.

#### Xml

Example:

<ViewportAntiAliasingChanged>

<ViewportID>24A</ViewportID>

<AntiAliasing>0</AntiAliasing>

</ViewportAntiAliasingChanged>

* ViewportID [string] – identifier of viewport to change anti-aliasing support state.
* AntiAliasing [OdUInt32] – new anti-aliasing state.

### onViewportSSAOChanged

**virtual** **void** onViewportSSAOChanged(OdTrVisViewportId viewportId, bool bEnabled) = 0;

#### C++

This method will be called by vectorizer in case if viewport SSAO state is to be changed.

#### Xml

Example:

<ViewportSSAOChanged>

<ViewportID>24A</ViewportID>

<SSAOEnabled>1</SSAOEnabled>

</ViewportSSAOChanged>

* ViewportID [string] – identifier of viewport to change SSAO state.
* SSAOEnabled [boolean] – new SSAO state.

### onViewportAmbientLightChanged

**virtual** **void** onViewportAmbientLightChanged(OdTrVisViewportId viewportId, **const** OdTrVisMaterialColor &ambientLight) = 0;

#### C++

This method will be called by vectorizer in case if vectorizer sent new ambient light color which could be added by renderer into shading program for material rendering. Ambient light color represented as OdTrVisMaterialColor structure – four separate color channels (red, gree, blue and alpha) as floating point values.

#### Xml

Example:

<ViewportAmbientLightChanged>

<ViewportID>34801098</ViewportID>

<AmbientLight>0.2, 0.2, 0.2, 1.0</AmbientLight>

</ViewportAmbientLightChanged>

* ViewportID [string] – identifier of viewport to modify ambient light setting.
* AmbientLight [float \* 4] – ambient environment color (RGBA color components in 0-1 range).

### onViewportDefaultLightsChanged

**virtual** **void** onViewportDefaultLightsChanged(OdTrVisViewportId viewportId, **bool** bDefaultLightsOn, **const** OdTrVisLightDef \*pLightsList, OdUInt32 nLightsListEntries) = 0;

#### C++

This method will be called by vectorizer in case if list of default lights changed for shaded rendering modes. 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\_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 [OdTrVisMaterialColor] – diffuse color of light (color components represented as four floating point values 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\_startAttenuationLimit [float] – distance from light source position where lighted range should be started (have sense only in case if attenuation limits usage enabled for this light source).
* m\_endAttenuationLimit [float] – distance from light source position where lighted range should be ended (have sense only in case if attenuation limits usage enabled for this light source).
* 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.
* m\_shadowMapSize [OdUInt16] – size of shadow map texture (have sense only in case if shadows usage enabled for this light source).
* m\_shadowSoftness [OdUInt16] – number of smoothing steps during shadow map application (have sense only in case if shadows usage enabled for this light source).
* m\_shadowSamples [OdUInt16] – number of shadow map samples which will be taken into account during shadow map application (have sense only in case if shadows usage enabled for this light source).
* m\_lightFlags [OdUInt16] – set of following bit flags:
  + OdTrVisLightDef::kLightEnabled – set if light source enabled.
  + OdTrVisLightDef::kLightUseLims – set if attenuation limits usage enabled.
  + OdTrVisLightDef::kLightShadows – set if shadows usage enabled.

#### Xml

Example:

<ViewportDefaultLightsChanged>

<ViewportID>478283480</ViewportID>

<DefaultLightsOn>0</DefaultLightsOn>

<NLights>1</NLights>

<LightsList>

<Light>

<Type>1</Type>

<On>1</On>

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

<Direction>0.0, 0.0, 1.0</Direction>

<Color>1.0, 1.0, 1.0, 1.0</Color>

<ConstantAttenuation>1.0</ConstantAttenuation>

<LinearAttenuation>0.0</LinearAttenuation>

<QuadraticAttenuation>0.0</QuadraticAttenuation>

<UseLimits>1</UseLimits>

<StartAttenuationLimit>1.0</StartAttenuationLimit>

<EndAttenuationLimit>10.0</EndAttenuationLimit>

<SpotCutoff>0.0</SpotCutoff>

<SpotExponent>0.0</SpotExponent>

<SpecularFactor>1.0</SpecularFactor>

<ShadowsEnabled>1</ShadowsEnabled>

<ShadowMapSize>2048</ShadowMapSize>

<ShadowSoftness>1</ShadowSoftness>

<ShadowSamples>16</ShadowSamples>

</Light>

</LightsList>

</ViewportDefaultLightsChanged>

* ViewportID [string] – identifier of viewport to modify default lights list.
* DefaultLightsOn [boolean] – informs renderer that default light sources must be taken into account during visualization or not.
* NLights [integer] – number of default lights in lights list.
* LightsList [tag] – set of <Light> tags each of which specifies default 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.
* On [boolean] – set to 1 if this light source is enabled or to 0 if this light source is disabled.
* Position [float \* 3] – light position in world space.
* Direction [float \* 3] – light direction in world space.
* Color [float \* 4] – light color (RGBA components in 0-1 range).
* ConstantAttenuation [float] – constant light attenuation component.
* LinearAttenuation [float] – linear light attenuation component.
* QuadraticAttenuation [float] – quadratic light attenuation component.
* UseLimits [boolean] – set to 1 if attenuation limits usage enabled or to 0 if attenuation limits usage disabled.
* StartAttenuationLimit [float] – distance from light source position where lighted range should be started (have sense only in case if attenuation limits usage enabled for this light source).
* EndAttenuationLimit [float] – distance from light source position where lighted range should be ended (have sense only in case if attenuation limits usage enabled for this light source).
* SpotCutoff [float] – light spot cutoff angle.
* SpotExponent [float] – light spot exponent factor.
* SpecularFactor [float] – multiplier for specular highlighting.
  + ShadowsEnabled [boolean] – set to 1 if shadows usage enabled or to 0 if shadows usage disabled.
* ShadowMapSize [integer] – size of shadow map texture (have sense only in case if shadows usage enabled for this light source).
* ShadowSoftness [integer] – number of smoothing steps during shadow map application (have sense only in case if shadows usage enabled for this light source).
* ShadowSamples [integer] – number of shadow map samples which will be taken into account during shadow map application (have sense only in case if shadows usage enabled for this light source).

### onViewportLightAttached

**virtual** **void** onViewportLightAttached(OdTrVisViewportId viewportId, OdTrVisLightId lightId, OdUInt32 nOverrides, **const** OdTrVisLightDef &pOverrides) = 0;

#### C++

This method will be called by vectorizer to attach exist (registered) scene light to viewport scene lights list. Each part of original light data can be overridden. This is list of possible overrides and members of OdTrVisLightDef structure which is influenced by this override flags:

* OdTrVisLightDef::kOverrideVisibility:
  + Invoke OdTrVisLightDef::kLightEnabled flag from m\_lightFlags [OdUInt16] member.
* OdTrVisLightDef::kOverrideOrientation:
  + 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).
* OdTrVisLightDef::kOverrideColor:
  + m\_color [OdTrVisMaterialColor] – diffuse color of light (color components represented as four floating point values in 0-1 range).
* OdTrVisLightDef::kOverrideAttenuation:
  + 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).
  + Invoke OdTrVisLightDef::kLightUseLims flag from m\_lightFlags [OdUInt16] member.
  + m\_startAttenuationLimit [float] – distance from light source position where lighted range should be started (have sense only in case if attenuation limits usage enabled for this light source).
  + m\_endAttenuationLimit [float] – distance from light source position where lighted range should be ended (have sense only in case if attenuation limits usage enabled for this light source).
* OdTrVisLightDef::kOverrideAngles:
  + m\_spotCutoff [float] – light spot cutoff angle (for spot light type).
  + m\_spotExponent [float] – light spot exponent factor (for spot light type).
* OdTrVisLightDef::kOverrideSpecularFactor:
  + m\_specularFactor [float] – multiplier for specular highlighting.
* OdTrVisLightDef::kOverrideShadow:
  + Invoke OdTrVisLightDef::kLightShadows flag from m\_lightFlags [OdUInt16] member.
  + m\_shadowMapSize [OdUInt16] – size of shadow map texture (have sense only in case if shadows usage enabled for this light source).
  + m\_shadowSoftness [OdUInt16] – number of smoothing steps during shadow map application (have sense only in case if shadows usage enabled for this light source).
  + m\_shadowSamples [OdUInt16] – number of shadow map samples which will be taken into account during shadow map application (have sense only in case if shadows usage enabled for this light source).

#### Xml

Example:

<ViewportLightAttached>

<ViewportID>7282046432</ViewportID>

<LightID>92837461</LightId>

<NOverrides>127</NOverrides>

<On>1</On>

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

<Direction>0.0, 0.0, 1.0</Direction>

<Color>1.0, 1.0, 1.0, 1.0</Color>

<ConstantAttenuation>1.0</ConstantAttenuation>

<LinearAttenuation>0.0</LinearAttenuation>

<QuadraticAttenuation>0.0</QuadraticAttenuation>

<UseLimits>1</UseLimits>

<StartAttenuationLimit>1.0</StartAttenuationLimit>

<EndAttenuationLimit>10.0</EndAttenuationLimit>

<SpotCutoff>0.0</SpotCutoff>

<SpotExponent>0.0</SpotExponent>

<SpecularFactor>1.0</SpecularFactor>

<ShadowsEnabled>1</ShadowsEnabled>

<ShadowMapSize>2048</ShadowMapSize>

<ShadowSoftness>1</ShadowSoftness>

<ShadowSamples>16</ShadowSamples>

</ViewportLightAttached>

* ViewportID [string] – identifier of viewport to attach exist scene light source.
* LightID [string] – identifier of exist scene light.
* NOverrides [integer] – set of bit flags (look for OdTrVisLightDef::kOverride… bit flags) which inform which parts of light data should be overridden for this viewport.
* On [boolean] – set to 1 if this light source is enabled or to 0 if this light source is disabled.
* Position [float \* 3] – light position in world space.
* Direction [float \* 3] – light direction in world space.
* Color [float \* 4] – light color (RGBA components in 0-1 range).
* ConstantAttenuation [float] – constant light attenuation component.
* LinearAttenuation [float] – linear light attenuation component.
* QuadraticAttenuation [float] – quadratic light attenuation component.
* UseLimits [boolean] – set to 1 if attenuation limits usage enabled or to 0 if attenuation limits usage disabled.
* StartAttenuationLimit [float] – distance from light source position where lighted range should be started (have sense only in case if attenuation limits usage enabled for this light source).
* EndAttenuationLimit [float] – distance from light source position where lighted range should be ended (have sense only in case if attenuation limits usage enabled for this light source).
* SpotCutoff [float] – light spot cutoff angle.
* SpotExponent [float] – light spot exponent factor.
* SpecularFactor [float] – multiplier for specular highlighting.
* ShadowsEnabled [boolean] – set to 1 if shadows usage enabled or to 0 if shadows usage disabled.
* ShadowMapSize [integer] – size of shadow map texture (have sense only in case if shadows usage enabled for this light source).
* ShadowSoftness [integer] – number of smoothing steps during shadow map application (have sense only in case if shadows usage enabled for this light source).
* ShadowSamples [integer] – number of shadow map samples which will be taken into account during shadow map application (have sense only in case if shadows usage enabled for this light source).

### onViewportLightDetached

**virtual** **void** onViewportLightDetached(OdTrVisViewportId viewportId, OdTrVisLightId lightId) = 0;

#### C++

This method will be called by vectorizer to detach exist (registered) scene light from viewport scene lights list.

#### Xml

Example:

<ViewportLightDetached>

<ViewportID>7282046432</ViewportID>

<LightID>92837461</LightId>

</ViewportLightDetached>

* ViewportID [string] – identifier of viewport to detach exist scene light source.
* LightID [string] – identifier of scene light.

### 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.

### onOverlayExtentsOverride

**virtual** **void** onOverlayExtentsOverride(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:

<OverlayExtentsOverride>

<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>

</OverlayExtentsOverride>

* 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.

### 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

### 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::kMfCastShadows – set if metafile contains geometry which can cast shadows;
* OdTrVisMetafileContainer::kMfRcivShadows – set if metafile contains geometry which can receive shadows.
* OdTrVisMetafileContainer::kMfUnited – set if metafile represents union of smaller metafile streams.
* 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>

<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.
* 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).
* ShadowsCast [boolean] – set to 1 if metafile contains geometry which can cast shadows.
* ShadowsReceive [boolean] – set to 1 if metafile contains geometry which can receive shadows.
* United [boolean] – set to 1 if metafile represents union of smaller metafile streams.
* 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.

### onMetafilePropertyAttached

**virtual** **void** onMetafilePropertyAttached(OdTrVisMetafileId metafileId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional metafile data, like highlighting branch, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this metafile object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this metafile object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this metafile object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this metafile object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this metafile object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this metafile object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this metafile object;
* OdTrVisPropertyDef::kNameProperty – optional metafile object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this metafile is a persistent object, which couldn’t be erased.

#### Xml

Example:

<MetafilePropertyAttached>

<MetafileID>17923746</MetafileID>

<PropType>3</PropType>

<PropVal>

<StateBranchID>89838376</StateBranchID>

</PropVal>

</MetafilePropertyAttached>

* MetafileID [string] – identifier of metafile object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional highlighting branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onMetafilePropertyDetached

**virtual** **void** onMetafilePropertyDetached(OdTrVisMetafileId metafileId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached metafile data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this metafile object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this metafile object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this metafile object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this metafile object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this metafile object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this metafile object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this metafile object;
* OdTrVisPropertyDef::kNameProperty – optional metafile object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this metafile is a persistent object, which couldn’t be erased.

#### Xml

Example:

<MetafilePropertyDetached>

<MetafileID>17923746</MetafileID>

<PropType>3</PropType>

</MetafilePropertyDetached>

* MetafileID [string] – identifier of metafile object to detach additional data.
* PropType [integer] – identifies property type to detach.

### onLightAdded

**virtual** **void** onLightAdded(OdTrVisLightId lightId, **const** OdTrVisLightDef &pDef) = 0;

#### C++

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

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\_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 [OdTrVisMaterialColor] – diffuse color of light (color components represented as four floating point values 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\_startAttenuationLimit [float] – distance from light source position where lighted range should be started (have sense only in case if attenuation limits usage enabled for this light source).
* m\_endAttenuationLimit [float] – distance from light source position where lighted range should be ended (have sense only in case if attenuation limits usage enabled for this light source).
* 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.
* m\_shadowMapSize [OdUInt16] – size of shadow map texture (have sense only in case if shadows usage enabled for this light source).
* m\_shadowSoftness [OdUInt16] – number of smoothing steps during shadow map application (have sense only in case if shadows usage enabled for this light source).
* m\_shadowSamples [OdUInt16] – number of shadow map samples which will be taken into account during shadow map application (have sense only in case if shadows usage enabled for this light source).
* m\_lightFlags [OdUInt16] – set of following bit flags:
  + OdTrVisLightDef::kLightEnabled – set if light source enabled.
  + OdTrVisLightDef::kLightUseLims – set if attenuation limits usage enabled.
  + OdTrVisLightDef::kLightShadows – set if shadows usage enabled.

#### Xml

Example:

<LightAdded>

<LightID>908002231</LightID>

<Type>1</Type>

<On>1</On>

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

<Direction>0.0, 0.0, 1.0</Direction>

<Color>1.0, 1.0, 1.0, 1.0</Color>

<ConstantAttenuation>1.0</ConstantAttenuation>

<LinearAttenuation>0.0</LinearAttenuation>

<QuadraticAttenuation>0.0</QuadraticAttenuation>

<UseLimits>1</UseLimits>

<StartAttenuationLimit>1.0</StartAttenuationLimit>

<EndAttenuationLimit>10.0</EndAttenuationLimit>

<SpotCutoff>0.0</SpotCutoff>

<SpotExponent>0.0</SpotExponent>

<SpecularFactor>1.0</SpecularFactor>

<ShadowsEnabled>1</ShadowsEnabled>

<ShadowMapSize>2048</ShadowMapSize>

<ShadowSoftness>1</ShadowSoftness>

<ShadowSamples>16</ShadowSamples>

</LightAdded>

* LightID [string] – identifier of scene light to add.
* Type [integer] – set to 0 for distant light type, set to 1 for point light type, or set to 2 for spot light type.
* On [boolean] – set to 1 if this light source is enabled or to 0 if this light source is disabled.
* Position [float \* 3] – light position in world space.
* Direction [float \* 3] – light direction in world space.
* Color [float \* 4] – light color (RGBA components in 0-1 range).
* ConstantAttenuation [float] – constant light attenuation component.
* LinearAttenuation [float] – linear light attenuation component.
* QuadraticAttenuation [float] – quadratic light attenuation component.
* UseLimits [boolean] – set to 1 if attenuation limits usage enabled or to 0 if attenuation limits usage disabled.
* StartAttenuationLimit [float] – distance from light source position where lighted range should be started (have sense only in case if attenuation limits usage enabled for this light source).
* EndAttenuationLimit [float] – distance from light source position where lighted range should be ended (have sense only in case if attenuation limits usage enabled for this light source).
* SpotCutoff [float] – light spot cutoff angle.
* SpotExponent [float] – light spot exponent factor.
* SpecularFactor [float] – multiplier for specular highlighting.
* ShadowsEnabled [boolean] – set to 1 if shadows usage enabled or to 0 if shadows usage disabled.
* ShadowMapSize [integer] – size of shadow map texture (have sense only in case if shadows usage enabled for this light source).
* ShadowSoftness [integer] – number of smoothing steps during shadow map application (have sense only in case if shadows usage enabled for this light source).
* ShadowSamples [integer] – number of shadow map samples which will be taken into account during shadow map application (have sense only in case if shadows usage enabled for this light source).

### onLightModified

**virtual** **void** onLightModified(OdTrVisLightId lightId, OdUInt32 nKindOfMod, **const** OdTrVisLightDef &pDef) = 0;

#### C++

This method will be called by vectorizer in case if exist light source is modified into graphics scene. This is list of possible changes (nKindOfMod argument) and members of OdTrVisLightDef structure which is influenced by this modification flags:

* OdTrVisLightDef::kOverrideVisibility:
  + Invoke OdTrVisLightDef::kLightEnabled flag from m\_lightFlags [OdUInt16] member.
* OdTrVisLightDef::kOverrideOrientation:
  + 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).
* OdTrVisLightDef::kOverrideColor:
  + m\_color [OdTrVisMaterialColor] – diffuse color of light (color components represented as four floating point values in 0-1 range).
* OdTrVisLightDef::kOverrideAttenuation:
  + 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).
  + Invoke OdTrVisLightDef::kLightUseLims flag from m\_lightFlags [OdUInt16] member.
  + m\_startAttenuationLimit [float] – distance from light source position where lighted range should be started (have sense only in case if attenuation limits usage enabled for this light source).
  + m\_endAttenuationLimit [float] – distance from light source position where lighted range should be ended (have sense only in case if attenuation limits usage enabled for this light source).
* OdTrVisLightDef::kOverrideAngles:
  + m\_spotCutoff [float] – light spot cutoff angle (for spot light type).
  + m\_spotExponent [float] – light spot exponent factor (for spot light type).
* OdTrVisLightDef::kOverrideSpecularFactor:
  + m\_specularFactor [float] – multiplier for specular highlighting.
* OdTrVisLightDef::kOverrideShadow:
  + Invoke OdTrVisLightDef::kLightShadows flag from m\_lightFlags [OdUInt16] member.
  + m\_shadowMapSize [OdUInt16] – size of shadow map texture (have sense only in case if shadows usage enabled for this light source).
  + m\_shadowSoftness [OdUInt16] – number of smoothing steps during shadow map application (have sense only in case if shadows usage enabled for this light source).
  + m\_shadowSamples [OdUInt16] – number of shadow map samples which will be taken into account during shadow map application (have sense only in case if shadows usage enabled for this light source).

#### Xml

Example:

<LightModified>

<LightID>92837461</LightId>

<NModified>127</NModified>

<On>1</On>

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

<Direction>0.0, 0.0, 1.0</Direction>

<Color>1.0, 1.0, 1.0, 1.0</Color>

<ConstantAttenuation>1.0</ConstantAttenuation>

<LinearAttenuation>0.0</LinearAttenuation>

<QuadraticAttenuation>0.0</QuadraticAttenuation>

<UseLimits>1</UseLimits>

<StartAttenuationLimit>1.0</StartAttenuationLimit>

<EndAttenuationLimit>10.0</EndAttenuationLimit>

<SpotCutoff>0.0</SpotCutoff>

<SpotExponent>0.0</SpotExponent>

<SpecularFactor>1.0</SpecularFactor>

<ShadowsEnabled>1</ShadowsEnabled>

<ShadowMapSize>2048</ShadowMapSize>

<ShadowSoftness>1</ShadowSoftness>

<ShadowSamples>16</ShadowSamples>

</LightModified>

* LightID [string] – identifier of exist scene light.
* NModified [integer] – set of bit flags (look for OdTrVisLightDef::kOverride… bit flags) which inform which parts of light data should be modified.
* On [boolean] – set to 1 if this light source is enabled or to 0 if this light source is disabled.
* Position [float \* 3] – light position in world space.
* Direction [float \* 3] – light direction in world space.
* Color [float \* 4] – light color (RGBA components in 0-1 range).
* ConstantAttenuation [float] – constant light attenuation component.
* LinearAttenuation [float] – linear light attenuation component.
* QuadraticAttenuation [float] – quadratic light attenuation component.
* UseLimits [boolean] – set to 1 if attenuation limits usage enabled or to 0 if attenuation limits usage disabled.
* StartAttenuationLimit [float] – distance from light source position where lighted range should be started (have sense only in case if attenuation limits usage enabled for this light source).
* EndAttenuationLimit [float] – distance from light source position where lighted range should be ended (have sense only in case if attenuation limits usage enabled for this light source).
* SpotCutoff [float] – light spot cutoff angle.
* SpotExponent [float] – light spot exponent factor.
* SpecularFactor [float] – multiplier for specular highlighting.
* ShadowsEnabled [boolean] – set to 1 if shadows usage enabled or to 0 if shadows usage disabled.
* ShadowMapSize [integer] – size of shadow map texture (have sense only in case if shadows usage enabled for this light source).
* ShadowSoftness [integer] – number of smoothing steps during shadow map application (have sense only in case if shadows usage enabled for this light source).
* ShadowSamples [integer] – number of shadow map samples which will be taken into account during shadow map application (have sense only in case if shadows usage enabled for this light source).

### onLightDeleted

**virtual** **void** onLightDeleted(OdTrVisLightId lightId) = 0;

#### C++

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

#### Xml

Example:

<LightDeleted>

<LightID>9030045321</LightID>

</LightDeleted>

* LightID [string] – identifier of deleted light source.

### onLightPropertyAttached

**virtual** **void** onLightPropertyAttached(OdTrVisLightId lightId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional light source data, like database owning object handle, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this light source object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this light source object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this light source object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this light source object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this light source object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this light source object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this light source object;
* OdTrVisPropertyDef::kNameProperty – optional light source object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this light source is a persistent object, which couldn’t be erased.

#### Xml

Example:

<LightPropertyAttached>

<LightID>827464521</LightID>

<PropType>0</PropType>

<PropVal>

<OwnerID>1F77A</OwnerID>

</PropVal>

</LightPropertyAttached>

* LightID [string] – identifier of light source object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional state branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onLightPropertyDetached

**virtual** **void** onLightPropertyDetached(OdTrVisLightId lightId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached light source data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this light source object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this light source object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this light source object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this light source object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this light source object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this light source object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this light source object;
* OdTrVisPropertyDef::kNameProperty – optional light source object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this light source is a persistent object, which couldn’t be erased.

#### Xml

Example:

<LightPropertyDetached>

<LightID>827464521</LightID>

<PropType>0</PropType>

</LightPropertyDetached>

* LightID [string] – identifier of light source object to detach additional data.
* PropType [integer] – identifies property type to detach.

### onStateBranchAdded

**virtual** **void** onStateBranchAdded(OdTrVisStateBranchId stateBranchId, **const** OdTrVisStateBranchDef &pDef) = 0;

#### C++

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

State branch consists from: 1) list of child state branches to traverse through state branches tree during rendering process; 2) list of state markers to process (highlight or hide) sub elements of graphic metafiles.

#### Xml

Example:

<StateBranchAdded>

<StateBranchID>51778696</StateBranchID>

<NChilds>1</NChilds>

<NMarkers>2</NMarkers>

<Childs>

<StateBranchId>51763776</StateBranchId>

</Childs>

<Markers>

<Marker>1</Marker>

<Marker>3</Marker>

</Markers>

</StateBranchAdded>

* StateBranchID [string] – identifier of state branch to be added into graphics scene.
* NChilds [integer] – number of child state branches for this state branch.
* NMarkers [integer] – number of state markers for this state branch.
* Childs [tag] – set of <StateBranchId> tags [string] each of which specifies child state branch for this state branch.
* Markers [tag] – set of <Marker> tags [64 bit integer] each of which specifies state marker for this state branch.

### onStateBranchModified

**virtual** **void** onStateBranchModified(OdTrVisStateBranchId stateBranchId, **const** OdTrVisStateBranchDef &pDef, **bool** bChildsModified, **bool** bMarkersModified) = 0;

#### C++

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

#### Xml

Example:

<StateBranchModified>

<StateBranchID>51778696</StateBranchID>

<NChilds>1</NChilds>

<NMarkers>2</NMarkers>

<Childs>

<StateBranchID>51763776</StateBranchID>

</Childs>

<Markers>

<Marker>1</Marker>

<Marker>3</Marker>

</Markers>

</StateBranchModified>

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

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

### onStateBranchDeleted

**virtual** **void** onStateBranchDeleted(OdTrVisStateBranchId stateBranchId) = 0;

#### C++

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

#### Xml

Example:

<StateBranchDeleted>

<StateBranchID>51778696</StateBranchID>

</StateBranchDeleted>

* StateBranchID [string] – identifier of deleted state branch.

### onStateBranchPropertyAttached

**virtual** **void** onStateBranchPropertyAttached(OdTrVisStateBranchId stateBranchId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional state branch data, like database owning object handle, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this state branch object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this state branch object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this state branch object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this state branch object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this state branch object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this state branch object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this state branch object;
* OdTrVisPropertyDef::kNameProperty – optional state branch object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this state branch is a persistent object, which couldn’t be erased.

#### Xml

Example:

<StateBranchPropertyAttached>

<StateBranchID>89838376</StateBranchID>

<PropType>0</PropType>

<PropVal>

<OwnerID>1F77A</OwnerID>

</PropVal>

</StateBranchPropertyAttached>

* StateBranchID [string] – identifier of state branch object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional state branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onStateBranchPropertyDetached

**virtual** **void** onStateBranchPropertyDetached(OdTrVisStateBranchId stateBranchId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached state branch data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this state branch object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this state branch object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this state branch object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this state branch object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this state branch object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this state branch object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this state branch object;
* OdTrVisPropertyDef::kNameProperty – optional state branch object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this state branch is a persistent object, which couldn’t be erased.

#### Xml

Example:

<StateBranchPropertyDetached>

<StateBranchID>89838376</StateBranchID>

<PropType>0</PropType>

</StateBranchPropertyDetached>

* StateBranchID [string] – identifier of state branch object to detach additional data.
* PropType [integer] – identifies property type to detach.

### 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.
  + OdTrVisTextureDef::kIntensityMap – informs renderer that even if image contains alpha channel, we must use RGB color components to compute intensity. This mode actual for height and displacement maps, but also can be useful for opacity material channel.
  + OdTrVisTextureDef::kNegative – informs renderer that texture color components is inverted and must be negated before use.
  + OdTrVisTextureDef::kNormalMap – enabled if texture represents bump or normal map, so calculation of normal map from color map isn’t required for bump or normal map material channels.
  + OdTrVisTextureDef::kSphereMap – informs renderer that texture represents spherical environment map which can be used for reflection material channel. In case if this flag set together with OdTrVisTextureDef::kCubeMap, spherical map will be calculated from exist cube map texture.
  + OdTrVisTextureDef::kCubeMap – informs renderer that texture represent cube environment map which can be used for reflection material channel. In case if this flag set together with OdTrVisTextureDef::kSphereMap, spherical map will be calculated from this cube map texture.
  + OdTrVisTextureDef::kGrayscale – enabled if texture already contains precalculated intensity in RGB color components. In this case renderer can avoid intensity calculation for height/intensity maps.
* m\_pTexture [pointer] – pointer to the OdTrVisTexture texture interface.

#### Xml

Example:

<TextureAdded>

<TextureID>145554768</TextureID>

<Type>1</Type>

<Smooth>1</Smooth>

<Modulate>0</Modulate>

<IntensityMap>0</IntensityMap>

<Negative>0</Negative>

<NormalMap>0</NormalMap>

<SphereMap>0</SphereMap>

<CubeMap>0</CubeMap>

<Grayscale>0</Grayscale>

<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.
* 6 – 4-half-float per pixel texture data with RGBA colors order.
* 7 – 1-byte per pixel texture data without palette (grayscale).
* 8 – 1-byte per pixel texture data without palette (alpha color component).
* 9 – 1-double-word per pixel texture data (32-bit depth buffer data).
* 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.
* IntensityMap [boolean] – set to 1 if renderer must use RGB color components to compute intensity even if image contains alpha channel. This mode actual for height and displacement maps, but also can be useful for opacity material channel.
* Negative [boolean] – set to 1 if renderer must invert color components before use.
* NormalMap [boolean] – set to 1 if texture represents bump or normal map, so calculation of normal map from color map isn’t required for bump or normal map material channels.
* SphereMap [boolean] – set to 1 if texture represents spherical environment map which can be used for reflection material channel. In case if this flag set together with CubeMap boolean flag, spherical map will be calculated from exist cube map texture.
* CubeMap [boolean] – set to 1 if texture represent cube environment map which can be used for reflection material channel. In case if this flag set together with SphereMap boolean flag, spherical map will be calculated from this cube map texture.
* Grayscale [boolean] – set to 1 texture already contains precalculated intensity in RGB color components. In this case renderer can avoid intensity calculation for height/intensity maps.
* 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 tag.
* 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.

### onTexturePropertyAttached

**virtual** **void** onTexturePropertyAttached(OdTrVisTextureId textureId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional texture data, like texture name, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this texture object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this texture object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this texture object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this texture object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this texture object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this texture object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this texture object;
* OdTrVisPropertyDef::kNameProperty – optional texture object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this texture is a persistent object, which couldn’t be erased.

#### Xml

Example:

<TexturePropertyAttached>

<TextureID>9898766</TextureID>

<PropType>7</PropType>

<PropVal>

<Text>Texture Name</Text>

</PropVal>

</TexturePropertyAttached>

* TextureID [string] – identifier of texture object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional state branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onTexturePropertyDetached

**virtual** **void** onTexturePropertyDetached(OdTrVisTextureId textureId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached texture data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this texture object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this texture object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this texture object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this texture object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this texture object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this texture object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this texture object;
* OdTrVisPropertyDef::kNameProperty – optional texture object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this texture is a persistent object, which couldn’t be erased.

#### Xml

Example:

<TexturePropertyDetached>

<TextureID>9898766</TextureID>

<PropType>7</PropType>

</TexturePropertyDetached>

* TextureID [string] – identifier of texture object to detach additional data.
* PropType [integer] – identifies property type to detach.

### 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\_ambientChannel – set of parameters describing material ambient lighting channel. See OdTrVisMaterialChannelDef structure member’s description.
* m\_diffuseChannel – set of parameters describing material diffuse lighting channel. See OdTrVisMaterialChannelDef structure member’s description.
* m\_specularChannel – set of parameters describing material specular highlighting channel. See OdTrVisMaterialChannelDef structure member’s description.
* m\_emissionChannel – set of parameters describing material self-illumination channel. See OdTrVisMaterialChannelDef structure member’s description.
* m\_opacityChannel – set of parameters describing material opacity channel. See OdTrVisMaterialChannelDef structure member’s description.
* m\_bumpMapChannel – set of parameters describing material bump-mapping lighting channel. See OdTrVisMaterialChannelDef structure member’s description.
* m\_reflectChannel – set of parameters describing material reflection channel. See OdTrVisMaterialChannelDef structure member’s description.
* m\_refractChannel – set of parameters describing material refraction channel. See OdTrVisMaterialChannelDef structure member’s description.
* m\_normMapChannel – set of parameters describing material normal map lighting channel. See OdTrVisMaterialChannelDef structure member’s description.
* m\_tintChannel – set of parameters describing material tinting channel. See OdTrVisMaterialChannelDef structure member’s description.

OdTrVisMaterialChannelDef member’s description:

* m\_realisticComponent – describe material channel parameters for realistic rendering modes. See OdTrVisMaterialChannelComponent structure member’s description.
* m\_shadingComponent – describe material channel parameters for non-realistic rendering modes. See OdTrVisMaterialChannelComponent structure member’s description.

OdTrVisMaterialChannelComponent member’s description:

* m\_channelColor [OdTrVisMaterialColor] – material color represented by RGB floating point color components in 0-1 range.
* m\_blendFactor [float] – factor to blend between material channel color and vertex color (0 – full material color; 1 – full vertex color).
* m\_opacityLevel [float] – material transparency (0 – completely transparent; 1 – opaque). Have sense for diffuse material channel only.
* m\_channelValue [float] – specular highlighting (gloss) factor for specular material channel. Effect scale for bump and normal map material channels. Reflectivity factor for reflection material channel. For other channels it is provide color multiplier which is typically set to 1.
* m\_textureBlend [float] – factor to blend between texture and material channel color (0 – full diffuse color; 1 – full texture color).
* m\_textureId (OdTrVisTextureId) – optional material texture identifier.
* m\_uWrap (enum) – horizontal texture wrapping mode:
* OdTrVisMaterialChannelComponent::kWrapRepeat – repeat texture if texture coordinates outside 0-1 range.
* OdTrVisMaterialChannelComponent::kWrapClamp – clamp texture to edges if texture coordinates outside 0-1 range.
* OdTrVisMaterialChannelComponent::kWrapCrop – crops texture to transparent color if texture coordinates outside 0-1 range.
* OdTrVisMaterialChannelComponent::kWrapMirror – similar as repeat, but each odd texture repetition is inverted.
* m\_vWrap (enum) – vertical texture wrapping mode:
* OdTrVisMaterialChannelComponent::kWrapRepeat – repeat texture if texture coordinates outside 0-1 range.
* OdTrVisMaterialChannelComponent::kWrapClamp – clamp texture to edges if texture coordinates outside 0-1 range.
* OdTrVisMaterialChannelComponent::kWrapCrop – crops texture to transparent color if texture coordinates outside 0-1 range.
* OdTrVisMaterialChannelComponent::kWrapMirror – similar as repeat, but each odd texture repetition is inverted.

#### Xml

Example:

<MaterialAdded>

<MaterialID>51697328</MaterialID>

<AmbientChannel>

<RealisticComponent>

<ChannelColor>0.501961, 0.501961, 0.501961</ChannelColor>

<BlendFactor>0</BlendFactor>

<OpacityLevel>1.000000</OpacityLevel>

<ChannelValue>1.000000</ChannelValue>

<TextureBlend>0</TextureBlend>

<TextureID>0</TextureID>

<UWrapType>0</UWrapType>

<VWrapType>0</VWrapType>

</RealisticComponent>

<ShadingComponent>

<ChannelColor>0.501961, 0.501961, 0.501961</ChannelColor>

<BlendFactor>0</BlendFactor>

<OpacityLevel>1.000000</OpacityLevel>

<ChannelValue>1.000000</ChannelValue>

<TextureBlend>0</TextureBlend>

<TextureID>0</TextureID>

<UWrapType>0</UWrapType>

<VWrapType>0</VWrapType>

</ShadingComponent>

</AmbientChannel>

<DiffuseChannel>

<RealisticComponent>

<ChannelColor>0.501961, 0.501961, 0.501961</ChannelColor>

<BlendFactor>0</BlendFactor>

<OpacityLevel>1.000000</OpacityLevel>

<ChannelValue>1.000000</ChannelValue>

<TextureBlend>1.000000</TextureBlend>

<TextureID>204</TextureID>

<UWrapType>0</UWrapType>

<VWrapType>0</VWrapType>

</RealisticComponent>

<ShadingComponent>

<ChannelColor>0.501961, 0.501961, 0.501961</ChannelColor>

<BlendFactor>0</BlendFactor>

<OpacityLevel>1.000000</OpacityLevel>

<ChannelValue>1.000000</ChannelValue>

<TextureBlend>0</TextureBlend>

<TextureID>0</TextureID>

<UWrapType>0</UWrapType>

<VWrapType>0</VWrapType>

</ShadingComponent>

</DiffuseChannel>

<SpecularChannel>

<RealisticComponent>

<ChannelColor>1.000000, 1.000000, 1.000000</ChannelColor>

<BlendFactor>0</BlendFactor>

<OpacityLevel>1.000000</OpacityLevel>

<ChannelValue>64.000000</ChannelValue>

<TextureBlend>0</TextureBlend>

<TextureID>0</TextureID>

<UWrapType>0</UWrapType>

<VWrapType>0</VWrapType>

</RealisticComponent>

<ShadingComponent>

<ChannelColor>1.000000, 1.000000, 1.000000</ChannelColor>

<BlendFactor>0</BlendFactor>

<OpacityLevel>1.000000</OpacityLevel>

<ChannelValue>64.000000</ChannelValue>

<TextureBlend>0</TextureBlend>

<TextureID>0</TextureID>

<UWrapType>0</UWrapType>

<VWrapType>0</VWrapType>

</ShadingComponent>

</SpecularChannel>

. . .

</MaterialAdded>

* MaterialID [string] – identifier of material to add.
* AmbientChannel [tag] – Describe parameters for material ambient lighting channel:
  + RealisticComponent [tag] – Describe material channel parameters for realistic rendering modes:
    - ChannelColor [float \* 3] – material channel color (RGB color components in 0-1 range).
    - BlendFactor [float] – factor to blend between material channel color and vertex color (0 – full material color; 1 – full vertex color).
    - OpacityLevel [float] – material transparency (0 – completely transparent; 1 – opaque). Have sense for diffuse material channel only.
    - ChannelValue [float] – specular highlighting (gloss) factor for specular material channel. Effect scale for bump and normal map material channels. Reflectivity factor for reflection material channel. For other channels it is provide color multiplier which is typically set to 1.
    - TextureBlend [float] – factor to blend between texture and material channel 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.
  + ShadingComponent [tag] – Describe material channel parameters for non-realistic rendering modes. Include similar set of parameters as for RealisticComponent tag.
* DiffuseChannel [tag] – Describe parameters for material diffuse lighting channel. Include similar set of parameters as for AmbientChannel tag.
* SpecularChannel [tag] – Describe parameters for material specular highlighting channel. Include similar set of parameters as for AmbientChannel tag.
* EmissionChannel [tag] – Describe parameters for material self-illumination channel. Include similar set of parameters as for AmbientChannel tag.
* OpacityChannel [tag] – Describe parameters for material opacity channel. Include similar set of parameters as for AmbientChannel tag.
* BumpMapChannel [tag] – Describe parameters for material bump-mapping lighting channel. Include similar set of parameters as for AmbientChannel tag.
* ReflectionChannel [tag] – Describe parameters for material reflection channel. Include similar set of parameters as for AmbientChannel tag.
* RefractionChannel [tag] – Describe parameters for material refraction channel. Include similar set of parameters as for AmbientChannel tag.
* NormalMapChannel [tag] – Describe parameters for material normal mapping lighting channel. Include similar set of parameters as for AmbientChannel tag.
* TintChannel [tag] – Describe parameters for material tinting channel. Include similar set of parameters as for AmbientChannel tag.

### 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.

### onMaterialPropertyAttached

**virtual** **void** onMaterialPropertyAttached(OdTrVisMaterialId materialId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional material data, like material name, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this material object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this material object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this material object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this material object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this material object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this material object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this material object;
* OdTrVisPropertyDef::kNameProperty – optional material object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this material is a persistent object, which couldn’t be erased.

#### Xml

Example:

<MaterialPropertyAttached>

<MaterialID>51697328</MaterialID>

<PropType>7</PropType>

<PropVal>

<Text>New Material</Text>

</PropVal>

</MaterialPropertyAttached>

* MaterialID [string] – identifier of material object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional state branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onMaterialPropertyDetached

**virtual** **void** onMaterialPropertyDetached(OdTrVisMaterialId materialId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached material data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this material object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this material object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this material object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this material object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this material object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this material object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this material object;
* OdTrVisPropertyDef::kNameProperty – optional material object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this material is a persistent object, which couldn’t be erased.

#### Xml

Example:

<MaterialPropertyDetached>

<MaterialID>51697328</MaterialID>

<PropType>7</PropType>

</MaterialPropertyDetached>

* MaterialID [string] – identifier of material object to detach additional data.
* PropType [integer] – identifies property type to detach.

### 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>

<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.
* 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>

<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.
* 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.

### onVisualStylePropertyAttached

**virtual** **void** onVisualStylePropertyAttached(OdTrVisVisualStyleId visualStyleId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional visual style data, like visual style name, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this visual style object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this visual style object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this visual style object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this visual style object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this visual style object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this visual style object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this visual style object;
* OdTrVisPropertyDef::kNameProperty – optional visual style object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this visual style is a persistent object, which couldn’t be erased.

#### Xml

Example:

<VisualStylePropertyAttached>

<VisualStyleID>65734937</VisualStyleID>

<PropType>7</PropType>

<PropVal>

<Text>2dWireframe</Text>

</PropVal>

</VisualStylePropertyAttached>

* VisualStyleID [string] – identifier of visual style object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional state branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onVisualStylePropertyDetached

**virtual** **void** onVisualStylePropertyDetached(OdTrVisVisualStyleId visualStyleId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached visual style data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this visual style object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this visual style object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this visual style object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this visual style object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this visual style object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this visual style object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this visual style object;
* OdTrVisPropertyDef::kNameProperty – optional visual style object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this visual style is a persistent object, which couldn’t be erased.

#### Xml

Example:

<VisualStylePropertyDetached>

<VisualStyleID>65734937</VisualStyleID>

<PropType>4</PropType>

</VisualStylePropertyDetached>

* VisualStyleID [string] – identifier of visual style object to detach additional data.
* PropType [integer] – identifies property type to detach.

### 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\_props [OdTrVisLayerProps] – structure represents set of generic layer properties and flags:
  + OdTrVisLayerProps::kLyVisible – layer is visible.
  + OdTrVisLayerProps::kLyFaded – layer is faded.

#### Xml

Example:

<LayerAdded>

<LayerID>443253466</LayerID>

<Name>0</Name>

<Visibility>1</Visibility>

<Faded>0</Faded>

</LayerAdded>

* LayerID [string] – identifier of layer to add.
* Visibility [boolean] – enables layer visibility.
* Faded [boolean] – enabled layer fading.

### 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.

### onLayerVisibilityChanged

**virtual** **void** onLayerVisibilityChanged(OdTrVisLayerId layerId, **bool** bVisibility) = 0;

#### C++

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

#### Xml

Example:

<LayerVisibilityChanged>

<LayerID>443253466</LayerID>

<Visibility>0</Visibility>

</LayerVisibilityChanged>

* LayerID [string] – identifier of layer to change visibility state.
* Visibility [boolean] – new visibility state.

### onLayerFadingChanged

**virtual** **void** onLayerFadingChanged(OdTrVisLayerId layerId, **bool** bFade) = 0;

#### C++

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

#### Xml

Example:

<LayerFadingChanged>

<LayerID>443253466</LayerID>

<Fade>0</Fade>

</LayerFadingChanged>

* LayerID [string] – identifier of layer to change fading state.
* Fade [boolean] – new fading state.

### onLayerViewportPropsOverride

**virtual** **void** onLayerVpDepPropsOverride(OdTrVisLayerId layerId, OdTrVisViewportId viewportId, **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:

<LayerVpDepPropsOverride>

<LayerID>1FA</LayerID>

<ViewportID>218</ViewportID>

<Override>1</Override>

<VpDepSpec>

<Visibility>1</Visibility>

<Faded>0</Faded>

</VpDepSpec>

</LayerVpDepPropsOverride>

<LayerVpDepPropsOverride>

<LayerID>1FA</LayerID>

<ViewportID>219</ViewportID>

<Override>0</Override>

</LayerVpDepPropsOverride>

* LayerID [string] – layer ID that contains specific properties.
* ViewportID [string] – viewport ID with which layer contains specific properties.
* Override [boolean] – set to 1 if layer properties must be overrides for specified viewport or 0 elsewhere.
* VpDepSep [tag] – include overridden layer property state:
  + Visibility [boolean] – layer visibility override;
  + Faded [boolean] – layer fading override.

### onLayerPropertyAttached

**virtual** **void** onLayerPropertyAttached(OdTrVisLayerId layerId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional layer data, like layer name, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this layer object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this layer object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this layer object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this layer object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this layer object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this layer object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this layer object;
* OdTrVisPropertyDef::kNameProperty – optional layer object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this layer is a persistent object, which couldn’t be erased.

#### Xml

Example:

<LayerPropertyAttached>

<LayerID>1FA</LayerID>

<PropType>7</PropType>

<PropVal>

<Text>My Layer Name</Text>

</PropVal>

</LayerPropertyAttached>

* LayerID [string] – identifier of layer object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional state branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onLayerPropertyDetached

**virtual** **void** onLayerPropertyDetached(OdTrVisLayerId layerId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached layer data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this layer object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this layer object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this layer object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this layer object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this layer object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this layer object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this layer object;
* OdTrVisPropertyDef::kNameProperty – optional layer object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this layer is a persistent object, which couldn’t be erased.

#### Xml

Example:

<LayerPropertyDetached>

<LayerID>1FA</LayerID>

<PropType>7</PropType>

</LayerPropertyDetached>

* LayerID [string] – identifier of layer object to detach additional data.
* PropType [integer] – identifies property type to detach.

### onGroupAdded

**virtual** **void** onGroupAdded(OdTrVisGroupId groupId) = 0;

#### C++

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

#### Xml

Example:

<GroupAdded>

<GroupID>5409209</GroupID>

</GroupAdded>

* GroupID [string] – identifier of newly created metafiles group.

### onGroupDeleted

**virtual** **void** onGroupDeleted(OdTrVisGroupId groupId) = 0;

#### C++

This method will be called by vectorizer in case if exist metafiles is deleted from graphics scene. All attached metafiles will be automatically detached from deleted metafiles group.

#### Xml

Example:

<GroupDeleted>

<GroupID>5409209</GroupID>

</GroupDeleted>

* GroupID [string] – identifier of metafiles group to be deleted.

### onGroupMetafilesAttached

**virtual** **void** onGroupMetafilesAttached(OdTrVisGroupId groupId, **const** OdTrVisMetafileId \*pMetafilesList, OdUInt32 nMetafiles) = 0;

#### C++

This method will be called by vectorizer to append list of exist metafiles into exist metafiles group.

#### Xml

Example:

<GroupMetafilesAttached>

<GroupID>5409209</GroupID>

<NMetafiles>3<NMetafiles>

<Metafiles>

<MetafileID>78298374</MetafileId>

<MetafileID>908809086</MetafileId>

<MetafileID>184756528</MetafileId>

</Metafiles>

</GroupMetafilesAttached>

* GroupID [string] – identifier of exist metafiles group.
* NMetafiles [integer] – number of metafiles to be attached into metafiles group.
* Metafiles [tag] – scopes metafile identifiers list:
  + MetafileID [string] – separate metafile identifier for each metafile in list.

### onGroupMetafilesDetached

**virtual** **void** onGroupMetafilesDetached(OdTrVisGroupId groupId, **const** OdTrVisMetafileId \*pMetafilesList, OdUInt32 nMetafiles) = 0;

#### C++

This method will be called by vectorizer to detach list of exist metafiles from exist metafiles group. If zero metafiles number passed into list of metafiles – all previously attached metafiles will be detached from metafiles group.

#### Xml

Example:

<GroupMetafilesDetached>

<GroupID>5409209</GroupID>

<NMetafiles>2<NMetafiles>

<Metafiles>

<MetafileID>78298374</MetafileId>

<MetafileID>184756528</MetafileId>

</Metafiles>

</GroupMetafilesDetached>

* GroupID [string] – identifier of exist metafiles group.
* NMetafiles [integer] – number of metafiles to be detached from metafiles group.
* Metafiles [tag] – scopes metafile identifiers list:
  + MetafileID [string] – separate metafile identifier for each metafile in list.

### Obsolete: onGroupMetafilesDeleted

**virtual** **void** onGroupMetafilesDeleted(OdTrVisGroupId groupId) = 0;

#### C++

This method can be called by vectorizer to delete all metafiles, which is previously attached to specified metafiles group, from graphics scene. After removing of metafiles, metafiles group will be empty, but this call will not delete metafiles group itself.

#### Xml

Example:

<GroupMetafilesDeleted>

<GroupID>5409209</GroupID>

</GroupMetafilesDeleted>

* GroupID [string] – identifier of metafiles group, containing attached metafiles to be deleted.

### onGroupMetafilesVisibilityChanged

**virtual** **void** onGroupMetafilesVisibilityChanged(OdTrVisGroupId groupId, **bool** bVisibility) = 0;

#### C++

This method will be called by vectorizer to change visibility state for all metafiles, which is attached to specified metafiles group.

#### Xml

Example:

<GroupMetafilesVisibilityChanged>

<GroupID>5409209</GroupID>

<Visibility>0</Visibility>

</GroupMetafilesVisibilityChanged>

* GroupID [string] – identifier of metafiles group to change visibility state.
* Visibility [boolean] – new visibility state for all attached metafiles.

### onGroupMetafilesHighlightingChanged

**virtual** **void** onGroupMetafilesHighlightingChanged(OdTrVisGroupId groupId, **bool** bHighlight) = 0;

#### C++

This method will be called by vectorizer to change highlighting state for all metafiles, which is attached to specified metafiles group.

#### Xml

Example:

<GroupMetafilesHighlightingChanged>

<GroupID>5409209</GroupID>

<Highlight>1</Highlight>

</GroupMetafilesHighlightingChanged>

* GroupID [string] – identifier of metafiles group to change highlighting state.
* Highlight [boolean] – new highlighting state for all attached metafiles.

### onGroupMetafilesFadingChanged

**virtual** **void** onGroupMetafilesFadingChanged(OdTrVisGroupId groupId, **bool** bFade) = 0;

#### C++

This method will be called by vectorizer to change fading state for all metafiles, which is attached to specified metafiles group.

#### Xml

Example:

<GroupMetafilesFadingChanged>

<GroupID>5409209</GroupID>

<Fade>1</Fade>

</GroupMetafilesFadingChanged>

* GroupID [string] – identifier of metafiles group to change fading state.
* Fade [boolean] – new fading state for all attached metafiles.

### onGroupPropertyAttached

**virtual** **void** onGroupPropertyAttached(OdTrVisGroupId groupId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach metafiles group data, like linked layer, for example. These properties will be attached to all metafiles, which is previously attached to this metafiles group, in case if properties attached to metafile itself will not override metafiles group properties. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this metafiles group object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this metafiles group object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this metafiles group object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this metafiles group object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this metafiles group object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this metafiles group object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this metafiles group object;
* OdTrVisPropertyDef::kNameProperty – optional metafiles group object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this metafiles group is a persistent object, which couldn’t be erased.

#### Xml

Example:

<GroupPropertyAttached>

<GroupID>5409209</GroupID>

<PropType>2</PropType>

<PropVal>

<LayerID>89876990</LayerID>

</PropVal>

</GroupPropertyAttached>

* GroupID [string] – identifier of metafiles group object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional state branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onGroupPropertyDetached

**virtual** **void** onGroupPropertyDetached(OdTrVisGroupId groupId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached metafiles group data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this metafiles group object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this metafiles group object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this metafiles group object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this metafiles group object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this metafiles group object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this metafiles group object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this metafiles group object;
* OdTrVisPropertyDef::kNameProperty – optional metafiles group object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this metafiles group is a persistent object, which couldn’t be erased.

#### Xml

Example:

<GroupPropertyDetached>

<GroupID>5409209</GroupID>

<PropType>2</PropType>

</GroupPropertyDetached>

* GroupID [string] – identifier of metafiles group object to detach additional data.
* PropType [integer] – identifies property type to detach.

### onExtensionObjectAdded

**virtual** **void** onExtensionObjectAdded(OdTrVisExtensionId extId, **const** OdTrVisExtensionObject \*pObj) = 0;

#### C++

This method will be called by vectorizer in case if new extension object is added into graphics scene. Each extension object represented by their own unique OdtrVisExtensionObject interface.

#### Xml

Example:

<ExtensionObjectAdded>

<ExtensionID>123356348</ExtensionID>

<ModuleName>ExtensionModule.tre</Type>

<ExtVersion>100</ExtVersion>

<ObjectType>0</ObjectType>

<ObjectData>A2EAA12590DA0DE1C2FEA115D3</ObjectData>

</ExtensionObjectAdded>

* ExtensionID [string] – identifier of added extension object.
* ModuleName [string] – name of module, which is provide implementation of this extension object.
* ExtVersion [integer] – version of extension module.
* ObjectType [integer] – identify extension object type inside specified extension module.
* ObjectData [binary] – binary object data, specific for this extension object type deserialization.

### onExtensionObjectDeleted

**virtual** **void** onExtensionObjectDeleted(OdTrVisExtensionId extId) = 0;

#### C++

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

#### Xml

Example:

<ExtensionObjectDeleted>

<ExtensionID>123356348</ExtensionID>

</ExtensionObjectDeleted>

* ExtensionID [string] – identifier of extension object to remove.

### onExtensionObjectSpecificData

**virtual** **void** onExtensionObjectSpecificData(OdTrVisExtensionId extId, OdUInt32 nDataId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to setup specific extension object data. Following data types currently supported:

* OdTrVisPropertyDef::kOwningProperty – OdTrVisOwning data type;
* OdTrVisPropertyDef::kXformProperty – OdGeMatrix3d data type;
* OdTrVisPropertyDef::kLayerProperty – OdTrVisLayerId data type;
* OdTrVisPropertyDef::kHlBranchProperty – OdTrVisStateBranchId data type;
* OdTrVisPropertyDef::kVsBranchProperty – OdTrVisStateBranchId data type.
* OdTrVisPropertyDef::kVisualStyleProperty – OdTrVisVisualStyleId data type;
* OdTrVisPropertyDef::kSelStyleProperty – OdUInt32 data type;
* OdTrVisPropertyDef::kNameProperty – OdString data type;
* OdTrVisPropertyDef::kSysDefaultProperty – boolean data type.

#### Xml

Example:

<ExtensionObjectSpecificData>

<ExtensionID>123356348</ExtensionID>

<DataID>7<DataID>

<PropType>8</PropType>

<PropVal>

<Boolean>1</Boolean>

</PropVal>

</ExtensionObjectSpecificData>

* ExtensionID [string] – identifier of extension object to setup additional data.
* DataID [integer] – internal data entry identifier.
* PropType [integer] – identifies property type to set.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + StateBranchID [string] – represents optional state branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onExtensionObjectVisibilityChanged

**virtual** **void** onExtensionObjectVisibilityChanged(OdTrVisExtensionId extId, **bool** bVisibility) = 0;

#### C++

This method will be called by vectorizer to change visibility state for specified extension object.

#### Xml

Example:

<ExtensionObjectVisibilityChanged>

<ExtensionID>123356348</ExtensionID>

<Visibility>0</Visibility>

</ExtensionObjectVisibilityChanged>

* ExtensionID [string] – identifier of extension object to change visibility state.
* Visibility [boolean] – new visibility state for specified extension object.

### onExtensionObjectPropertyAttached

**virtual** **void** onExtensionObjectPropertyAttached(OdTrVisExtensionId extId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional extension object data, like extension object name, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this extension object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this extension object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this extension object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this extension object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this extension object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this extension object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this extension object;
* OdTrVisPropertyDef::kNameProperty – optional extension object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this extension object is a persistent object, which couldn’t be erased.

#### Xml

Example:

<ExtensionObjectPropertyAttached>

<ExtensionID>123356348</ExtensionID>

<PropType>7</PropType>

<PropVal>

<Text>Extension Object Name</Text>

</PropVal>

</ExtensionObjectPropertyAttached>

* ExtensionID [string] – identifier of extension object to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + HlBranchID [string] – represents optional highlighting branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onExtensionObjectPropertyDetached

**virtual** **void** onExtensionObjectPropertyDetached(OdTrVisExtensionId extId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached extension object data properties by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this extension object;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this extension object;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this extension object;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this extension object;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this extension object;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this extension object;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this extension object;
* OdTrVisPropertyDef::kNameProperty – optional extension object name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this extension object is a persistent object, which couldn’t be erased.

#### Xml

Example:

<ExtensionObjectPropertyDetached>

<ExtensionID>123256348</ExtensionID>

<PropType>7</PropType>

</ExtensionObjectPropertyDetached>

* ExtensionID [string] – identifier of extension object to detach property data.
* PropType [integer] – identifies property type to detach.

### onBlockAdded

**virtual** **void** onBlockAdded(OdTrVisBlockId blockId, **const** OdTrVisDisplayId \*pList, OdUInt32 nEntries) = 0;

#### C++

This method will be called by vectorizer in case if new block is added into graphics scene. Block objects represented as simple display lists.

#### Xml

Example:

<BlockAdded>

<BlockID>123256348</BlockID>

<ListLen>8</ListLen>

<DisplayList>

<MetafileID>51778696</MetafileID>

<MetafileID>51763776</MetafileID>

<MetafileID>51614288</MetafileID>

<MetafileID>145564568</MetafileID>

<MetafileID>145990928</MetafileID>

<MetafileID>145815672</MetafileID>

<MetafileID>145590088</MetafileID>

<MetafileID>145555960</MetafileID>

</DisplayList>

</BlockAdded>

* BlockID [string] – identifier of added block
* ListLen [integer] – number of metafiles in display list.
* DisplayList [tag] – set of <MetafileID> tags [string] each of which specifies metafile identifier to be added in display list.

### onBlockModified

**virtual** **void** onBlockModified(OdTrVisBlockId blockId, OdUInt32 nInvalidFrom, OdUInt32 nInvalidLen, **const** OdTrVisDisplayId \*pValidFrom, OdUInt32 nValidLen) = 0;

#### C++

This method will be called by vectorizer in case if exist block display list needs to be changed. Syntax of method provides way to remove or modify display list by parts, or add new display list entries in different display list places.

#### Xml

Example:

<BlockModified>

<BlockID>123256348</BlockID>

<InvalidFrom>0</InvalidFrom>

<InvalidLen>0</InvalidLen>

<ValidLen>8</ValidLen>

<DisplayList>

<MetafileID>51778696</MetafileID>

<MetafileID>51763776</MetafileID>

<MetafileID>51614288</MetafileID>

<MetafileID>145564568</MetafileID>

<MetafileID>145990928</MetafileID>

<MetafileID>145815672</MetafileID>

<MetafileID>145590088</MetafileID>

<MetafileID>145555960</MetafileID>

</DisplayList>

</BlockModified>

* BlockID [string] – identifier of block to modify display 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 display list.

### onBlockDeleted

**virtual** **void** onBlockDeleted(OdTrVisBlockId blockId) = 0;

#### C++

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

#### Xml

Example:

<BlockDeleted>

<BlockID>123256348</BlockID>

</BlockDeleted>

* BlockID [string] – identifier block to remove.

### onBlockPropertyAttached

**virtual** **void** onBlockPropertyAttached(OdTrVisBlockId blockId, **const** OdTrVisPropertyDef &propDef) = 0;

#### C++

This method will be called by vectorizer to attach additional block data, like extension object name, for example. Following property types currently supported:

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this block;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this block;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this block;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this block;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this block;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this block;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this block;
* OdTrVisPropertyDef::kNameProperty – optional block name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this block is a persistent object, which couldn’t be erased.

#### Xml

Example:

<BlockPropertyAttached>

<BlockID>123256348</BlockID>

<PropType>7</PropType>

<PropVal>

<Text>Block Name</Text>

</PropVal>

</BlockPropertyAttached>

* BlockID [string] – identifier of block to attach additional data.
* PropType [integer] – identifies property type to attach.
* PropVal [tag] – contains property data:
  + OwnerID [string] – represents optional database handle (in case if property type set as 0);
  + Matrix [float \* 16] – represents optional transformation matrix (in case if property type set as 1);
  + LayerID [string] – represents optional layer linkage (in case if property type set as 2);
  + HlBranchID [string] – represents optional highlighting branch linkage (in case if property type set as 3 or 4);
  + VisualStyleID [string] – represents optional visual style linkage (in case if property type set as 5);
  + StyleIndex [integer] – represents optional selection style index (in case if property type set as 6);
  + Text [string] – represents optional text string (in case if property type set as 7);
  + Boolean [boolean] – represents optional boolean flag (in case if property type set as 8).

### onBlockPropertyDetached

**virtual** **void** onBlockPropertyDetached(OdTrVisBlockId blockId, **const** OdTrVisPropertyDef::PropertyType propType) = 0;

#### C++

This method will be called by vectorizer to detach previously attached block properties data by property type.

* OdTrVisPropertyDef::kOwningProperty – optional database handle, linked with this block;
* OdTrVisPropertyDef::kXformProperty – optional transformation matrix, linked with this block;
* OdTrVisPropertyDef::kLayerProperty – optional layer object, linked with this block;
* OdTrVisPropertyDef::kHlBranchProperty – optional highlighting branch object, linked with this block;
* OdTrVisPropertyDef::kVsBranchProperty – optional visibility branch object, linked with this block;
* OdTrVisPropertyDef::kVisualStyleProperty – optional visual style object, linked with this block;
* OdTrVisPropertyDef::kSelStyleProperty – optional selection style index, linked with this block;
* OdTrVisPropertyDef::kNameProperty – optional block name;
* OdTrVisPropertyDef::kSysDefaultProperty – optional boolean flag, which identifies that this block is a persistent object, which couldn’t be erased.

#### Xml

Example:

<BlockPropertyDetached>

<BlockID>123356348</BlockID>

<PropType>7</PropType>

</BlockPropertyDetached>

* BlockID [string] – identifier of block to detach property data.
* PropType [integer] – identifies property type to detach.

### 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 (for type size 2), double (for type size 3)

Count: 3 (XYZ) components per vertex

##### Normals array

Data type: half-float (for type size 1), float (for type size 2), double (for type size 3)

Count: 3 (XYZ) components per normal

##### Colors array

Data type: OdUInt8 (for type size 0), float (for type size 2), double (for type size 3)

Count: 4 (RGBA) components per color

##### Texture coordinates array

Data type: float (for type size 2), double (for type size 3)

Count: 2 (UV) components per texture coordinate

##### Depths array

Data type: float (for type size 2), double (for type size 3)

Count: 1 (single component per depth)

##### Indexes array

Data type: OdUInt8 (for type size 0), OdUInt16 (for type size 1), OdUInt32 (for type size 2)

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 – Lighting – enable faces lighting for shaded mode.
* 3 – Highlighting – enable geometry highlighting.

#### 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.
* 2 – Lighting – disable faces lighting for shaded mode.
* 3 – Highlighting – disable geometry highlighting.

#### 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 – diffuse texture coordinates array.
* 3 – opacity texture coordinates array.
* 4 – bump-map texture coordinates array.
* 5 – normals array.
* 6 – secondary normals array.
* 7 – bump-mapping T-normals array.
* 8 – bump-mapping B-normals array.
* 9 – 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 – diffuse texture coordinates array.
* 3 – opacity texture coordinates array.
* 4 – bump-map texture coordinates array.
* 5 – normals array.
* 6 – secondary normals array.
* 7 – bump-mapping T-normals array.
* 8 – bump-mapping B-normals array.
* 9 – 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 – Gouraud – enable per-vertex color buffer.
* 1 – MultiNormals – marks facets geometry in case if vertex normal is specified (separate secondary array of normals can be used for FlatShaded render modes).
* 2 – Disable2dLineweights – disable lineweights in 2dOptimized render mode.
* 3 – NoColorOverride – marks geometry which shouldn’t be influenced by color modifiers.
* 4 – LinkedLinesList – marks line strips or loops geometry, exploded onto separate line segments.

#### 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 – Gouraud – disable per-vertex color buffer.
* 1 – MultiNormals – disable multiple normal arrays.
* 2 – Disable2dLineweights – enables lineweights in 2dOptimized render mode.
* 3 – NoColorOverride – ends geometry which shouldn’t be influenced by color modifiers.
* 4 – LinkedLinesList – ends line stips or loops geometry, exploded onto separate line segments.

#### 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 – VisibilityFlags record

Chunk: 25

Data size: 1

Description:

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

Data structure:

* OdUInt8 – set of bit flags (as in OdTrVisVisibilityFlags 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); bit 5 – don’t display geometry in 2d rendering modes; bit 6 – don’t select geometry in 2d rendering modes; bit 7 – don’t display geometry in 3d rendering modes; bit 8 – don’t select geometry in 3d rendering modes.

#### 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.

#### 36 – ExtensionObject record

Chunk: 36

Data size: 8 bytes.

Description:

Draws extension object.

Data structure:

* OdUInt64 – extension object identifier.

#### 37 – SelectionStyle record

Chunk: 37

Data size: 4 bytes.

Description:

Highlights upcoming geometry section using specified selection style.

Data structure:

* OdUInt32 – selection style index.

#### 38 – DPoint record

Chunk: 38

Data size: 24 bytes

Description:

Draw single point with double-precision.

Data structure:

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

#### 39 – DLine record

Chunk: 39

Data size: 48 bytes

Description:

Draw single line with double-precision.

Data structure:

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

#### 40 – Block record

Chunk: 40

Data size: variable

Description:

Render block object, represented as simple display list.

Data structure:

* char – block encoding method:
  + ‘A’ – Auto-block. Display list will be represented as subsequent set of “Metafile” records. Auto-blocks can be represented as different sets of metafile lists for single block identifier;
  + ‘B’ – Built-in block. Render display list from exist block object, which is previously added onto client using “onBlockAdded” method;
  + ‘C’ – Similar with auto-block, but display list will be represented as subsequent array of OdTrVisDisplayId’s;
  + ‘D’ – Similar with embedded block, but display list will be represented as subsequent array of OdTrVisDisplayId’s;
  + ‘E’ – Embedded block. Display list will be represented as subsequent set of “Metafile” records. Embedded block differs from auto-block by possibility to specify only single variation of metafile list (this is something like built-in block, but with embedded display list);
  + ‘F’ – Represents end of ‘A’ or ‘E’ subsequent set of “Metafile” records.
* OdUInt64 – block identifier (available only for ‘A’, ‘B’, ‘C’, ‘D’ and ‘E’ encodings);
* OdUInt32 – number of entries in subsequent display list (available only for ‘C’ and ‘D’ encodings);
* OdUInt32 – reserved padding value must be always set to null (available only for ‘C’ and ‘D’ encodings);
* OdUInt64 \* number of entries – subsequent display list for ‘C’ and ‘D’ encodings.

### Xml graphics stream representation

Graphics metafile stream example:

<MetafileData>

<NArrays>3</NArrays>

<Array>

<Type>0</Type>

<DataType>2</DataType>

<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>

<DataType>2</DataType>

<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>

<DataType>2</DataType>

<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>

<DataType>2</DataType>

<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.
* DataType [integer] – can be 2 – for single floating-point data precision, or 3 – for double floating-point data precision.
* NData [integer] – number of vertexes.
* ArrayData [float \* 3 \* NData] – vertexes array; 3 floating-point variables (XYZ) per vertex.

##### Normals array

Example:

<Array>

<Type>1</Type>

<DataType>2</DataType>

<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.
* DataType [integer] – can be 1 – for half floating-point data precision, 2 – for single floating-point data precision, or 3 – for double floating-point data precision.
* NData [integer] – number of normals.
* ArrayData [float \* 3 \* NData] – normals array; 3 floating-point variables (XYZ) per normal. Half-float values will be passed as single floating-point data through Xml protocol.

##### Colors array

Example:

<Array>

<Type>2</Type>

<DataType>2</DataType>

<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.
* DataType [integer] – can be 0 – for single byte integer precision, 2 – for single floating-point data precision, or 3 – for double floating-point data precision.
* NData [integer] – number of colors in array.
* ArrayData [{OdUInt8 or float} \* 4 \* NData] – colors array; 4 variables (RGBA) per color; colors in 0-1 range for data types 2 and 3, or in 0-255 range for data type 0.

##### Texture coordinates array

Example:

<Array>

<Type>3</Type>

<DataType>2</DataType>

<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.
* DataType [integer] – can be 2 – for single floating-point data precision, or 3 – for double floating-point data precision.
* NData [integer] – number of texture coordinates in array.
* ArrayData [float \* 2 \* NData] – texture coordinates array; 2 floating-point variables (UV) per texture coordinate.

##### Depths array

Example:

<Array>

<Type>4</Type>

<DataType>2</DataType>

<NData>4</NData>

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

</Array>

* Type [integer] – 4 for depths array.
* DataType [integer] – can be 2 – for single floating-point data precision, or 3 – for double floating-point data precision.
* NData [integer] – number of depths in array.
* ArrayData [float \* NData] – depths array; single floating-point variable per depth.

##### Indexes array

Example:

<Array>

<Type>5</Type>

<DataType>1</DataType>

<NData>6</NData>

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

</Array>

* Type [integer] – 5 for indexes array.
* DataType [integer] – can be 0 – single byte data precision, 1 – for single word data precision, or 2 – for double word data precision.
* NData [integer] – number of indexes in array.
* ArrayData [integer \* NData] – indexes array.

##### Markers array

Example:

<Array>

<Type>6</Type>

<DataType>4</DataType>

<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.
* DataType [integer] – always 4 for 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.
* 2 – Lighting – enable faces lighting for shaded mode.
* 3 – Highlighting – enable geometry highlighting.

#### 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.
* 2 – Lighting – disable faces lighting for shaded mode.
* 3 – Highlighting – disable geometry highlighting.

#### 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 – diffuse texture coordinates array.
* 3 – opacity texture coordinates array.
* 4 – bump-map texture coordinates array.
* 5 – normals array.
* 6 – secondary normals array.
* 7 – bump-mapping T-normals array.
* 8 – bump-mapping B-normals array.
* 9 – 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 – diffuse texture coordinates array.
* 3 – opacity texture coordinates array.
* 4 – bump-map texture coordinates array.
* 5 – normals array.
* 6 – secondary normals array.
* 7 – bump-mapping T-normals array.
* 8 – bump-mapping B-normals array.
* 9 – 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 – Gouraud – enable per-vertex color buffer.
* 1 – MultiNormals – marks facets geometry in case if vertex normal is specified (separate secondary array of normals can be used for FlatShaded render modes).
* 2 – Disable2dLineweights – disable lineweights in 2dOptimized render mode.
* 3 – NoColorOverride – marks geometry which shouldn’t be influenced by color modifiers.
* 4 – LinkedLinesList – marks line strips or loops geometry, exploded onto separate line segments.

#### 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 – Gouraud – enable per-vertex color buffer.
* 1 – MultiNormals – disable multiple normal arrays.
* 2 – Disable2dLineweights – enables lineweights in 2dOptimized render mode.
* 3 – NoColorOverride – ends geometry which shouldn’t be influenced by color modifiers.
* 4 – LinkedLinesList – ends line strips or loops geometry, exploded onto separate line segments.

#### 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.

#### VisibilityFlags record

Example:

<VisibilityFlags>

<Flags>10</Flags>

</VisibilityFlags>

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

* 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); bit 5 – don’t display geometry in 2d rendering modes; bit 6 – don’t select geometry in 2d rendering modes; bit 7 – don’t display geometry in 3d rendering modes; bit 8 – don’t select geometry in 3d rendering modes.

#### 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.

#### ExtensionObject record

Example:

<ExtensionObject>

<ExtensionID>998313442</ExtensionID>

</ExtensionObject>

Draws extension object.

* ExtensionID [string] – extension object identifier.

#### SelectionStyle record

Example:

<SelectionStyle>

<Index>1</Index>

</SelectionStyle>

Highlights upcoming geometry using specified selection style.

* Index [integer] – selection style index.

#### DPoint record

Example:

<DPoint>

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

</DPoint>

Draw single point with double-precision.

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

#### DLine record

Example:

<DLine>

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

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

</DLine>

Draw single line with double-precision.

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

#### Block record

Example:

<Block>

<Encoding>C</Encoding>

<BlockID>34355245672</BlockID>

<NumEntries>4</NumEntries>

<Entries>

<MetafileID>3489321</MetafileID>

<MetafileID>1928753</MetafileID>

<MetafileID>2017749</MetafileID>

<MetafileID>1029243</MetafileID>

</Entries>

</Block>

Draw block display list.

* Encoding [string] – block encoding method:
  + ‘A’ – Auto-block. Display list will be represented as subsequent set of “Metafile” records. Auto-blocks can be represented as different sets of metafile lists for single block identifier;
  + ‘B’ – Built-in block. Render display list from exist block object, which is previously added onto client using “onBlockAdded” method;
  + ‘C’ – Similar with auto-block, but display list will be represented as subsequent array of OdTrVisDisplayId’s;
  + ‘D’ – Similar with embedded block, but display list will be represented as subsequent array of OdTrVisDisplayId’s;
  + ‘E’ – Embedded block. Display list will be represented as subsequent set of “Metafile” records. Embedded block differs from auto-block by possibility to specify only single variation of metafile list (this is something like built-in block, but with embedded display list);
  + ‘F’ – Represents end of ‘A’ or ‘E’ subsequent set of “Metafile” records.
* BlockID [string] – block identifier (available only for ‘A’, ‘B’, ‘C’, ‘D’ and ‘E’ encodings);
* NumEntries [integer] – number of entries in subsequent display list (available only for ‘C’ and ‘D’ encodings);
* Entries [tag] – set of <MetafileID> tags [string] each of which specifies metafile identifier to be added in block display list (available only for ‘C’ and ‘D’ encodings).

## Remarks

Last edits: 21.12.2021