

1 Afterwarp Framework	1
2 Namespace Index	3
2.1 Namespace List	3
3 Class Index	5
3.1 Class List	5
4 File Index	9
4.1 File List	9
5 Namespace Documentation	11
5.1 pxt Namespace Reference	11
5.1.1 Typedef Documentation	17
5.1.1.1 ActorCamera_t	17
5.1.1.2 AmbientOcclusionParameters	17
5.1.1.3 Application_t	17
5.1.1.4 ApplicationConfiguration	17
5.1.1.5 ApplicationEvents	17
5.1.1.6 BoolFunc	17
5.1.1.7 Buffer_t	17
5.1.1.8 CameraConstraints	18
5.1.1.9 Canvas_t	18
5.1.1.10 CanvasAttributes	18
5.1.1.11 CanvasSamplerState	18
5.1.1.12 ColorDithering_t	18
5.1.1.13 ComputeBindTextureFormat	18
5.1.1.14 ComputeProgram_t	18
5.1.1.15 DeviceAttributes	19
5.1.1.16 DeviceBehavior	19
5.1.1.17 DeviceLegacyBits	19
5.1.1.18 EventFunc	19
5.1.1.19 EventHookFunc	19
5.1.1.20 FogParameters	19
5.1.1.21 FontEffect	19
5.1.1.22 FontParameters	19
5.1.1.23 GaussianBlur_t	20
5.1.1.24 Grapher_t	20
5.1.1.25 ImageAtlas_t	20
5.1.1.26 ImageRegion	20
5.1.1.27 KawaseBlur_t	20
5.1.1.28 KeyboardFunc	20
5.1.1.29 LibraryBool	20
5.1.1.30 MeshAligns	21

5.1.1.31 MeshBuffer_t	21
5.1.1.32 MeshBufferEntry	21
5.1.1.33 MeshLoadSaveFeedback	21
5.1.1.34 MeshMetaTag_t	21
5.1.1.35 MeshMetaTagPortion	21
5.1.1.36 MeshMetaTags_t	21
5.1.1.37 MeshModel_t	22
5.1.1.38 MeshVoxel_t	22
5.1.1.39 MeshVoxelVisualizeFunc	22
5.1.1.40 MouseFunc	22
5.1.1.41 ObjectMaterial	22
5.1.1.42 ObjectMaterials_t	22
5.1.1.43 ObjectModel_t	22
5.1.1.44 ObjectModelCompareFunc	23
5.1.1.45 ObjectModelID	23
5.1.1.46 ObjectModels_t	23
5.1.1.47 ObjectModelsIterator_t	23
5.1.1.48 ObjectModelView_t	23
5.1.1.49 ObjectPayload	23
5.1.1.50 OceanMaterial	23
5.1.1.51 OceanSimulation_t	24
5.1.1.52 OceanWavesParameters	24
5.1.1.53 ParallaxMappingParameters	24
5.1.1.54 Program_t	24
5.1.1.55 ProgramElement	24
5.1.1.56 ProgramVariable	24
5.1.1.57 RenderingState	24
5.1.1.58 Sampler_t	24
5.1.1.59 SamplerState	25
5.1.1.60 Scene_t	25
5.1.1.61 SceneAttributes	25
5.1.1.62 SceneLight	25
5.1.1.63 SceneLights_t	25
5.1.1.64 SceneMesh_t	25
5.1.1.65 SceneMeshes_t	25
5.1.1.66 SceneMeshLatch	25
5.1.1.67 SceneMeshLatches_t	26
5.1.1.68 SceneMeshMaterial_t	26
5.1.1.69 SceneMeshMaterialRange	26
5.1.1.70 SceneMeshMaterials_t	26
5.1.1.71 SceneMeshMaterialShading	26
5.1.1.72 SelectionHighlight_t	26

5.1.1.73 SelectionHighlightParameters	26
5.1.1.74 ShadowCaster_t	26
5.1.1.75 ShadowCastingAtlas_t	27
5.1.1.76 ShadowParameters	27
5.1.1.77 SignedDistanceField	27
5.1.1.78 SpatialFog_t	27
5.1.1.79 StatusFunc	27
5.1.1.80 Surface_t	27
5.1.1.81 TextEntryRect	27
5.1.1.82 TextModeller_t	27
5.1.1.83 TextRenderer_t	28
5.1.1.84 TextRenderModifiers	28
5.1.1.85 Texture_t	28
5.1.1.86 TextureAttributes	28
5.1.1.87 TextureCabinet_t	28
5.1.1.88 TextureCabinetAttributes	28
5.1.1.89 TextureParameters	28
5.1.1.90 Timer_t	28
5.1.1.91 ToneMappingBloom	29
5.1.1.92 UnicodeChar	29
5.1.1.93 UntypedHandle	29
5.1.1.94 VertexElement	29
5.1.1.95 Widget_t	29
5.1.1.96 WidgetExternalEvent	29
5.1.1.97 WidgetProperty	29
5.1.1.98 WidgetPropertyData	30
5.1.2 Function Documentation	30
5.1.2.1 deviceAttributeExtensions()	30
6 Class Documentation	31
6.1 AmbientOcclusionParameters Struct Reference	31
6.1.1 Detailed Description	31
6.1.2 Member Data Documentation	32
6.1.2.1 attenuation	32
6.1.2.2 blurFallOff	32
6.1.2.3 blurSigma	32
6.1.2.4 contrast	32
6.1.2.5 depthBias	32
6.1.2.6 kernelBias	32
6.1.2.7 radius	32
6.1.2.8 samples	33
6.1.2.9 strength	33

6.2 ApplicationConfiguration Struct Reference	33
6.2.1 Detailed Description	34
6.2.2 Member Data Documentation	34
6.2.2.1 handleIconBig	34
6.2.2.2 handleIconSmall	34
6.2.2.3 handleInstance	35
6.2.2.4 [struct]	35
6.2.2.5 parameterCount	35
6.2.2.6 parameterStrings	35
6.2.2.7 position	35
6.2.2.8 renderingType	35
6.2.2.9 size	35
6.2.2.10 [union]	35
6.2.2.11 state	36
6.2.2.12 [struct]	36
6.2.2.13 windowClassName	36
6.3 ApplicationEvents Struct Reference	36
6.3.1 Detailed Description	37
6.3.2 Member Data Documentation	37
6.3.2.1 eventCreate	37
6.3.2.2 eventDestroy	37
6.3.2.3 eventHook	37
6.3.2.4 eventIdle	38
6.3.2.5 eventKey	38
6.3.2.6 eventMouse	38
6.3.2.7 eventRender	38
6.3.2.8 eventResize	38
6.3.2.9 eventStatus	38
6.4 Blend Struct Reference	38
6.4.1 Detailed Description	39
6.4.2 Member Data Documentation	39
6.4.2.1 dest	39
6.4.2.2 op	39
6.4.2.3 source	39
6.5 CameraConstraints Struct Reference	40
6.5.1 Detailed Description	40
6.5.2 Member Data Documentation	40
6.5.2.1 positionMax	40
6.5.2.2 positionMin	41
6.5.2.3 rotationMax	41
6.5.2.4 rotationMin	41
6.6 CanvasSamplerState Struct Reference	41

6.6.1 Detailed Description	41
6.6.2 Member Data Documentation	42
6.6.2.1 addressU	42
6.6.2.2 addressV	42
6.6.2.3 borderColor	42
6.6.2.4 filterMag	42
6.6.2.5 filterMin	42
6.6.2.6 filterMip	42
6.7 ColorPair Struct Reference	42
6.7.1 Detailed Description	43
6.7.2 Constructor & Destructor Documentation	43
6.7.2.1 ColorPair() [1/2]	43
6.7.2.2 ColorPair() [2/2]	43
6.7.3 Member Data Documentation	43
6.7.3.1 first	43
6.7.3.2 second	44
6.8 ColorRect Struct Reference	44
6.8.1 Detailed Description	44
6.8.2 Member Data Documentation	44
6.8.2.1 bottomLeft	44
6.8.2.2 bottomRight	44
6.8.2.3 topLeft	45
6.8.2.4 topRight	45
6.9 ComputeBindTextureFormat Struct Reference	45
6.9.1 Detailed Description	45
6.9.2 Member Data Documentation	45
6.9.2.1 access	45
6.9.2.2 channel	46
6.9.2.3 format	46
6.9.2.4 layer	46
6.9.2.5 mipLevel	46
6.10 FloatColor Struct Reference	46
6.10.1 Detailed Description	47
6.10.2 Member Data Documentation	47
6.10.2.1 alpha	47
6.10.2.2 blue	47
6.10.2.3 green	47
6.10.2.4 red	47
6.11 FloatColorRGB Struct Reference	47
6.11.1 Detailed Description	48
6.11.2 Member Data Documentation	48
6.11.2.1 blue	48

6.11.2.2 green	48
6.11.2.3 red	48
6.12 FogParameters Struct Reference	48
6.12.1 Detailed Description	49
6.12.2 Member Data Documentation	49
6.12.2.1 [union]	49
6.12.2.2 [union]	49
6.12.2.3 bias	49
6.12.2.4 color	50
6.12.2.5 densityGround	50
6.12.2.6 distance	50
6.12.2.7 extinction	50
6.12.2.8 groundPlane	50
6.12.2.9 opacity	50
6.12.2.10 scattering	50
6.13 FontEffect Struct Reference	51
6.13.1 Detailed Description	51
6.13.2 Member Data Documentation	51
6.13.2.1 borderBrightness	51
6.13.2.2 borderOpacity	52
6.13.2.3 borderThickness	52
6.13.2.4 borderType	52
6.13.2.5 fillBrightness	52
6.13.2.6 fillOpacity	52
6.13.2.7 shadowBrightness	52
6.13.2.8 shadowDistance	52
6.13.2.9 shadowOpacity	53
6.13.2.10 shadowSmoothness	53
6.13.2.11 signedFieldDistance	53
6.14 FontParameters Struct Reference	53
6.14.1 Detailed Description	54
6.14.2 Constructor & Destructor Documentation	54
6.14.2.1 FontParameters()	54
6.14.3 Member Function Documentation	54
6.14.3.1 FontSettings()	54
6.14.4 Member Data Documentation	55
6.14.4.1 attributes	55
6.14.4.2 effect	55
6.14.4.3 family	55
6.14.4.4 size	55
6.14.4.5 slant	55
6.14.4.6 stretch	55

6.14.4.7 weight	55
6.15 ImageRegion Struct Reference	56
6.15.1 Detailed Description	56
6.15.2 Member Enumeration Documentation	56
6.15.2.1 anonymous enum	56
6.15.3 Member Data Documentation	57
6.15.3.1 height	57
6.15.3.2 index	57
6.15.3.3 left	57
6.15.3.4 top	57
6.15.3.5 width	57
6.16 Margins Struct Reference	57
6.16.1 Detailed Description	58
6.16.2 Member Data Documentation	58
6.16.2.1 bottom	58
6.16.2.2 left	58
6.16.2.3 right	58
6.16.2.4 top	58
6.17 Matrix Struct Reference	59
6.17.1 Detailed Description	59
6.17.2 Member Data Documentation	59
6.17.2.1 data	59
6.18 Matrix3x2 Struct Reference	59
6.18.1 Detailed Description	59
6.18.2 Member Data Documentation	60
6.18.2.1 data	60
6.19 MeshAligns Struct Reference	60
6.19.1 Detailed Description	60
6.19.2 Constructor & Destructor Documentation	60
6.19.2.1 MeshAligns() [1/2]	60
6.19.2.2 MeshAligns() [2/2]	61
6.19.3 Member Data Documentation	61
6.19.3.1 bias	61
6.19.3.2 x	61
6.19.3.3 y	61
6.19.3.4 z	61
6.20 MeshBufferEntry Struct Reference	62
6.20.1 Detailed Description	62
6.20.2 Member Data Documentation	62
6.20.2.1 color	62
6.20.2.2 normal	63
6.20.2.3 position	63

6.20.2.4 tangent	63
6.20.2.5 texCoord	63
6.21 MeshMetaTagPortion Struct Reference	63
6.21.1 Detailed Description	64
6.21.2 Member Data Documentation	64
6.21.2.1 boundsMax	64
6.21.2.2 boundsMin	64
6.21.2.3 firstIndex	64
6.21.2.4 firstVertex	64
6.21.2.5 indexCount	65
6.21.2.6 vertexCount	65
6.22 ObjectMaterial Struct Reference	65
6.22.1 Detailed Description	66
6.22.2 Member Data Documentation	66
6.22.2.1 albedoColor	66
6.22.2.2 ambientColor	66
6.22.2.3 bitmask	66
6.22.2.4 emissiveColor	67
6.22.2.5 frostedGlass	67
6.22.2.6 occlusion	67
6.22.2.7 payload	67
6.22.2.8 roughness	67
6.22.2.9 shadows	67
6.22.2.10 specularColor	67
6.22.2.11 specularExponent	67
6.22.2.12 technique	68
6.23 OceanMaterial Struct Reference	68
6.23.1 Detailed Description	69
6.23.2 Member Data Documentation	69
6.23.2.1 albedoColor	69
6.23.2.2 ambientColor	69
6.23.2.3 bitmask	69
6.23.2.4 emissiveColor	69
6.23.2.5 extinction	69
6.23.2.6 fresnel	69
6.23.2.7 shadows	70
6.23.2.8 specularColor	70
6.23.2.9 specularExponent	70
6.24 OceanWavesParameters Struct Reference	70
6.24.1 Detailed Description	71
6.24.2 Member Data Documentation	71
6.24.2.1 choppiness	71

6.24.2.2 resolution	71
6.24.2.3 waveSize	71
6.24.2.4 wind	71
6.25 ParallaxMappingParameters Struct Reference	72
6.25.1 Detailed Description	72
6.25.2 Member Data Documentation	72
6.25.2.1 occlusion	72
6.25.2.2 samplesMax	72
6.25.2.3 samplesMin	72
6.25.2.4 scale	73
6.26 PathElement Union Reference	73
6.26.1 Detailed Description	73
6.26.2 Constructor & Destructor Documentation	74
6.26.2.1 PathElement() [1/3]	74
6.26.2.2 PathElement() [2/3]	74
6.26.2.3 PathElement() [3/3]	74
6.26.3 Member Data Documentation	74
6.26.3.1 command	74
6.26.3.2 length	74
6.26.3.3 reserved	74
6.26.3.4 [struct]	75
6.26.3.5 value	75
6.27 Point Struct Reference	75
6.27.1 Detailed Description	75
6.27.2 Member Data Documentation	75
6.27.2.1 x	75
6.27.2.2 y	76
6.28 PointF Struct Reference	76
6.28.1 Detailed Description	76
6.28.2 Member Data Documentation	76
6.28.2.1 x	76
6.28.2.2 y	76
6.29 ProgramElement Struct Reference	77
6.29.1 Detailed Description	77
6.29.2 Member Data Documentation	77
6.29.2.1 channel	77
6.29.2.2 element	77
6.29.2.3 index	77
6.29.2.4 name	78
6.29.2.5 shader	78
6.29.2.6 size	78
6.30 ProgramVariable Struct Reference	78

6.30.1 Detailed Description	78
6.30.2 Member Data Documentation	79
6.30.2.1 count	79
6.30.2.2 format	79
6.30.2.3 index	79
6.30.2.4 name	79
6.30.2.5 offset	79
6.30.2.6 shader	79
6.30.2.7 size	79
6.31 Quad Struct Reference	80
6.31.1 Detailed Description	80
6.31.2 Member Data Documentation	80
6.31.2.1 bottomLeft	80
6.31.2.2 bottomRight	81
6.31.2.3 topLeft	81
6.31.2.4 topRight	81
6.32 Quaternion Struct Reference	81
6.32.1 Detailed Description	81
6.32.2 Member Data Documentation	82
6.32.2.1 w	82
6.32.2.2 x	82
6.32.2.3 y	82
6.32.2.4 z	82
6.33 Rect Struct Reference	82
6.33.1 Detailed Description	83
6.33.2 Member Data Documentation	83
6.33.2.1 height	83
6.33.2.2 left	83
6.33.2.3 top	83
6.33.2.4 width	83
6.34 RectF Struct Reference	83
6.34.1 Detailed Description	84
6.34.2 Member Data Documentation	84
6.34.2.1 height	84
6.34.2.2 left	84
6.34.2.3 top	84
6.34.2.4 width	84
6.35 RenderingState Struct Reference	85
6.35.1 Detailed Description	86
6.35.2 Member Data Documentation	86
6.35.2.1 blendAlpha	86
6.35.2.2 blendColor	86

6.35.2.3 blendConstant	86
6.35.2.4 clampDepthBias	86
6.35.2.5 cullFace	86
6.35.2.6 depthBias	86
6.35.2.7 depthFunc	87
6.35.2.8 slopeDepthBias	87
6.35.2.9 states	87
6.35.2.10 stencilBack	87
6.35.2.11 stencilFront	87
6.35.2.12 stencilRefMask	87
6.35.2.13 stencilRefValue	87
6.35.2.14 stencilWriteMask	88
6.36 SamplerState Struct Reference	88
6.36.1 Detailed Description	89
6.36.2 Member Data Documentation	89
6.36.2.1 address	89
6.36.2.2 anisotropy	89
6.36.2.3 biasLOD	89
6.36.2.4 borderColor	89
6.36.2.5 compareFunc	89
6.36.2.6 compareToRef	89
6.36.2.7 filterMag	90
6.36.2.8 filterMin	90
6.36.2.9 filterMip	90
6.36.2.10 maxLOD	90
6.36.2.11 minLOD	90
6.37 SceneLight Struct Reference	90
6.37.1 Detailed Description	91
6.37.2 Member Data Documentation	91
6.37.2.1 albedoColor	91
6.37.2.2 ambientColor	91
6.37.2.3 angle	92
6.37.2.4 angleCutoff	92
6.37.2.5 attenuationEnd	92
6.37.2.6 attenuationStart	92
6.37.2.7 bitmask	92
6.37.2.8 direction	92
6.37.2.9 intensity	92
6.37.2.10 payload	92
6.37.2.11 position	93
6.37.2.12 reserved	93
6.37.2.13 shadowCaster	93

6.37.2.14 specularColor	93
6.37.2.15 specularExponent	93
6.38 SceneMeshLatch Struct Reference	93
6.38.1 Detailed Description	94
6.38.2 Member Data Documentation	94
6.38.2.1 group	94
6.38.2.2 name	94
6.38.2.3 orientation	94
6.38.2.4 position	94
6.38.2.5 type	95
6.39 SceneMeshMaterialRange Struct Reference	95
6.39.1 Detailed Description	95
6.39.2 Member Data Documentation	95
6.39.2.1 indexCount	95
6.39.2.2 indexStart	95
6.40 SceneMeshMaterialShading Struct Reference	96
6.40.1 Detailed Description	96
6.40.2 Member Data Documentation	97
6.40.2.1 ambient	97
6.40.2.2 bloom	97
6.40.2.3 diffuse	97
6.40.2.4 emissive	97
6.40.2.5 lighting	97
6.40.2.6 roughness	97
6.40.2.7 specular	97
6.40.2.8 specularExponent	98
6.41 SelectionHighlightParameters Struct Reference	98
6.41.1 Detailed Description	98
6.41.2 Member Data Documentation	99
6.41.2.1 blurOffset	99
6.41.2.2 blurPasses	99
6.41.2.3 glowIntensity	99
6.41.2.4 outlineColor	99
6.41.2.5 outlineStart	99
6.42 ShadowParameters Struct Reference	99
6.42.1 Detailed Description	100
6.42.2 Member Data Documentation	100
6.42.2.1 bias	100
6.42.2.2 bleedSigma	100
6.42.2.3 blurSamples	100
6.42.2.4 blurSigma	100
6.42.2.5 exponent1	101

6.42.2.6 exponent2	101
6.42.2.7 variance	101
6.43 SignedDistanceField Struct Reference	101
6.43.1 Detailed Description	102
6.43.2 Member Data Documentation	102
6.43.2.1 outlineDistanceMaxSDF	102
6.43.2.2 outlineDistanceMinSDF	102
6.43.2.3 outlineOffsetSDF	102
6.43.2.4 signedFieldDistance	102
6.43.2.5 superSampleSDF	103
6.44 StencilState Struct Reference	103
6.44.1 Detailed Description	103
6.44.2 Member Data Documentation	103
6.44.2.1 depthFailOp	103
6.44.2.2 depthPassOp	103
6.44.2.3 failOp	104
6.44.2.4 func	104
6.45 TextEntryRect Struct Reference	104
6.45.1 Detailed Description	104
6.45.2 Member Data Documentation	105
6.45.2.1 position	105
6.45.2.2 rectangle	105
6.46 TextRenderModifiers Struct Reference	105
6.46.1 Detailed Description	105
6.46.2 Member Data Documentation	105
6.46.2.1 effect	105
6.46.2.2 interleave	106
6.46.2.3 scale	106
6.46.2.4 verticalSpace	106
6.47 TextureParameters Struct Reference	106
6.47.1 Detailed Description	107
6.47.2 Member Data Documentation	107
6.47.2.1 attributes	107
6.47.2.2 depthStencil	107
6.47.2.3 format	107
6.47.2.4 height	107
6.47.2.5 layers	107
6.47.2.6 multisamples	107
6.47.2.7 type	108
6.47.2.8 width	108
6.48 ToneMappingBloom Struct Reference	108
6.48.1 Detailed Description	109

6.48.2 Member Data Documentation	109
6.48.2.1 bloomBlurSamples	109
6.48.2.2 bloomBlurSigma	109
6.48.2.3 bloomCoefficients	109
6.48.2.4 bloomColorShift	110
6.48.2.5 bloomGamma	110
6.48.2.6 bloomThreshold	110
6.48.2.7 frostedPower	110
6.48.2.8 glassyBuckets	110
6.48.2.9 toneFactors	110
6.48.2.10 toneWhite	110
6.49 Vector Struct Reference	111
6.49.1 Detailed Description	111
6.49.2 Member Data Documentation	111
6.49.2.1 x	111
6.49.2.2 y	111
6.49.2.3 z	111
6.50 Vector4 Struct Reference	112
6.50.1 Detailed Description	112
6.50.2 Member Data Documentation	112
6.50.2.1 w	112
6.50.2.2 x	112
6.50.2.3 y	112
6.50.2.4 z	113
6.51 VertexElement Struct Reference	113
6.51.1 Detailed Description	113
6.51.2 Member Data Documentation	113
6.51.2.1 channel	113
6.51.2.2 count	113
6.51.2.3 format	114
6.51.2.4 name	114
6.51.2.5 offset	114
6.52 WidgetProperty Struct Reference	114
6.52.1 Detailed Description	115
6.52.2 Member Data Documentation	115
6.52.2.1 attributes	115
6.52.2.2 behavior	115
6.52.2.3 data	115
6.52.2.4 location	115
6.52.2.5 name	116
6.52.2.6 reserved	116
6.52.2.7 type	116

6.53 WidgetPropertyData Union Reference	116
6.53.1 Detailed Description	117
6.53.2 Member Data Documentation	117
6.53.2.1 valueBool	117
6.53.2.2 valueColor	117
6.53.2.3 valueColorPair	118
6.53.2.4 valueColorRect	118
6.53.2.5 valueEnum	118
6.53.2.6 valueFloat	118
6.53.2.7 valueFont	118
6.53.2.8 valueInt64	118
6.53.2.9 valueInteger	118
6.53.2.10 valueMargins	118
6.53.2.11 valuePoint	119
6.53.2.12 valueReal	119
6.53.2.13 valueRect	119
6.53.2.14 valueString	119
6.53.2.15 valueVector	119
7 File Documentation	121
7.1 Afterwarp.h File Reference	121
7.1.1 Function Documentation	163
7.1.1.1 ActorCameraCommand()	163
7.1.1.2 ActorCameraCreate()	163
7.1.1.3 ActorCameraDestroy()	163
7.1.1.4 ActorCameraGetCeiling()	163
7.1.1.5 ActorCameraGetConstraints()	163
7.1.1.6 ActorCameraGetDistance()	164
7.1.1.7 ActorCameraGetForward()	164
7.1.1.8 ActorCameraGetPosition()	164
7.1.1.9 ActorCameraGetQuaternion()	164
7.1.1.10 ActorCameraGetRight()	164
7.1.1.11 ActorCameraGetRotation()	164
7.1.1.12 ActorCameraGetView()	165
7.1.1.13 ActorCameraSetConstraints()	165
7.1.1.14 ActorCameraSetDistance()	165
7.1.1.15 ActorCameraSetPosition()	165
7.1.1.16 ActorCameraSetQuaternion()	165
7.1.1.17 ActorCameraSetRotation()	165
7.1.1.18 ActorCameraZoom()	166
7.1.1.19 ActorCameraZoomOrtho()	166
7.1.1.20 AddColors()	166

7.1.1.21 ApplicationCaptureMouseInput()	166
7.1.1.22 ApplicationConvertPortableKey()	166
7.1.1.23 ApplicationCreate()	167
7.1.1.24 ApplicationDestroy()	167
7.1.1.25 ApplicationExecute()	167
7.1.1.26 ApplicationFileChooserDialog()	167
7.1.1.27 ApplicationGetClientRect()	167
7.1.1.28 ApplicationGetCursor()	167
7.1.1.29 ApplicationGetExecutablePath()	168
7.1.1.30 ApplicationGetIconTitle()	168
7.1.1.31 ApplicationGetMinimalSize()	168
7.1.1.32 ApplicationGetTitle()	168
7.1.1.33 ApplicationGetWindowHandle()	168
7.1.1.34 ApplicationGetWindowRect()	169
7.1.1.35 ApplicationGetWindowScale()	169
7.1.1.36 ApplicationGetWindowState()	169
7.1.1.37 ApplicationInvalidate()	169
7.1.1.38 ApplicationMouseInputCaptured()	169
7.1.1.39 ApplicationReadTextFromClipboard()	169
7.1.1.40 ApplicationReleaseMouseInput()	170
7.1.1.41 ApplicationSetClientSize()	170
7.1.1.42 ApplicationSetCursor()	170
7.1.1.43 ApplicationSetEvents()	170
7.1.1.44 ApplicationSetIcons()	170
7.1.1.45 ApplicationSetIconsFromFiles()	170
7.1.1.46 ApplicationSetIconTitle()	171
7.1.1.47 ApplicationSetMinimalSize()	171
7.1.1.48 ApplicationSetTitle()	171
7.1.1.49 ApplicationSetWindowRect()	171
7.1.1.50 ApplicationSetWindowState()	171
7.1.1.51 ApplicationTerminate()	171
7.1.1.52 ApplicationTranslateVirtualKey()	172
7.1.1.53 ApplicationWriteTextToClipboard()	172
7.1.1.54 AverageColors()	172
7.1.1.55 AverageFourColors()	172
7.1.1.56 AverageSixColors()	172
7.1.1.57 BiasTransform()	173
7.1.1.58 BlendColors()	173
7.1.1.59 BlendFourColors()	173
7.1.1.60 BufferCopy()	173
7.1.1.61 BufferCreate()	173
7.1.1.62 BufferDestroy()	174

7.1.1.63 BufferGetAccessType()	174
7.1.1.64 BufferGetDataType()	174
7.1.1.65 BufferGetDevice()	174
7.1.1.66 BufferGetFormat()	174
7.1.1.67 BufferGetPitch()	174
7.1.1.68 BufferGetPlatformHandle()	174
7.1.1.69 BufferGetSize()	175
7.1.1.70 BufferRetrieve()	175
7.1.1.71 BufferUpdate()	175
7.1.1.72 CanvasArc()	175
7.1.1.73 CanvasArcGrad()	176
7.1.1.74 CanvasBegin()	176
7.1.1.75 CanvasBufferClear()	176
7.1.1.76 CanvasBufferClearAndShrink()	176
7.1.1.77 CanvasBufferCreate()	176
7.1.1.78 CanvasBufferDestroy()	176
7.1.1.79 CanvasBufferGetColors()	177
7.1.1.80 CanvasBufferGetExtents()	177
7.1.1.81 CanvasBufferGetIndexCount()	177
7.1.1.82 CanvasBufferGetIndices()	177
7.1.1.83 CanvasBufferGetVertexCount()	177
7.1.1.84 CanvasBufferGetVertices()	177
7.1.1.85 CanvasBufferSetIndexCount()	177
7.1.1.86 CanvasBufferSetVertexCount()	178
7.1.1.87 CanvasCreate()	178
7.1.1.88 CanvasDestroy()	178
7.1.1.89 CanvasDrawBuffer()	178
7.1.1.90 CanvasEllipse()	178
7.1.1.91 CanvasEnd()	178
7.1.1.92 CanvasFillRect()	179
7.1.1.93 CanvasFillRoundRect()	179
7.1.1.94 CanvasFillRoundRectBottom()	179
7.1.1.95 CanvasFillRoundRectTop()	179
7.1.1.96 CanvasFillRoundRectTopInverse()	180
7.1.1.97 CanvasFlush()	180
7.1.1.98 CanvasFrameRect()	180
7.1.1.99 CanvasFrameRoundRect()	180
7.1.1.100 CanvasGetAttributes()	180
7.1.1.101 CanvasGetBatchCount()	181
7.1.1.102 CanvasGetClipRect()	181
7.1.1.103 CanvasGetContextState()	181
7.1.1.104 CanvasGetDevice()	181

7.1.1.105 CanvasGetSamplerState()	181
7.1.1.106 CanvasGetSignedDistanceField()	181
7.1.1.107 CanvasGetTransform()	182
7.1.1.108 CanvasGetViewProjection()	182
7.1.1.109 CanvasGetWorld()	182
7.1.1.110 CanvasHexagon()	182
7.1.1.111 CanvasHexagonGrad()	182
7.1.1.112 CanvasHighlight()	183
7.1.1.113 CanvasLine()	183
7.1.1.114 CanvasLineCircle()	183
7.1.1.115 CanvasLineEllipse()	183
7.1.1.116 CanvasLineHexagon()	184
7.1.1.117 CanvasLineHexagonGrad()	184
7.1.1.118 CanvasLineQuad()	184
7.1.1.119 CanvasLines()	184
7.1.1.120 CanvasLineTriangle()	185
7.1.1.121 CanvasPixel()	185
7.1.1.122 CanvasPixels()	185
7.1.1.123 CanvasQuad()	185
7.1.1.124 CanvasQuadImage()	185
7.1.1.125 CanvasRectWithHole()	186
7.1.1.126 CanvasReset()	186
7.1.1.127 CanvasResetSamplerState()	186
7.1.1.128 CanvasRibbon()	186
7.1.1.129 CanvasRibbonGrad()	186
7.1.1.130 CanvasRibbonTri()	187
7.1.1.131 CanvasSetAttributes()	187
7.1.1.132 CanvasSetClipRect()	187
7.1.1.133 CanvasSetContextState()	187
7.1.1.134 CanvasSetSamplerState()	187
7.1.1.135 CanvasSetSignedDistanceField()	188
7.1.1.136 CanvasSetTransform()	188
7.1.1.137 CanvasSetViewProjection()	188
7.1.1.138 CanvasSetWorld()	188
7.1.1.139 CanvasTape()	188
7.1.1.140 CanvasTapeGrad()	189
7.1.1.141 CanvasTapeTri()	189
7.1.1.142 CanvasTexturedQuad()	189
7.1.1.143 CanvasTexturedQuadRegion()	189
7.1.1.144 CanvasTexturedRoundRect()	190
7.1.1.145 CanvasTexturedRoundRectRegion()	190
7.1.1.146 CanvasTexturedTriangle()	190

7.1.1.147 CanvasTexturedTriangleRegion()	191
7.1.1.148 CanvasTexturedTriangles()	191
7.1.1.149 CanvasThickLine()	191
7.1.1.150 CanvasThickLineCircle()	191
7.1.1.151 CanvasThickLineEllipse()	192
7.1.1.152 CanvasThickLineHexagon()	192
7.1.1.153 CanvasThickLineHexagonGrad()	192
7.1.1.154 CanvasThickLineQuad()	192
7.1.1.155 CanvasThickLineTriangle()	193
7.1.1.156 CanvasTriangle()	193
7.1.1.157 CanvasTriangles()	193
7.1.1.158 CatmullRom()	193
7.1.1.159 ColorDitheringCreate()	194
7.1.1.160 ColorDitheringDestroy()	194
7.1.1.161 ColorDitheringExecute()	194
7.1.1.162 ColorDitheringExecuteTo()	194
7.1.1.163 ColorDitheringGetDevice()	194
7.1.1.164 ColorDitheringGetFormat()	194
7.1.1.165 ColorDitheringSetFormat()	195
7.1.1.166 ColorToGray()	195
7.1.1.167 ColorToGray16()	195
7.1.1.168 ColorToGrayF()	195
7.1.1.169 ComposeColors()	195
7.1.1.170 ComputeProgramBegin()	195
7.1.1.171 ComputeProgramBindBuffer()	196
7.1.1.172 ComputeProgramBindTexture()	196
7.1.1.173 ComputeProgramCommit()	196
7.1.1.174 ComputeProgramCreate()	196
7.1.1.175 ComputeProgramDestroy()	196
7.1.1.176 ComputeProgramDispatch()	197
7.1.1.177 ComputeProgramEnd()	197
7.1.1.178 ComputeProgramGetDevice()	197
7.1.1.179 ComputeProgramResetBindings()	197
7.1.1.180 ComputeProgramUnbindBuffer()	197
7.1.1.181 ComputeProgramUnbindTexture()	197
7.1.1.182 Cubic()	198
7.1.1.183 DeviceClear()	198
7.1.1.184 DeviceCreate()	198
7.1.1.185 DeviceCreateWrapped()	198
7.1.1.186 DeviceDestroy()	198
7.1.1.187 DeviceGetAttributes()	199
7.1.1.188 DeviceGetBehavior()	199

7.1.1.189 DeviceGetLegacyBits()	199
7.1.1.190 DeviceGetPlatform()	199
7.1.1.191 DeviceGetPlatformDevice()	199
7.1.1.192 DeviceGetRenderingState()	199
7.1.1.193 DeviceGetScissor()	199
7.1.1.194 DeviceGetTechFeatureVersion()	200
7.1.1.195 DeviceGetTechnology()	200
7.1.1.196 DeviceGetTechVersion()	200
7.1.1.197 DeviceGetViewport()	200
7.1.1.198 DeviceMemoryBarrier()	200
7.1.1.199 DeviceResetCache()	200
7.1.1.200 DeviceSetRenderingState()	201
7.1.1.201 DeviceSetScissor()	201
7.1.1.202 DeviceSetViewport()	201
7.1.1.203 DisplaceRB()	201
7.1.1.204 GainTransform()	201
7.1.1.205 GaussianBlurCreate()	201
7.1.1.206 GaussianBlurDestroy()	202
7.1.1.207 GaussianBlurGetChroma()	202
7.1.1.208 GaussianBlurGetDevice()	202
7.1.1.209 GaussianBlurGetFixedSamples()	202
7.1.1.210 GaussianBlurGetHardwareFiltering()	202
7.1.1.211 GaussianBlurGetSamples()	202
7.1.1.212 GaussianBlurGetSigma()	202
7.1.1.213 GaussianBlurSetChroma()	203
7.1.1.214 GaussianBlurSetHardwareFiltering()	203
7.1.1.215 GaussianBlurSetSamples()	203
7.1.1.216 GaussianBlurSetSigma()	203
7.1.1.217 GaussianBlurUpdate()	203
7.1.1.218 GaussianBlurUpdateAt()	204
7.1.1.219 GetColorAlpha()	204
7.1.1.220 GetColorAlphaF()	204
7.1.1.221 GetSystemTicks()	204
7.1.1.222 GrapherArrow()	204
7.1.1.223 GrapherBegin()	205
7.1.1.224 GrapherBoundingBox()	205
7.1.1.225 GrapherCreate()	205
7.1.1.226 GrapherDestroy()	205
7.1.1.227 GrapherDottedLine()	205
7.1.1.228 GrapherEnd()	206
7.1.1.229 GrapherFlush()	206
7.1.1.230 GrapherGetBatchCount()	206

7.1.1.231 GrapherGetDevice()	206
7.1.1.232 GrapherGetTransform()	206
7.1.1.233 GrapherLine()	206
7.1.1.234 GrapherLines()	207
7.1.1.235 GrapherPoint()	207
7.1.1.236 GrapherPoints()	207
7.1.1.237 GrapherReset()	207
7.1.1.238 GrapherSetTransform()	207
7.1.1.239 Hermite()	208
7.1.1.240 ImageAtlasClearRegions()	208
7.1.1.241 ImageAtlasClearTextures()	208
7.1.1.242 ImageAtlasCreate()	208
7.1.1.243 ImageAtlasCreateRegion()	208
7.1.1.244 ImageAtlasCreateTexture()	209
7.1.1.245 ImageAtlasDestroy()	209
7.1.1.246 ImageAtlasGetDevice()	209
7.1.1.247 ImageAtlasMakeRegions()	209
7.1.1.248 ImageAtlasPackRegion()	209
7.1.1.249 ImageAtlasPackSurface()	210
7.1.1.250 ImageAtlasRegion()	210
7.1.1.251 ImageAtlasRegionCount()	210
7.1.1.252 ImageAtlasRemoveRegion()	210
7.1.1.253 ImageAtlasRemoveTexture()	210
7.1.1.254 ImageAtlasTexture()	210
7.1.1.255 ImageAtlasTextureCount()	211
7.1.1.256 InvertColor()	211
7.1.1.257 KawaseBlurCreate()	211
7.1.1.258 KawaseBlurDestroy()	211
7.1.1.259 KawaseBlurGetDevice()	211
7.1.1.260 KawaseBlurGetOffset()	211
7.1.1.261 KawaseBlurGetPasses()	211
7.1.1.262 KawaseBlurSetOffset()	212
7.1.1.263 KawaseBlurSetPasses()	212
7.1.1.264 KawaseBlurSinglePass()	212
7.1.1.265 KawaseBlurUpdate()	212
7.1.1.266 Lerp()	212
7.1.1.267 LibraryGetVersion()	213
7.1.1.268 LibrarySerialCode()	213
7.1.1.269 MakeColor()	213
7.1.1.270 MakeColorAlpha()	213
7.1.1.271 MakeColorAlphaF()	213
7.1.1.272 MakeColorF()	213

7.1.1.273 MakeColorGray()	214
7.1.1.274 MakeColorGrayF()	214
7.1.1.275 MakeColorRGB()	214
7.1.1.276 MakeColorRGBF()	214
7.1.1.277 MakeColorWithGray()	214
7.1.1.278 MakeColorWithGrayF()	215
7.1.1.279 MeshBoundsTagOffset()	215
7.1.1.280 MeshBoundsToMatrixModel()	215
7.1.1.281 MeshBoundsToMatrixVolume()	215
7.1.1.282 MeshBoundsToMatrixVolumeTag()	215
7.1.1.283 MeshBufferCalculateBounds()	216
7.1.1.284 MeshBufferCalculateFlatNormals()	216
7.1.1.285 MeshBufferCalculateNormals()	216
7.1.1.286 MeshBufferCalculateNormalsWeld()	216
7.1.1.287 MeshBufferCentralize()	216
7.1.1.288 MeshBufferClear()	217
7.1.1.289 MeshBufferCombine()	217
7.1.1.290 MeshBufferConeSimple()	217
7.1.1.291 MeshBufferCreate()	217
7.1.1.292 MeshBufferCreateModel()	217
7.1.1.293 MeshBufferCube()	218
7.1.1.294 MeshBufferCubeMinimal()	218
7.1.1.295 MeshBufferCubeRound()	218
7.1.1.296 MeshBufferCylinder()	218
7.1.1.297 MeshBufferDestroy()	219
7.1.1.298 MeshBufferDisc()	219
7.1.1.299 MeshBufferEliminateUnusedVertices()	219
7.1.1.300 MeshBufferExtrusion()	220
7.1.1.301 MeshBufferFrustumVolume()	220
7.1.1.302 MeshBufferGeosphere()	220
7.1.1.303 MeshBufferGetIndexCount()	220
7.1.1.304 MeshBufferGetIndices()	220
7.1.1.305 MeshBufferGetTransform()	221
7.1.1.306 MeshBufferGetVertexCount()	221
7.1.1.307 MeshBufferGetVertices()	221
7.1.1.308 MeshBufferInvertIndexOrder()	221
7.1.1.309 MeshBufferInvertNormals()	221
7.1.1.310 MeshBufferJoinDuplicateVertices()	221
7.1.1.311 MeshBufferLoadFromFile()	222
7.1.1.312 MeshBufferLoadFromFileEx()	222
7.1.1.313 MeshBufferPlane()	222
7.1.1.314 MeshBufferSaveToFile()	223

7.1.1.315 MeshBufferSaveToFileEx()	223
7.1.1.316 MeshBufferSetIndexCount()	223
7.1.1.317 MeshBufferSetTransform()	223
7.1.1.318 MeshBufferSetVertexCount()	223
7.1.1.319 MeshBufferSuperEllipse()	224
7.1.1.320 MeshBufferSupertoroid()	224
7.1.1.321 MeshBufferTorus()	224
7.1.1.322 MeshBufferTorusKnot()	225
7.1.1.323 MeshBufferTransferIndexElements()	225
7.1.1.324 MeshBufferTransferIndices()	226
7.1.1.325 MeshBufferTransferVertexElements()	226
7.1.1.326 MeshBufferTransferVertices()	226
7.1.1.327 MeshBufferTransformVertices()	226
7.1.1.328 MeshBufferVoxelize()	227
7.1.1.329 MeshMetaTagGetBounds()	227
7.1.1.330 MeshMetaTagGetName()	227
7.1.1.331 MeshMetaTagGetParent()	227
7.1.1.332 MeshMetaTagGetPortionCount()	227
7.1.1.333 MeshMetaTagGetType()	227
7.1.1.334 MeshMetaTagPortionAdd()	228
7.1.1.335 MeshMetaTagPortionErase()	228
7.1.1.336 MeshMetaTagPortionGet()	228
7.1.1.337 MeshMetaTagPortionsClear()	228
7.1.1.338 MeshMetaTagPortionsCopy()	228
7.1.1.339 MeshMetaTagsAdd()	228
7.1.1.340 MeshMetaTagsClear()	229
7.1.1.341 MeshMetaTagsCopy()	229
7.1.1.342 MeshMetaTagsCount()	229
7.1.1.343 MeshMetaTagsCreate()	229
7.1.1.344 MeshMetaTagsDestroy()	229
7.1.1.345 MeshMetaTagsErase()	229
7.1.1.346 MeshMetaTagsGetByIndex()	230
7.1.1.347 MeshMetaTagsGetByName()	230
7.1.1.348 MeshMetaTagsTakeAway()	230
7.1.1.349 MeshModelCreate()	230
7.1.1.350 MeshModelDestroy()	230
7.1.1.351 MeshModelDismantle()	230
7.1.1.352 MeshModelDraw()	231
7.1.1.353 MeshModelDrawInstances()	231
7.1.1.354 MeshModelGetChannel()	231
7.1.1.355 MeshModelGetIndexBuffer()	231
7.1.1.356 MeshModelGetIndexCount()	231

7.1.1.357 MeshModelGetVertexBuffer()	232
7.1.1.358 MeshModelGetVertexCount()	232
7.1.1.359 MeshModelRenderable()	232
7.1.1.360 MeshModelTakeAway()	232
7.1.1.361 MeshVoxelComputeParameters()	232
7.1.1.362 MeshVoxelCreate()	232
7.1.1.363 MeshVoxelDestroy()	233
7.1.1.364 MeshVoxelExtents()	233
7.1.1.365 MeshVoxelIntersect()	233
7.1.1.366 MeshVoxelLoadFromFile()	233
7.1.1.367 MeshVoxelLoadFromFileInMemory()	233
7.1.1.368 MeshVoxelSaveToFile()	234
7.1.1.369 MeshVoxelTakeAway()	234
7.1.1.370 MeshVoxelVisualize()	234
7.1.1.371 MultiplyColors()	234
7.1.1.372 ObjectMaterialsAdd()	234
7.1.1.373 ObjectMaterialsClear()	234
7.1.1.374 ObjectMaterialsCreate()	235
7.1.1.375 ObjectMaterialsDestroy()	235
7.1.1.376 ObjectMaterialsErase()	235
7.1.1.377 ObjectMaterialsGetCount()	235
7.1.1.378 ObjectMaterialsGetElement()	235
7.1.1.379 ObjectModelConnectLatches()	235
7.1.1.380 ObjectModelConnectLatchesByName()	236
7.1.1.381 ObjectModelGetAABB()	236
7.1.1.382 ObjectModelGetAlignments()	236
7.1.1.383 ObjectModelGetAttributes()	236
7.1.1.384 ObjectModelGetChild()	236
7.1.1.385 ObjectModelGetChildCount()	236
7.1.1.386 ObjectModelGetColor()	237
7.1.1.387 ObjectModelGetConnectedLatches()	237
7.1.1.388 ObjectModelGetDepthBias()	237
7.1.1.389 ObjectModelGetDescription()	237
7.1.1.390 ObjectModelGetHighlight()	237
7.1.1.391 ObjectModelGetID()	237
7.1.1.392 ObjectModelGetLatchTransform()	238
7.1.1.393 ObjectModelGetLatchTransformByName()	238
7.1.1.394 ObjectModelGetLatchWaypointCouple()	238
7.1.1.395 ObjectModelGetLatchWaypointCoupleMatrix()	238
7.1.1.396 ObjectModelGetLayers()	238
7.1.1.397 ObjectModelGetMaterial()	239
7.1.1.398 ObjectModelGetMesh()	239

7.1.1.399 ObjectModelGetMeshName()	239
7.1.1.400 ObjectModelGetName()	239
7.1.1.401 ObjectModelGetNext()	239
7.1.1.402 ObjectModelGetOrderIndex()	239
7.1.1.403 ObjectModelGetOwner()	240
7.1.1.404 ObjectModelGetParent()	240
7.1.1.405 ObjectModelGetPayload()	240
7.1.1.406 ObjectModelGetPosition()	240
7.1.1.407 ObjectModelGetSize()	240
7.1.1.408 ObjectModelGetTransform()	240
7.1.1.409 ObjectModelGetVoxel()	241
7.1.1.410 ObjectModelGetWaypointDistance()	241
7.1.1.411 ObjectModelInvalidate()	241
7.1.1.412 ObjectModelsAdd()	241
7.1.1.413 ObjectModelsAddWithID()	241
7.1.1.414 ObjectModelsClear()	241
7.1.1.415 ObjectModelsClearViews()	242
7.1.1.416 ObjectModelsCreate()	242
7.1.1.417 ObjectModelsCreateView()	242
7.1.1.418 ObjectModelsDestroy()	242
7.1.1.419 ObjectModelsErase()	242
7.1.1.420 ObjectModelsEraseView()	242
7.1.1.421 ObjectModelSetAlignments()	243
7.1.1.422 ObjectModelSetAttributes()	243
7.1.1.423 ObjectModelSetColor()	243
7.1.1.424 ObjectModelSetDepthBias()	243
7.1.1.425 ObjectModelSetDescription()	243
7.1.1.426 ObjectModelSetHighlight()	243
7.1.1.427 ObjectModelSetLayers()	244
7.1.1.428 ObjectModelSetMaterial()	244
7.1.1.429 ObjectModelSetMesh()	244
7.1.1.430 ObjectModelSetMeshName()	244
7.1.1.431 ObjectModelSetName()	244
7.1.1.432 ObjectModelSetOrderIndex()	244
7.1.1.433 ObjectModelSetParent()	245
7.1.1.434 ObjectModelSetSize()	245
7.1.1.435 ObjectModelSetTransform()	245
7.1.1.436 ObjectModelSetVoxel()	245
7.1.1.437 ObjectModelsGetFirst()	245
7.1.1.438 ObjectModelsGetMeshes()	245
7.1.1.439 ObjectModelsGetObjectByID()	246
7.1.1.440 ObjectModelsGetObjectByName()	246

7.1.1.441 ObjectModelsGetObjectCount()	246
7.1.1.442 ObjectModelsPayload()	246
7.1.1.443 ObjectModelViewAutoDraw()	246
7.1.1.444 ObjectModelViewGetIntersectedObjects()	246
7.1.1.445 ObjectModelViewGetIntersectedRays()	247
7.1.1.446 ObjectModelViewGetLayers()	247
7.1.1.447 ObjectModelViewGetObject()	247
7.1.1.448 ObjectModelViewGetObjectCount()	247
7.1.1.449 ObjectModelViewGetObjectsNotCulled()	247
7.1.1.450 ObjectModelViewGetOwner()	247
7.1.1.451 ObjectModelViewGetProjection()	248
7.1.1.452 ObjectModelViewGetView()	248
7.1.1.453 ObjectModelViewGetViewProjection()	248
7.1.1.454 ObjectModelViewInvalidate()	248
7.1.1.455 ObjectModelViewSelect()	248
7.1.1.456 ObjectModelViewSelectAny()	248
7.1.1.457 ObjectModelViewSetLayers()	249
7.1.1.458 ObjectModelViewSetProjection()	249
7.1.1.459 ObjectModelViewSetView()	249
7.1.1.460 ObjectModelViewSort()	249
7.1.1.461 ObjectModelViewSortWith()	249
7.1.1.462 ObjectModelViewUpdate()	249
7.1.1.463 ObjectModelViewUpdateNeeded()	250
7.1.1.464 OceanSimulationCreate()	250
7.1.1.465 OceanSimulationDestroy()	250
7.1.1.466 OceanSimulationGetAttributes()	250
7.1.1.467 OceanSimulationGetDevice()	250
7.1.1.468 OceanSimulationGetMaterial()	250
7.1.1.469 OceanSimulationGetPlane()	250
7.1.1.470 OceanSimulationGetProjection()	251
7.1.1.471 OceanSimulationGetSamplerShadow()	251
7.1.1.472 OceanSimulationGetScale()	251
7.1.1.473 OceanSimulationGetSections()	251
7.1.1.474 OceanSimulationGetView()	251
7.1.1.475 OceanSimulationGetViewDistance()	251
7.1.1.476 OceanSimulationGetWavesParameters()	252
7.1.1.477 OceanSimulationGetWavesTexture()	252
7.1.1.478 OceanSimulationRender()	252
7.1.1.479 OceanSimulationSetAttributes()	252
7.1.1.480 OceanSimulationSetMaterial()	252
7.1.1.481 OceanSimulationSetPlane()	252
7.1.1.482 OceanSimulationSetProjection()	253

7.1.1.483 OceanSimulationSetScale()	253
7.1.1.484 OceanSimulationSetSections()	253
7.1.1.485 OceanSimulationSetView()	253
7.1.1.486 OceanSimulationSetViewDistance()	253
7.1.1.487 OceanSimulationSetWavesParameters()	253
7.1.1.488 OceanSimulationUpdate()	254
7.1.1.489 PathBrokerCreate()	254
7.1.1.490 PathBrokerDestroy()	254
7.1.1.491 PathBrokerFill()	254
7.1.1.492 PathBrokerReset()	254
7.1.1.493 PathBrokerStroke()	254
7.1.1.494 PixelFormatBits()	255
7.1.1.495 PixelFormatConvert()	255
7.1.1.496 PixelFormatConvertArray()	255
7.1.1.497 PixelFormatValid()	255
7.1.1.498 PremultiplyAlpha()	255
7.1.1.499 ProgramBegin()	255
7.1.1.500 ProgramBind()	256
7.1.1.501 ProgramCommit()	256
7.1.1.502 ProgramCreate()	256
7.1.1.503 ProgramDestroy()	256
7.1.1.504 ProgramDraw()	257
7.1.1.505 ProgramDrawIndexed()	257
7.1.1.506 ProgramDrawInstances()	257
7.1.1.507 ProgramDrawInstancesIndexed()	257
7.1.1.508 ProgramEnd()	257
7.1.1.509 ProgramGetDevice()	258
7.1.1.510 ProgramResetBindings()	258
7.1.1.511 ProgramResetCache()	258
7.1.1.512 ProgramSetPatchVertices()	258
7.1.1.513 ProgramUnbind()	258
7.1.1.514 ProgramUpdateByIndex()	258
7.1.1.515 ProgramUpdateByName()	259
7.1.1.516 RandomSequenceGenerate()	259
7.1.1.517 RandomSequenceGenerate64()	259
7.1.1.518 RandomSequenceGenerateDouble()	259
7.1.1.519 RandomSequenceGenerateFloat()	259
7.1.1.520 RandomSequenceGenerateGaussian()	259
7.1.1.521 RandomSequenceGenerateRanged()	260
7.1.1.522 RandomSequenceInit()	260
7.1.1.523 RandomSequenceInitBySeed()	260
7.1.1.524 RayCreate()	260

7.1.1.525 RayIntersectCubeVolume()	260
7.1.1.526 RayIntersectPlane()	261
7.1.1.527 RayIntersectTriangle()	261
7.1.1.528 SamplerBind()	261
7.1.1.529 SamplerCreate()	261
7.1.1.530 SamplerDestroy()	261
7.1.1.531 SamplerGetDevice()	262
7.1.1.532 SamplerGetState()	262
7.1.1.533 SamplerSetState()	262
7.1.1.534 SamplerUnbind()	262
7.1.1.535 SceneBegin()	262
7.1.1.536 SceneCreateDepthsNormals()	262
7.1.1.537 SceneCreateModeling()	263
7.1.1.538 SceneDestroy()	263
7.1.1.539 SceneEnd()	263
7.1.1.540 SceneGetActiveVertexElements()	263
7.1.1.541 SceneGetAttributes()	263
7.1.1.542 SceneGetDevice()	263
7.1.1.543 SceneGetInstancesCount()	263
7.1.1.544 SceneGetLights()	264
7.1.1.545 SceneGetMaterial()	264
7.1.1.546 SceneGetParallaxMappingParameters()	264
7.1.1.547 SceneGetProgram()	264
7.1.1.548 SceneGetProjection()	264
7.1.1.549 SceneGetSampler()	264
7.1.1.550 SceneGetShadowCastingAtlas()	265
7.1.1.551 SceneGetTexture()	265
7.1.1.552 SceneGetTextureCabinet()	265
7.1.1.553 SceneGetToneMappingCoefficients()	265
7.1.1.554 SceneGetVertexElementCount()	265
7.1.1.555 SceneGetVertexElements()	265
7.1.1.556 SceneGetView()	266
7.1.1.557 SceneGetWorld()	266
7.1.1.558 SceneInstances()	266
7.1.1.559 SceneLightsAdd()	266
7.1.1.560 SceneLightsClear()	266
7.1.1.561 SceneLightsCreate()	266
7.1.1.562 SceneLightsDestroy()	267
7.1.1.563 SceneLightsErase()	267
7.1.1.564 SceneLightsExecute()	267
7.1.1.565 SceneLightsGetClusters()	267
7.1.1.566 SceneLightsGetClusterSize()	267

7.1.1.567 SceneLightsGetCount()	267
7.1.1.568 SceneLightsGetCullingMode()	268
7.1.1.569 SceneLightsGetDepthSlices()	268
7.1.1.570 SceneLightsGetDevice()	268
7.1.1.571 SceneLightsGetElement()	268
7.1.1.572 SceneLightsGetIndices()	268
7.1.1.573 SceneLightsGetViewSize()	268
7.1.1.574 SceneLightsRenderDebug()	269
7.1.1.575 SceneLightsSetClusterSize()	269
7.1.1.576 SceneLightsSetCullingMode()	269
7.1.1.577 SceneLightsSetDepthSlices()	269
7.1.1.578 SceneLightsSetViewSize()	269
7.1.1.579 SceneMeshAutoDraw()	270
7.1.1.580 SceneMeshAutoDrawSliced()	270
7.1.1.581 SceneMeshesAdd()	270
7.1.1.582 SceneMeshesAddFromBuffer()	271
7.1.1.583 SceneMeshesAddFromFile()	271
7.1.1.584 SceneMeshesClear()	271
7.1.1.585 SceneMeshesCreate()	271
7.1.1.586 SceneMeshesDestroy()	272
7.1.1.587 SceneMeshesErase()	272
7.1.1.588 SceneMeshesGetCount()	272
7.1.1.589 SceneMeshesGetDevice()	272
7.1.1.590 SceneMeshesGetMeshByIndex()	272
7.1.1.591 SceneMeshesGetMeshByName()	272
7.1.1.592 SceneMeshesPayload()	273
7.1.1.593 SceneMeshesSlice()	273
7.1.1.594 SceneMeshGetBounds()	273
7.1.1.595 SceneMeshGetLatches()	273
7.1.1.596 SceneMeshGetMaterials()	273
7.1.1.597 SceneMeshGetModel()	273
7.1.1.598 SceneMeshGetName()	274
7.1.1.599 SceneMeshGetPayload()	274
7.1.1.600 SceneMeshGetScale()	274
7.1.1.601 SceneMeshGetSize()	274
7.1.1.602 SceneMeshGetSlice()	274
7.1.1.603 SceneMeshGetTags()	274
7.1.1.604 SceneMeshGetVertexElementsIndex()	275
7.1.1.605 SceneMeshGetVoxel()	275
7.1.1.606 SceneMeshLatchesAdd()	275
7.1.1.607 SceneMeshLatchesClear()	275
7.1.1.608 SceneMeshLatchesCopy()	275

7.1.1.609 SceneMeshLatchesCreate()	275
7.1.1.610 SceneMeshLatchesDestroy()	276
7.1.1.611 SceneMeshLatchesErase()	276
7.1.1.612 SceneMeshLatchesGetCount()	276
7.1.1.613 SceneMeshLatchesGetLatch()	276
7.1.1.614 SceneMeshLatchesGetLatchIndex()	276
7.1.1.615 SceneMeshLatchesGetWaypointCouple()	276
7.1.1.616 SceneMeshLatchesGetWaypointDistance()	277
7.1.1.617 SceneMeshLatchesInvalidateWaypoints()	277
7.1.1.618 SceneMeshLatchesLoadFromFile()	277
7.1.1.619 SceneMeshLatchesLoadFromFileInMemory()	277
7.1.1.620 SceneMeshLatchesSaveToFile()	277
7.1.1.621 SceneMeshLatchesSetLatch()	277
7.1.1.622 SceneMeshLatchesTakeAway()	278
7.1.1.623 SceneMeshMaterialAddRange()	278
7.1.1.624 SceneMeshMaterialClearRanges()	278
7.1.1.625 SceneMeshMaterialCommit()	278
7.1.1.626 SceneMeshMaterialCopy()	278
7.1.1.627 SceneMeshMaterialGetName()	278
7.1.1.628 SceneMeshMaterialGetRange()	279
7.1.1.629 SceneMeshMaterialGetRangeCount()	279
7.1.1.630 SceneMeshMaterialGetShading()	279
7.1.1.631 SceneMeshMaterialGetTexture()	279
7.1.1.632 SceneMeshMaterialReleaseTextures()	279
7.1.1.633 SceneMeshMaterialsAdd()	279
7.1.1.634 SceneMeshMaterialsClear()	280
7.1.1.635 SceneMeshMaterialsCommit()	280
7.1.1.636 SceneMeshMaterialsCopy()	280
7.1.1.637 SceneMeshMaterialsCreate()	280
7.1.1.638 SceneMeshMaterialsDestroy()	280
7.1.1.639 SceneMeshMaterialsErase()	280
7.1.1.640 SceneMeshMaterialSetRange()	281
7.1.1.641 SceneMeshMaterialSetShading()	281
7.1.1.642 SceneMeshMaterialSetTexture()	281
7.1.1.643 SceneMeshMaterialsGetCount()	281
7.1.1.644 SceneMeshMaterialsGetMaterial()	281
7.1.1.645 SceneMeshMaterialsGetName()	281
7.1.1.646 SceneMeshMaterialsSetName()	282
7.1.1.647 SceneMeshMaterialsTakeAway()	282
7.1.1.648 SceneMeshMaterialsTexturing()	282
7.1.1.649 SceneMeshSetSlice()	282
7.1.1.650 SceneMeshSetVertexElementsIndex()	282

7.1.1.651 ScenePrepare()	282
7.1.1.652 SceneRendering()	283
7.1.1.653 SceneResetCache()	283
7.1.1.654 SceneSetActiveVertexElements()	283
7.1.1.655 SceneSetAttributes()	283
7.1.1.656 SceneSetLights()	283
7.1.1.657 SceneSetMaterial()	283
7.1.1.658 SceneSetParallaxMappingParameters()	284
7.1.1.659 SceneSetProjection()	284
7.1.1.660 SceneSetShadowCastingAtlas()	284
7.1.1.661 SceneSetTexture()	284
7.1.1.662 SceneSetTextureCabinet()	284
7.1.1.663 SceneSetToneMappingCoefficients()	284
7.1.1.664 SceneSetVertexElements()	285
7.1.1.665 SceneSetVertexElementsFromTextModeller()	285
7.1.1.666 SceneSetView()	285
7.1.1.667 SceneSetWorld()	285
7.1.1.668 SelectionHighlightBegin()	285
7.1.1.669 SelectionHighlightClear()	285
7.1.1.670 SelectionHighlightCreate()	286
7.1.1.671 SelectionHighlightDestroy()	286
7.1.1.672 SelectionHighlightEnd()	286
7.1.1.673 SelectionHighlightFilter()	286
7.1.1.674 SelectionHighlightGetDepthStencil()	286
7.1.1.675 SelectionHighlightGetDevice()	286
7.1.1.676 SelectionHighlightGetGrayscale()	287
7.1.1.677 SelectionHighlightGetParameters()	287
7.1.1.678 SelectionHighlightGetSamples()	287
7.1.1.679 SelectionHighlightGetSize()	287
7.1.1.680 SelectionHighlightGetTexture()	287
7.1.1.681 SelectionHighlightGetTextureCoordinates()	287
7.1.1.682 SelectionHighlightRendering()	288
7.1.1.683 SelectionHighlightSetParameters()	288
7.1.1.684 SelectionHighlightSetSize()	288
7.1.1.685 ShadowCasterBegin()	288
7.1.1.686 ShadowCasterClear()	288
7.1.1.687 ShadowCasterEnd()	288
7.1.1.688 ShadowCasterFilter()	288
7.1.1.689 ShadowCasterGetAtlas()	289
7.1.1.690 ShadowCasterGetDevice()	289
7.1.1.691 ShadowCasterGetPosition()	289
7.1.1.692 ShadowCasterGetSize()	289

7.1.1.693 ShadowCasterGetTexture()	289
7.1.1.694 ShadowCasterGetViewProjection()	289
7.1.1.695 ShadowCasterRendering()	290
7.1.1.696 ShadowCasterSetViewProjection()	290
7.1.1.697 ShadowCastingAtlasAdd()	290
7.1.1.698 ShadowCastingAtlasBorderFill()	290
7.1.1.699 ShadowCastingAtlasClear()	290
7.1.1.700 ShadowCastingAtlasCreate()	290
7.1.1.701 ShadowCastingAtlasDestroy()	291
7.1.1.702 ShadowCastingAtlasErase()	291
7.1.1.703 ShadowCastingAtlasGetCaster()	291
7.1.1.704 ShadowCastingAtlasGetCount()	291
7.1.1.705 ShadowCastingAtlasGetDevice()	291
7.1.1.706 ShadowCastingAtlasGetPadding()	291
7.1.1.707 ShadowCastingAtlasGetParameters()	292
7.1.1.708 ShadowCastingAtlasGetSamples()	292
7.1.1.709 ShadowCastingAtlasGetSize()	292
7.1.1.710 ShadowCastingAtlasGetTechnique()	292
7.1.1.711 ShadowCastingAtlasGetTexture()	292
7.1.1.712 ShadowCastingAtlasSetParameters()	292
7.1.1.713 SineAccelerate()	293
7.1.1.714 SineCycle()	293
7.1.1.715 SineDecelerate()	293
7.1.1.716 SineTransform()	293
7.1.1.717 SineTwoCycle()	293
7.1.1.718 SmootherStep()	293
7.1.1.719 SmoothStep()	294
7.1.1.720 SpatialFogCreate()	294
7.1.1.721 SpatialFogDestroy()	294
7.1.1.722 SpatialFogExecute()	294
7.1.1.723 SpatialFogExecuteGlassy()	294
7.1.1.724 SpatialFogGetDepthDistance()	294
7.1.1.725 SpatialFogGetDevice()	295
7.1.1.726 SpatialFogGetFormula()	295
7.1.1.727 SpatialFogGetParameters()	295
7.1.1.728 SpatialFogGetProjection()	295
7.1.1.729 SpatialFogGetView()	295
7.1.1.730 SpatialFogSetDepthDistance()	295
7.1.1.731 SpatialFogSetFormula()	296
7.1.1.732 SpatialFogSetParameters()	296
7.1.1.733 SpatialFogSetProjection()	296
7.1.1.734 SpatialFogSetView()	296

7.1.1.735 SubtractColors()	296
7.1.1.736 SurfaceClear()	296
7.1.1.737 SurfaceClearWith()	297
7.1.1.738 SurfaceConvertFormat()	297
7.1.1.739 SurfaceCopyFrom()	297
7.1.1.740 SurfaceCopyRect()	297
7.1.1.741 SurfaceCreate()	297
7.1.1.742 SurfaceCreateFromFile()	298
7.1.1.743 SurfaceCreateFromFileInMemory()	298
7.1.1.744 SurfaceDestroy()	298
7.1.1.745 SurfaceEmpty()	298
7.1.1.746 SurfaceFlip()	298
7.1.1.747 SurfaceGetBits()	299
7.1.1.748 SurfaceGetByteSize()	299
7.1.1.749 SurfaceGetBytesPerPixel()	299
7.1.1.750 SurfaceGetFormat()	299
7.1.1.751 SurfaceGetHeight()	299
7.1.1.752 SurfaceGetPitch()	299
7.1.1.753 SurfaceGetPixel()	300
7.1.1.754 SurfaceGetPixelBilinear()	300
7.1.1.755 SurfaceGetPixelPtr()	300
7.1.1.756 SurfaceGetPremultipliedAlpha()	300
7.1.1.757 SurfaceGetScanline()	300
7.1.1.758 SurfaceGetWidth()	300
7.1.1.759 SurfaceHasAlphaChannel()	301
7.1.1.760 SurfaceInvert()	301
7.1.1.761 SurfaceMakeSignedDistanceField()	301
7.1.1.762 SurfaceMirror()	301
7.1.1.763 SurfacePremultiplyAlpha()	301
7.1.1.764 SurfaceResetAlpha()	302
7.1.1.765 SurfaceResize()	302
7.1.1.766 SurfaceSaveToFile()	302
7.1.1.767 SurfaceSaveToFileInMemory()	302
7.1.1.768 SurfaceSetPixel()	302
7.1.1.769 SurfaceSetPremultipliedAlpha()	303
7.1.1.770 SurfaceShrinkFrom()	303
7.1.1.771 SurfaceStretchRect()	303
7.1.1.772 SurfaceStretchRectBilinear()	303
7.1.1.773 SurfaceUnpremultiplyAlpha()	303
7.1.1.774 SwapChainBegin()	304
7.1.1.775 SwapChainCreate()	304
7.1.1.776 SwapChainDestroy()	304

7.1.1.777 SwapChainEnd()	304
7.1.1.778 SwapChainGetDepthStencil()	304
7.1.1.779 SwapChainGetDevice()	304
7.1.1.780 SwapChainGetFormat()	305
7.1.1.781 SwapChainGetHeight()	305
7.1.1.782 SwapChainGetMultisamples()	305
7.1.1.783 SwapChainGetVSync()	305
7.1.1.784 SwapChainGetWidth()	305
7.1.1.785 SwapChainGetWindowHandle()	305
7.1.1.786 SwapChainResize()	305
7.1.1.787 TextModellerClear()	306
7.1.1.788 TextModellerCopyToMeshBuffer()	306
7.1.1.789 TextModellerCreate()	306
7.1.1.790 TextModellerDestroy()	306
7.1.1.791 TextModellerDraw()	306
7.1.1.792 TextModellerDrawCurved()	307
7.1.1.793 TextModellerDrawCurvedToCanvas()	307
7.1.1.794 TextModellerDrawCurvedToCanvasBuffer()	307
7.1.1.795 TextModellerDrawDepthCurved()	308
7.1.1.796 TextModellerDrawToCanvas()	308
7.1.1.797 TextModellerDrawToCanvasBuffer()	308
7.1.1.798 TextModellerExtent()	308
7.1.1.799 TextModellerExtentByShape()	309
7.1.1.800 TextModellerGetDevice()	309
7.1.1.801 TextModellerGetFontProvider()	309
7.1.1.802 TextModellerGetTransform()	309
7.1.1.803 TextModellerPrepare()	309
7.1.1.804 TextModellerRects()	309
7.1.1.805 TextModellerRender()	310
7.1.1.806 TextModellerReset()	310
7.1.1.807 TextModellerSetFontParameters()	310
7.1.1.808 TextModellerSetFontProvider()	310
7.1.1.809 TextModellerSetTransform()	310
7.1.1.810 TextRendererCreate()	310
7.1.1.811 TextRendererDestroy()	311
7.1.1.812 TextRendererDraw()	311
7.1.1.813 TextRendererDrawAligned()	311
7.1.1.814 TextRendererDrawAlignedByPixels()	311
7.1.1.815 TextRendererDrawCentered()	312
7.1.1.816 TextRendererDrawCenteredByPixels()	312
7.1.1.817 TextRendererExtent()	312
7.1.1.818 TextRendererExtentByPixels()	312

7.1.1.819 TextRendererGetCanvas()	312
7.1.1.820 TextRendererGetFontParameters()	313
7.1.1.821 TextRendererRects()	313
7.1.1.822 TextRendererSetFontParameters()	313
7.1.1.823 TextureAttach()	313
7.1.1.824 TextureBegin()	313
7.1.1.825 TextureBind()	314
7.1.1.826 TextureCabinetBegin()	314
7.1.1.827 TextureCabinetClear()	314
7.1.1.828 TextureCabinetCreate()	314
7.1.1.829 TextureCabinetDestroy()	314
7.1.1.830 TextureCabinetEnd()	314
7.1.1.831 TextureCabinetFilter()	315
7.1.1.832 TextureCabinetGetAmbientOcclusionParameters()	315
7.1.1.833 TextureCabinetGetAttributes()	315
7.1.1.834 TextureCabinetGetDepthStencil()	315
7.1.1.835 TextureCabinetGetDevice()	315
7.1.1.836 TextureCabinetGetFidelity()	315
7.1.1.837 TextureCabinetGetFinalTexture()	316
7.1.1.838 TextureCabinetGetSamples()	316
7.1.1.839 TextureCabinetGetSize()	316
7.1.1.840 TextureCabinetGetTexture()	316
7.1.1.841 TextureCabinetGetToneMappingBloom()	316
7.1.1.842 TextureCabinetPresent()	316
7.1.1.843 TextureCabinetRendering()	317
7.1.1.844 TextureCabinetResolve()	317
7.1.1.845 TextureCabinetRetrieveGlassyMetrics()	317
7.1.1.846 TextureCabinetSetAmbientOcclusionParameters()	317
7.1.1.847 TextureCabinetSetAttributes()	317
7.1.1.848 TextureCabinetSetSize()	317
7.1.1.849 TextureCabinetSetToneMappingBloom()	318
7.1.1.850 TextureClear()	318
7.1.1.851 TextureClearWith()	318
7.1.1.852 TextureCopy()	318
7.1.1.853 TextureCopyFromSurface()	318
7.1.1.854 TextureCopyToSurface()	319
7.1.1.855 TextureCreate()	319
7.1.1.856 TextureCreateFromFile()	319
7.1.1.857 TextureCreateFromFileInMemory()	319
7.1.1.858 TextureCreateFromFileNormalsAndOcclusion()	320
7.1.1.859 TextureCreateFromFileParallax()	320
7.1.1.860 TextureCreateFromSurface()	320

7.1.1.861 TextureDestroy()	320
7.1.1.862 TextureDetach()	320
7.1.1.863 TextureEnd()	320
7.1.1.864 TextureGenerateMipMaps()	321
7.1.1.865 TextureGetDevice()	321
7.1.1.866 TextureGetParameters()	321
7.1.1.867 TextureGetPlatformHandle()	321
7.1.1.868 TextureLoadFromFile()	321
7.1.1.869 TextureResetCache()	321
7.1.1.870 TextureRetrieve()	322
7.1.1.871 TextureSaveToFile()	322
7.1.1.872 TextureUnbind()	322
7.1.1.873 TextureUpdate()	322
7.1.1.874 TimerCreate()	322
7.1.1.875 TimerDestroy()	323
7.1.1.876 TimerExtractTokens()	323
7.1.1.877 TimerGetFrameRate()	323
7.1.1.878 TimerGetLatency()	323
7.1.1.879 TimerGetSkippedTimeSlices()	323
7.1.1.880 TimerGetSpeed()	323
7.1.1.881 TimerGetTimeSlice()	324
7.1.1.882 TimerNextSlice()	324
7.1.1.883 TimerReset()	324
7.1.1.884 TimerSetSpeed()	324
7.1.1.885 TimerTrimSkippedTimeSlices()	324
7.1.1.886 TimerUpdate()	324
7.1.1.887 TimerUpdateNextSlice()	325
7.1.1.888 UnpremultiplyAlpha()	325
7.1.1.889 VertexElementsEstimatePitch()	325
7.1.1.890 VolumeCalculateNearFarPlanes()	325
7.1.1.891 VolumeCalculateVisibleFrame()	325
7.1.1.892 WidgetAcceptKey()	325
7.1.1.893 WidgetAcceptMouse()	326
7.1.1.894 WidgetAccomodate()	326
7.1.1.895 WidgetBringToFront()	326
7.1.1.896 WidgetCreate()	326
7.1.1.897 WidgetDestroy()	326
7.1.1.898 WidgetFindAt()	326
7.1.1.899 WidgetGetChildByIndex()	327
7.1.1.900 WidgetGetChildCount()	327
7.1.1.901 WidgetGetChildIndex()	327
7.1.1.902 WidgetGetExternalEvent()	327

7.1.1.903 WidgetGetManager()	327
7.1.1.904 WidgetGetParent()	327
7.1.1.905 WidgetGetPayload()	328
7.1.1.906 WidgetGetProperty()	328
7.1.1.907 WidgetGetPropertyAsString()	328
7.1.1.908 WidgetGetPropertyCount()	328
7.1.1.909 WidgetInvalidate()	328
7.1.1.910 WidgetInvokeEvent()	328
7.1.1.911 WidgetLocalToScreen()	329
7.1.1.912 WidgetManagerBatchCount()	329
7.1.1.913 WidgetManagerCreate()	329
7.1.1.914 WidgetManagerGetApplication()	329
7.1.1.915 WidgetManagerGetAttributes()	329
7.1.1.916 WidgetManagerGetFormat()	329
7.1.1.917 WidgetManagerGetMultisamples()	330
7.1.1.918 WidgetManagerGetTextRenderer()	330
7.1.1.919 WidgetManagerGetTexture()	330
7.1.1.920 WidgetManagerGetWidgetByName()	330
7.1.1.921 WidgetManagerGetWidgetByPayload()	330
7.1.1.922 WidgetManagerPresent()	330
7.1.1.923 WidgetPropertyIdentify()	331
7.1.1.924 WidgetScreenToLocal()	331
7.1.1.925 WidgetSendToBack()	331
7.1.1.926 WidgetSetExternalEvent()	331
7.1.1.927 WidgetSetParent()	331
7.1.1.928 WidgetSetProperty()	332
7.1.1.929 WidgetUpdate()	332
7.2 Afterwarp.h	332
7.3 Afterwarp.Structs.h File Reference	363
7.4 Afterwarp.Structs.h	369
7.5 Afterwarp.Types.h File Reference	382
7.5.1 Typedef Documentation	394
7.5.1.1 AlphaFormatRequest	394
7.5.1.2 AppCursor	395
7.5.1.3 ApplicationEvent	395
7.5.1.4 ApplicationWindowState	395
7.5.1.5 BlendFactor	395
7.5.1.6 BlendingEffect	395
7.5.1.7 BlendOp	395
7.5.1.8 BufferAccessType	395
7.5.1.9 BufferDataType	395
7.5.1.10 CameraCommand	396

7.5.1.11 CanvasContextState	396
7.5.1.12 ClustersCullingMode	396
7.5.1.13 Color	396
7.5.1.14 ColorDitheringFormat	396
7.5.1.15 ColorPair	396
7.5.1.16 ColorRect	396
7.5.1.17 ComparisonFunc	397
7.5.1.18 ComputeTextureAccess	397
7.5.1.19 DepthFogDistance	397
7.5.1.20 DevicePlatform	397
7.5.1.21 DeviceTechnology	397
7.5.1.22 ElementFormat	397
7.5.1.23 FileChooserDialog	397
7.5.1.24 FloatColor	398
7.5.1.25 FloatColorRGB	398
7.5.1.26 FogFormula	398
7.5.1.27 FontAttributes	398
7.5.1.28 FontBorder	398
7.5.1.29 FontSlant	398
7.5.1.30 FontStretch	398
7.5.1.31 FontWeight	399
7.5.1.32 Key	399
7.5.1.33 KeyEvent	399
7.5.1.34 LineCaps	399
7.5.1.35 Margins	399
7.5.1.36 Matrix	399
7.5.1.37 Matrix3x2	399
7.5.1.38 MeshAlign	399
7.5.1.39 ModelTransform	400
7.5.1.40 MouseButton	400
7.5.1.41 MouseEvent	400
7.5.1.42 ObjectModelViewCompare	400
7.5.1.43 PathJoint	400
7.5.1.44 PixelFormat	400
7.5.1.45 Point	400
7.5.1.46 PointF	400
7.5.1.47 PointShape	401
7.5.1.48 PrimitiveTopology	401
7.5.1.49 Quad	401
7.5.1.50 Quaternion	401
7.5.1.51 Rect	401
7.5.1.52 RectF	401

7.5.1.53 SceneSamplerType	401
7.5.1.54 SceneTextureType	401
7.5.1.55 SelectionHighlightTextureType	402
7.5.1.56 ShaderElement	402
7.5.1.57 ShaderType	402
7.5.1.58 StencilOp	402
7.5.1.59 SuperSampleSDF	402
7.5.1.60 TechniqueLighting	402
7.5.1.61 TechniqueShadows	402
7.5.1.62 TextAlignment	402
7.5.1.63 TextureAddress	403
7.5.1.64 TextureCabinetFilterType	403
7.5.1.65 TextureCabinetPass	403
7.5.1.66 TextureCabinetType	403
7.5.1.67 TextureFidelity	403
7.5.1.68 TextureFilter	403
7.5.1.69 TextureType	403
7.5.1.70 TriangleFace	403
7.5.1.71 Vector	404
7.5.1.72 Vector4	404
7.5.1.73 WidgetAlignment	404
7.5.1.74 WidgetManagerTextureType	404
7.5.1.75 WidgetPropertyBehavior	404
7.5.1.76 WidgetPropertyType	404
7.5.2 Enumeration Type Documentation	404
7.5.2.1 anonymous enum	404
7.5.2.2 anonymous enum	405
7.5.2.3 anonymous enum	405
7.5.2.4 anonymous enum	405
7.5.2.5 anonymous enum	406
7.5.2.6 anonymous enum	406
7.5.2.7 anonymous enum	407
7.5.2.8 anonymous enum	408
7.5.2.9 anonymous enum	408
7.5.2.10 anonymous enum	409
7.5.2.11 anonymous enum	409
7.5.2.12 anonymous enum	409
7.5.2.13 anonymous enum	410
7.5.2.14 anonymous enum	410
7.5.2.15 anonymous enum	411
7.5.2.16 anonymous enum	411
7.5.2.17 anonymous enum	411

7.5.2.18 anonymous enum	412
7.5.2.19 anonymous enum	412
7.5.2.20 anonymous enum	413
7.5.2.21 anonymous enum	413
7.5.2.22 anonymous enum	413
7.5.2.23 anonymous enum	414
7.5.2.24 anonymous enum	414
7.5.2.25 anonymous enum	414
7.5.2.26 anonymous enum	415
7.5.2.27 anonymous enum	415
7.5.2.28 anonymous enum	416
7.5.2.29 anonymous enum	416
7.5.2.30 anonymous enum	416
7.5.2.31 anonymous enum	417
7.5.2.32 anonymous enum	417
7.5.2.33 anonymous enum	417
7.5.2.34 anonymous enum	417
7.5.2.35 anonymous enum	418
7.5.2.36 anonymous enum	418
7.5.2.37 anonymous enum	419
7.5.2.38 anonymous enum	419
7.5.2.39 anonymous enum	419
7.5.2.40 anonymous enum	419
7.5.2.41 anonymous enum	420
7.5.2.42 anonymous enum	420
7.5.2.43 anonymous enum	420
7.5.2.44 anonymous enum	420
7.5.2.45 anonymous enum	421
7.5.2.46 anonymous enum	421
7.5.2.47 anonymous enum	421
7.5.2.48 anonymous enum	422
7.5.2.49 anonymous enum	422
7.5.2.50 anonymous enum	422
7.5.2.51 anonymous enum	423
7.5.2.52 anonymous enum	423
7.5.2.53 anonymous enum	423
7.5.2.54 anonymous enum	424
7.5.2.55 anonymous enum	424
7.5.2.56 anonymous enum	425
7.5.2.57 anonymous enum	425
7.5.2.58 anonymous enum	425
7.5.2.59 anonymous enum	425

7.5.2.60 anonymous enum	426
7.5.2.61 anonymous enum	426
7.5.2.62 anonymous enum	427
7.5.2.63 anonymous enum	427
7.5.2.64 anonymous enum	427
7.5.2.65 anonymous enum	428
7.5.2.66 anonymous enum	428
7.5.2.67 anonymous enum	429
7.5.2.68 anonymous enum	429
7.5.2.69 anonymous enum	429
7.5.2.70 anonymous enum	430
7.5.2.71 anonymous enum	430
7.5.2.72 anonymous enum	430
7.5.2.73 anonymous enum	431
7.5.2.74 anonymous enum	431
7.5.2.75 anonymous enum	432
7.5.2.76 anonymous enum	432
7.5.2.77 anonymous enum	433
7.5.2.78 anonymous enum	433
7.5.2.79 anonymous enum	433
7.5.2.80 Key	435
7.5.2.81 WidgetManagerAttribute	437
7.6 Afterwarp.Types.h	437
7.7 Afterwarp.Vectors.h File Reference	456
7.7.1 Macro Definition Documentation	467
7.7.1.1 AFTERWARP_VECTORS_H	467
7.7.1.2 AFTERWARP_VECTORS_INTERFACE	468
7.7.1.3 MAX	468
7.7.1.4 MIN	468
7.7.2 Function Documentation	468
7.7.2.1 colorPair()	468
7.7.2.2 floatColor()	468
7.7.2.3 floatColorAdd()	468
7.7.2.4 floatColorAddRGB()	468
7.7.2.5 floatColorAverage()	469
7.7.2.6 floatColorAverageRGB()	469
7.7.2.7 floatColorDivide()	469
7.7.2.8 floatColorDivideRGB()	469
7.7.2.9 floatColorEmpty()	469
7.7.2.10 floatColorEmptyRGB()	469
7.7.2.11 floatColorEquals()	470
7.7.2.12 floatColorEqualsRGB()	470

7.7.2.13 floatColorFromValue()	470
7.7.2.14 floatColorFromValueRGB()	470
7.7.2.15 floatColorGray()	470
7.7.2.16 floatColorGrayRGB()	470
7.7.2.17 floatColorInverse()	471
7.7.2.18 floatColorInverseRGB()	471
7.7.2.19 floatColorLerp()	471
7.7.2.20 floatColorLerpRGB()	471
7.7.2.21 floatColorMultiply()	471
7.7.2.22 floatColorMultiplyRGB()	471
7.7.2.23 floatColorNegate()	472
7.7.2.24 floatColorNegateRGB()	472
7.7.2.25 floatColorPremultiply()	472
7.7.2.26 floatColorRGB()	472
7.7.2.27 floatColorSaturate()	472
7.7.2.28 floatColorSaturateRGB()	472
7.7.2.29 floatColorSubtract()	472
7.7.2.30 floatColorSubtractRGB()	473
7.7.2.31 floatColorToRGB()	473
7.7.2.32 floatColorUnpremultiply()	473
7.7.2.33 floatToColor()	473
7.7.2.34 floatToColorRGB()	473
7.7.2.35 makeBounds()	473
7.7.2.36 makeBoundsF()	474
7.7.2.37 makeColorPair()	474
7.7.2.38 makeFloatColor()	474
7.7.2.39 makeFloatColorRGB()	474
7.7.2.40 makePoint()	474
7.7.2.41 makePointF()	474
7.7.2.42 makeQuad()	475
7.7.2.43 makeQuadRotated()	475
7.7.2.44 makeQuadRotatedAt()	475
7.7.2.45 makeQuadRotatedTL()	475
7.7.2.46 makeQuadScaled()	476
7.7.2.47 makeQuadSkewedHoriz()	476
7.7.2.48 makeQuadSkewedVert()	476
7.7.2.49 makeQuadWith()	476
7.7.2.50 makeRect()	476
7.7.2.51 makeRectF()	477
7.7.2.52 makeVector()	477
7.7.2.53 makeVector3D()	477
7.7.2.54 matrixAdjoint()	477

7.7.2.55 matrixDeterminant()	477
7.7.2.56 matrixDeterminant3x2()	477
7.7.2.57 matrixEmpty()	478
7.7.2.58 matrixEmpty3x2()	478
7.7.2.59 matrixEquals()	478
7.7.2.60 matrixEquals3x2()	478
7.7.2.61 matrixEyePosition()	478
7.7.2.62 matrixGet3x2()	478
7.7.2.63 matrixHeadingPitchBank()	479
7.7.2.64 matrixInverse()	479
7.7.2.65 matrixInverse3x2()	479
7.7.2.66 matrixInverseTranspose3x2()	479
7.7.2.67 matrixLookAt()	479
7.7.2.68 matrixMultiply()	479
7.7.2.69 matrixMultiply3x2()	480
7.7.2.70 matrixOrthogonalBDS()	480
7.7.2.71 matrixOrthogonalVOL()	480
7.7.2.72 matrixPerspectiveBDS()	480
7.7.2.73 matrixPerspectiveFOVX()	480
7.7.2.74 matrixPerspectiveFOVY()	481
7.7.2.75 matrixPerspectiveVOL()	481
7.7.2.76 matrixQuaternion()	481
7.7.2.77 matrixReflect()	481
7.7.2.78 matrixRescale()	481
7.7.2.79 matrixRescale3x2()	482
7.7.2.80 matrixRotate()	482
7.7.2.81 matrixRotate3x2()	482
7.7.2.82 matrixRotateBy3x2()	482
7.7.2.83 matrixRotateX()	482
7.7.2.84 matrixRotateY()	482
7.7.2.85 matrixRotateZ()	483
7.7.2.86 matrixScale()	483
7.7.2.87 matrixScale3x2()	483
7.7.2.88 matrixScaleBy3x2()	483
7.7.2.89 matrixSub3x3()	483
7.7.2.90 matrixTranslate()	483
7.7.2.91 matrixTranslate3x2()	483
7.7.2.92 matrixTranspose()	484
7.7.2.93 matrixWorldPosition()	484
7.7.2.94 matrixYawPitchRoll()	484
7.7.2.95 pointAdd()	484
7.7.2.96 pointAddF()	484

7.7.2.97 pointAngle()	484
7.7.2.98 pointAngleF()	485
7.7.2.99 pointCross()	485
7.7.2.100 pointCrossF()	485
7.7.2.101 pointDistance()	485
7.7.2.102 pointDistanceF()	485
7.7.2.103 pointDivide()	485
7.7.2.104 pointDivideF()	486
7.7.2.105 pointDot()	486
7.7.2.106 pointDotF()	486
7.7.2.107 pointEmpty()	486
7.7.2.108 pointEmptyF()	486
7.7.2.109 pointEquals()	486
7.7.2.110 pointEqualsF()	487
7.7.2.111 pointF()	487
7.7.2.112 pointInRect()	487
7.7.2.113 pointInRectF()	487
7.7.2.114 pointLength()	487
7.7.2.115 pointLengthF()	487
7.7.2.116 pointLerp()	488
7.7.2.117 pointLerpF()	488
7.7.2.118 pointMultiply()	488
7.7.2.119 pointMultiplyF()	488
7.7.2.120 pointNegate()	488
7.7.2.121 pointNegateF()	488
7.7.2.122 pointNormalizeF()	489
7.7.2.123 pointRescale()	489
7.7.2.124 pointRescaleF()	489
7.7.2.125 pointSubtract()	489
7.7.2.126 pointSubtractF()	489
7.7.2.127 pointTransformF()	489
7.7.2.128 pointValue()	490
7.7.2.129 pointValueF()	490
7.7.2.130 quadFlip()	490
7.7.2.131 quadFromRect()	490
7.7.2.132 quadFromRectF()	490
7.7.2.133 quadMirror()	490
7.7.2.134 quadOffset()	491
7.7.2.135 quadScale()	491
7.7.2.136 quadTransform()	491
7.7.2.137 quaternionAngle()	491
7.7.2.138 quaternionAxis()	491

7.7.2.139 quaternionConjugate()	491
7.7.2.140 quaternionDot()	492
7.7.2.141 quaternionExponentiate()	492
7.7.2.142 quaternionLength()	492
7.7.2.143 quaternionMatrix()	492
7.7.2.144 quaternionMultiply()	492
7.7.2.145 quaternionNormalize()	492
7.7.2.146 quaternionRotate()	493
7.7.2.147 quaternionRotateInertialToObject()	493
7.7.2.148 quaternionRotateObjectToInertial()	493
7.7.2.149 quaternionRotateX()	493
7.7.2.150 quaternionRotateY()	493
7.7.2.151 quaternionRotateZ()	493
7.7.2.152 quaternionSlerp()	494
7.7.2.153 rectEmpty()	494
7.7.2.154 rectEmptyF()	494
7.7.2.155 rectEquals()	494
7.7.2.156 rectEqualsF()	494
7.7.2.157 rectF()	494
7.7.2.158 rectGetBottom()	495
7.7.2.159 rectGetBottomF()	495
7.7.2.160 rectGetBottomLeft()	495
7.7.2.161 rectGetBottomLeftF()	495
7.7.2.162 rectGetBottomRight()	495
7.7.2.163 rectGetBottomRightF()	495
7.7.2.164 rectGetRight()	495
7.7.2.165 rectGetRightF()	496
7.7.2.166 rectGetSize()	496
7.7.2.167 rectGetSizeF()	496
7.7.2.168 rectGetTopLeft()	496
7.7.2.169 rectGetTopLeftF()	496
7.7.2.170 rectGetTopRight()	496
7.7.2.171 rectGetTopRightF()	496
7.7.2.172 rectInflate()	497
7.7.2.173 rectInflateF()	497
7.7.2.174 rectInRect()	497
7.7.2.175 rectInRectF()	497
7.7.2.176 rectIntersect()	497
7.7.2.177 rectIntersectF()	497
7.7.2.178 rectJoin()	498
7.7.2.179 rectJoinF()	498
7.7.2.180 rectOffset()	498

7.7.2.181 rectOffsetF()	498
7.7.2.182 rectOverlap()	498
7.7.2.183 rectOverlapF()	498
7.7.2.184 rectSetBottom()	499
7.7.2.185 rectSetBottomF()	499
7.7.2.186 rectSetBottomRight()	499
7.7.2.187 rectSetBottomRightF()	499
7.7.2.188 rectSetRight()	499
7.7.2.189 rectSetRightF()	499
7.7.2.190 rectSetSize()	500
7.7.2.191 rectSetSizeF()	500
7.7.2.192 rectSetTopLeft()	500
7.7.2.193 rectSetTopLeftF()	500
7.7.2.194 rectToIntRectF()	500
7.7.2.195 vectorAdd()	500
7.7.2.196 vectorAdd3D()	501
7.7.2.197 vectorAngle()	501
7.7.2.198 vectorCross()	501
7.7.2.199 vectorDistance()	501
7.7.2.200 vectorDistance3D()	501
7.7.2.201 vectorDivide()	501
7.7.2.202 vectorDivide3D()	502
7.7.2.203 vectorDot()	502
7.7.2.204 vectorDot3D()	502
7.7.2.205 vectorEmpty()	502
7.7.2.206 vectorEmpty3D()	502
7.7.2.207 vectorEquals()	502
7.7.2.208 vectorEquals3D()	503
7.7.2.209 vectorFromValue3D()	503
7.7.2.210 vectorLength()	503
7.7.2.211 vectorLength3D()	503
7.7.2.212 vectorLerp()	503
7.7.2.213 vectorLerp3D()	503
7.7.2.214 vectorMultiply()	504
7.7.2.215 vectorMultiply3D()	504
7.7.2.216 vectorNegate()	504
7.7.2.217 vectorNegate3D()	504
7.7.2.218 vectorNormalize()	504
7.7.2.219 vectorNormalize3D()	504
7.7.2.220 vectorParallel()	505
7.7.2.221 vectorPerpendicular()	505
7.7.2.222 vectorProject()	505

7.7.2.223 vectorProjectTarget()	505
7.7.2.224 vectorReflect()	505
7.7.2.225 vectorRescale()	505
7.7.2.226 vectorRescale3D()	506
7.7.2.227 vectorSubtract()	506
7.7.2.228 vectorSubtract3D()	506
7.7.2.229 vectorTransform()	506
7.7.2.230 vectorTransform3D()	506
7.7.2.231 vectorValue()	506
7.7.3 Variable Documentation	507
7.7.3.1 colorBlack	507
7.7.3.2 colorPairBlack	507
7.7.3.3 colorPairTraslucentBlack	507
7.7.3.4 colorPairTraslucentWhite	507
7.7.3.5 colorPairWhite	507
7.7.3.6 colorRectBlack	507
7.7.3.7 colorRectTraslucentBlack	507
7.7.3.8 colorRectTraslucentWhite	508
7.7.3.9 colorRectWhite	508
7.7.3.10 colorTraslucentBlack	508
7.7.3.11 colorTraslucentWhite	508
7.7.3.12 colorWhite	508
7.7.3.13 floatColorBlack	508
7.7.3.14 floatColorBlackRGB	508
7.7.3.15 floatColorTraslucentBlack	508
7.7.3.16 floatColorTraslucentWhite	509
7.7.3.17 floatColorWhite	509
7.7.3.18 floatColorWhiteRGB	509
7.7.3.19 fontEffectDefault	509
7.7.3.20 matrixIdentity	509
7.7.3.21 matrixIdentity3x2	509
7.7.3.22 matrixZero	509
7.7.3.23 matrixZero3x2	509
7.7.3.24 pointAxisX	510
7.7.3.25 pointAxisXF	510
7.7.3.26 pointAxisY	510
7.7.3.27 pointAxisYF	510
7.7.3.28 pointUnity	510
7.7.3.29 pointUnityF	510
7.7.3.30 pointZero	510
7.7.3.31 pointZeroF	510
7.7.3.32 quadUnity	511

7.7.3.33 quadZero	511
7.7.3.34 quaternionIdentity	511
7.7.3.35 rectZero	511
7.7.3.36 rectZeroF	511
7.7.3.37 vectorAxisW3D	511
7.7.3.38 vectorAxisX	511
7.7.3.39 vectorAxisX3D	511
7.7.3.40 vectorAxisY	512
7.7.3.41 vectorAxisY3D	512
7.7.3.42 vectorAxisZ	512
7.7.3.43 vectorAxisZ3D	512
7.7.3.44 vectorEpsilon	512
7.7.3.45 vectorUnity	512
7.7.3.46 vectorUnity3D	512
7.7.3.47 vectorZero	513
7.7.3.48 vectorZero3D	513
7.8 Afterwarp.Vectors.h	513
Index	547

Chapter 1

Afterwarp Framework

Afterwarp Framework ® is a cross-platform framework for development of scientific, industrial and interactive real-time applications. The framework includes both low-level API with full hardware abstraction, allowing development of custom graphics engine using programmable pipeline and shaders, including compute shaders for GPGPU work, and a high-level framework that can be used to quickly develop 3D applications without knowledge of shaders.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

pxt	11
-------------------------------	----

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

AmbientOcclusionParameters	
Parameters that define how ambient occlusion is performed	31
AmbientOcclusionParameters	
Parameters that define how ambient occlusion is performed	31
ApplicationConfiguration	
Structure that contains all the parameters necessary to create a new application and its window	33
ApplicationConfiguration	
Structure that contains all the parameters necessary to create a new application and its window	33
ApplicationEvents	36
ApplicationEvents	36
Blend	
Blending parameters that are specific to a certain component, or a portion of color	38
Blend	
Blending parameters that are specific to a certain component, or a portion of color	38
CameraConstraints	40
CameraConstraints	40
CanvasSamplerState	
Sampler parameters that can be easily configured by the canvas	41
CanvasSamplerState	
Sampler parameters that can be easily configured by the canvas	41
ColorPair	42
ColorRect	44
ComputeBindTextureFormat	
Compute texture parameters and characteristics	45
ComputeBindTextureFormat	
Compute texture parameters and characteristics	45
FloatColor	46
FloatColorRGB	47
FogParameters	
Structure containing spatial fog characteristics	48
FogParameters	
Structure containing spatial fog characteristics	48
FontEffect	
Attributes and characteristics that define how font glyphs are rasterized	51
FontEffect	
Attributes and characteristics that define how font glyphs are rasterized	51

FontParameters	Parameters that define the appearance and important characteristics of the font	53
FontParameters	Parameters that define the appearance and important characteristics of the font	53
ImageRegion	An image region defined by its texture number and bounding rectangle on that texture	56
ImageRegion	An image region defined by its texture number and bounding rectangle on that texture	56
Margins	Margins defined by top, left, right and bottom edge paddings	57
Matrix	4x4 matrix representing transformation information in context of 3D graphics	59
Matrix3x2	3x2 matrix representing rotation and translation in context of 2D graphics	59
MeshAligns	Alignment for all three axes that determine the placement of mesh around its model	60
MeshAligns	Alignment for all three axes that determine the placement of mesh around its model	60
MeshBufferEntry	Calculated values for a single mesh vertex that can be modified by callback	62
MeshBufferEntry	Calculated values for a single mesh vertex that can be modified by callback	62
MeshMetaTagPortion	A portion of the mesh corresponding to a particular tag	63
MeshMetaTagPortion	A portion of the mesh corresponding to a particular tag	63
ObjectMaterial	Material that describes a particular object in 3D world	65
ObjectMaterial	Material that describes a particular object in 3D world	65
OceanMaterial	Material that describes ocean water surface in a 3D simulation	68
OceanMaterial	Material that describes ocean water surface in a 3D simulation	68
OceanWavesParameters	Parameters that define the appearance of waves simulation	70
OceanWavesParameters	Parameters that define the appearance of waves simulation	70
ParallaxMappingParameters	Parameters that define how parallax mapping is performed	72
ParallaxMappingParameters	Parameters that define how parallax mapping is performed	72
PathElement	A single element in path declaration	73
PathElement	A single element in path declaration	73
Point	2D integer point (or vector)	75
PointF	2D floating-point vector	76
ProgramElement	Structure that describes a single physical element of shader program	77
ProgramElement	Structure that describes a single physical element of shader program	77
ProgramVariable	Structure that describes a single (uniform) variable in a shader program	78
ProgramVariable	Structure that describes a single (uniform) variable in a shader program	78

Quad	Floating-point quadrilateral defined by four vertices in clockwise order starting from top/left . . .	80
Quaternion	3D quaternion	81
Rect	Rectangle defined by integer top and left margins, width and height	82
RectF	Rectangle defined by floating-point top and left margins, width and height	83
RenderingState	Parameters that define <code>depth</code> , <code>stencil</code> , <code>rasterizer</code> and <code>blending</code> operations	85
RenderingState	Parameters that define <code>depth</code> , <code>stencil</code> , <code>rasterizer</code> and <code>blending</code> operations	85
SamplerState	Sampler parameters that are associated with a particular texture unit	88
SamplerState	Sampler parameters that are associated with a particular texture unit	88
SceneLight	Light source parameters in 3D scene	90
SceneLight	Light source parameters in 3D scene	90
SceneMeshLatch	A latch in 3D scene mesh, which describes a bone joint connection or a movement waypoint .	93
SceneMeshLatch	A latch in 3D scene mesh, which describes a bone joint connection or a movement waypoint .	93
SceneMeshMaterialRange	A range of indices in the mesh that a material applies to	95
SceneMeshMaterialRange	A range of indices in the mesh that a material applies to	95
SceneMeshMaterialShading	Shading parameters for scene mesh material	96
SceneMeshMaterialShading	Shading parameters for scene mesh material	96
SelectionHighlightParameters	Parameters that define how selection highlight technique is performed	98
SelectionHighlightParameters	Parameters that define how selection highlight technique is performed	98
ShadowParameters	Parameters that define how shadows are rendered	99
ShadowParameters	Parameters that define how shadows are rendered	99
SignedDistanceField	Signed Distance Field (SDF) parameters	101
SignedDistanceField	Signed Distance Field (SDF) parameters	101
StencilState	Stencil state that is independent for each front and back facing	103
StencilState	Stencil state that is independent for each front and back facing	103
TextEntryRect	Individual text entry that will appear as rendered	104
TextEntryRect	Individual text entry that will appear as rendered	104
TextRenderModifiers	Modifier attributes that can be applied to the rendered text	105
TextRenderModifiers	Modifier attributes that can be applied to the rendered text	105
TextureParameters	Texture parameters and characteristics	106

TextureParameters	
Texture parameters and characteristics	106
ToneMappingBloom	
Parameters that define behavior of tone-mapping and bloom effects	108
ToneMappingBloom	
Parameters that define behavior of tone-mapping and bloom effects	108
Vector	
3D floating-point vector	111
Vector4	
4-component (XYZ + W) floating-point vector	112
VertexElement	
Structure that describes the layout of a single buffer element	113
VertexElement	
Structure that describes the layout of a single buffer element	113
WidgetProperty	
Information about a widget property	114
WidgetProperty	
Information about a widget property	114
WidgetPropertyData	
A flexible structure for storing property data	116
WidgetPropertyData	
A flexible structure for storing property data	116

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Afterwarp.h	121
Afterwarp.Structs.h	363
Afterwarp.Types.h	382
Afterwarp.Vectors.h	456

Chapter 5

Namespace Documentation

5.1 pxt Namespace Reference

Classes

- struct [AmbientOcclusionParameters](#)
Parameters that define how ambient occlusion is performed.
- struct [ApplicationConfiguration](#)
Structure that contains all the parameters necessary to create a new application and its window.
- struct [ApplicationEvents](#)
- struct [Blend](#)
Blending parameters that are specific to a certain component, or a portion of color.
- struct [CameraConstraints](#)
- struct [CanvasSamplerState](#)
Sampler parameters that can be easily configured by the canvas.
- struct [ComputeBindTextureFormat](#)
Compute texture parameters and characteristics.
- struct [FogParameters](#)
Structure containing spatial fog characteristics.
- struct [FontEffect](#)
Attributes and characteristics that define how font glyphs are rasterized.
- struct [FontParameters](#)
Parameters that define the appearance and important characteristics of the font.
- struct [ImageRegion](#)
An image region defined by its texture number and bounding rectangle on that texture.
- struct [MeshAligns](#)
Alignment for all three axes that determine the placement of mesh around its model.
- struct [MeshBufferEntry](#)
Calculated values for a single mesh vertex that can be modified by callback.
- struct [MeshMetaTagPortion](#)
A portion of the mesh corresponding to a particular tag.
- struct [ObjectMaterial](#)
Material that describes a particular object in 3D world.
- struct [OceanMaterial](#)
Material that describes ocean water surface in a 3D simulation.
- struct [OceanWavesParameters](#)

- Parameters that define the appearance of waves simulation.*
- struct [ParallaxMappingParameters](#)
 - Parameters that define how parallax mapping is performed.*
- union [PathElement](#)
 - A single element in path declaration.*
- struct [ProgramElement](#)
 - Structure that describes a single physical element of shader program.*
- struct [ProgramVariable](#)
 - Structure that describes a single (uniform) variable in a shader program.*
- struct [RenderingState](#)
 - Parameters that define `depth`, `stencil`, `rasterizer` and `blending` operations.*
- struct [SamplerState](#)
 - Sampler parameters that are associated with a particular texture unit.*
- struct [SceneLight](#)
 - Light source parameters in 3D scene.*
- struct [SceneMeshLatch](#)
 - A latch in 3D scene mesh, which describes a bone joint connection or a movement waypoint.*
- struct [SceneMeshMaterialRange](#)
 - A range of indices in the mesh that a material applies to.*
- struct [SceneMeshMaterialShading](#)
 - Shading parameters for scene mesh material.*
- struct [SelectionHighlightParameters](#)
 - Parameters that define how selection highlight technique is performed.*
- struct [ShadowParameters](#)
 - Parameters that define how shadows are rendered.*
- struct [SignedDistanceField](#)
 - Signed Distance Field (SDF) parameters.*
- struct [StencilState](#)
 - Stencil state that is independent for each front and back facing.*
- struct [TextEntryRect](#)
 - Individual text entry that will appear as rendered.*
- struct [TextRenderModifiers](#)
 - Modifier attributes that can be applied to the rendered text.*
- struct [TextureParameters](#)
 - Texture parameters and characteristics.*
- struct [ToneMappingBloom](#)
 - Parameters that define behavior of tone-mapping and bloom effects.*
- struct [VertexElement](#)
 - Structure that describes the layout of a single buffer element.*
- struct [WidgetProperty](#)
 - Information about a widget property.*
- union [WidgetPropertyData](#)
 - A flexible structure for storing property data.*

Typedefs

- typedef void * [UntypedHandle](#)
Generic untyped handle.
- typedef uint8_t [LibraryBool](#)
Boolean type returned by the library.
- typedef void * [ObjectPayload](#)
Custom object's payload.
- typedef uint32_t [DeviceAttributes](#)
One or more device attribute flags combined (ORed) together.
- typedef uint32_t [DeviceBehavior](#)
Device behavior flags that affect the rendering code path.
- typedef uint32_t [DeviceLegacyBits](#)
Legacy feature bits that define older hardware features.
- typedef struct [pxt::RenderingState](#) [RenderingState](#)
Parameters that define depth, stencil, rasterizer and blending operations.
- typedef struct [Buffer_t](#) [Buffer_t](#)
Buffer that is typically stored on graphics hardware.
- typedef struct [pxt::VertexElement](#) [VertexElement](#)
Structure that describes the layout of a single buffer element.
- typedef struct [pxt::ProgramElement](#) [ProgramElement](#)
Structure that describes a single physical element of shader program.
- typedef struct [pxt::ProgramVariable](#) [ProgramVariable](#)
Structure that describes a single (uniform) variable in a shader program.
- typedef struct [Program_t](#) [Program_t](#)
Shader program that is typically executed on graphics hardware.
- typedef struct [pxt::ComputeBindTextureFormat](#) [ComputeBindTextureFormat](#)
Compute texture parameters and characteristics.
- typedef struct [ComputeProgram_t](#) [ComputeProgram_t](#)
Compute shader program for general processing on GPU (GPGPU).
- typedef struct [pxt::SamplerState](#) [SamplerState](#)
Sampler parameters that are associated with a particular texture unit.
- typedef struct [Sampler_t](#) [Sampler_t](#)
Sampler object that defines how texture is read by the shaders.
- typedef uint32_t [TextureAttributes](#)
One or more texture attribute flags combined together.
- typedef struct [pxt::TextureParameters](#) [TextureParameters](#)
Texture parameters and characteristics.
- typedef struct [Texture_t](#) [Texture_t](#)
Texture that contains image data typically stored in GPU memory.
- typedef struct [Surface_t](#) [Surface_t](#)
Raster surface that contains image in memory for pixel manipulation.
- typedef uint32_t [CanvasAttributes](#)
One or more canvas attributes that define rendering behavior.
- typedef struct [pxt::SignedDistanceField](#) [SignedDistanceField](#)
Signed Distance Field (SDF) parameters.
- typedef struct [pxt::CanvasSamplerState](#) [CanvasSamplerState](#)
Sampler parameters that can be easily configured by the canvas.
- typedef struct [pxt::ImageRegion](#) [ImageRegion](#)
An image region defined by its texture number and bounding rectangle on that texture.
- typedef struct [Canvas_t](#) [Canvas_t](#)

- Canvas that can render geometry either in 2D or on a particular plane in 3D world.*

 - typedef struct [ImageAtlas_t](#) [ImageAtlas_t](#)
Image atlas, containing sub-images that can be rendered with canvas.
 - typedef uint32_t [UnicodeChar](#)
32-bit Unicode character type
 - typedef struct [pxt::TextEntryRect](#) [TextEntryRect](#)
Individual text entry that will appear as rendered.
 - typedef struct [pxt::TextRenderModifiers](#) [TextRenderModifiers](#)
Modifier attributes that can be applied to the rendered text.
 - typedef struct [pxt::FontEffect](#) [FontEffect](#)
Attributes and characteristics that define how font glyphs are rasterized.
 - typedef struct [pxt::FontParameters](#) [FontParameters](#)
Parameters that define the appearance and important characteristics of the font.
 - typedef struct [TextRenderer_t](#) [TextRenderer_t](#)
Text renderer utility that can draw text on the canvas.
 - typedef struct [TextModeller_t](#) [TextModeller_t](#)
2D and 3D text rendering module.
 - typedef struct [Grapher_t](#) [Grapher_t](#)
3D graph plotting module.
 - typedef struct [MeshModel_t](#) [MeshModel_t](#)
3D mesh model that has vertex and optionally index buffers for rendering.
 - typedef int32_t(* [MeshLoadSaveFeedback](#)) (char const *message, int32_t progress, void *user)
 - typedef struct [pxt::MeshBufferEntry](#) [MeshBufferEntry](#)
Calculated values for a single mesh vertex that can be modified by callback.
 - typedef struct [pxt::MeshMetaTagPortion](#) [MeshMetaTagPortion](#)
A portion of the mesh corresponding to a particular tag.
 - typedef struct [MeshMetaTag_t](#) [MeshMetaTag_t](#)
Meta-tag, which describes a certain important axis-aligned bounding area inside a mesh.
 - typedef struct [MeshMetaTags_t](#) [MeshMetaTags_t](#)
List of meta-tags that describe important axis-aligned bounding areas inside a mesh.
 - typedef struct [MeshBuffer_t](#) [MeshBuffer_t](#)
Buffer that contains 3D mesh information.
 - typedef struct [GaussianBlur_t](#) [GaussianBlur_t](#)
Gaussian Blur module.
 - typedef struct [KawaseBlur_t](#) [KawaseBlur_t](#)
Kawase Blur module.
 - typedef struct [ColorDithering_t](#) [ColorDithering_t](#)
Color dithering module.
 - typedef struct [pxt::SelectionHighlightParameters](#) [SelectionHighlightParameters](#)
Parameters that define how selection highlight technique is performed.
 - typedef struct [SelectionHighlight_t](#) [SelectionHighlight_t](#)
Selection highlight module.
 - typedef struct [pxt::FogParameters](#) [FogParameters](#)
Structure containing spatial fog characteristics.
 - typedef struct [SpatialFog_t](#) [SpatialFog_t](#)
Module that performs both ground fog and distance-based fog simultaneously.
 - typedef struct [pxt::ShadowParameters](#) [ShadowParameters](#)
Parameters that define how shadows are rendered.
 - typedef struct [pxt::ToneMappingBloom](#) [ToneMappingBloom](#)
Parameters that define behavior of tone-mapping and bloom effects.
 - typedef struct [pxt::AmbientOcclusionParameters](#) [AmbientOcclusionParameters](#)

- Parameters that define how ambient occlusion is performed.*
- typedef struct [pxt::ParallaxMappingParameters](#) [ParallaxMappingParameters](#)
Parameters that define how parallax mapping is performed.
- typedef struct [pxt::SceneLight](#) [SceneLight](#)
Light source parameters in 3D scene.
- typedef struct [pxt::ObjectMaterial](#) [ObjectMaterial](#)
Material that describes a particular object in 3D world.
- typedef struct [pxt::OceanWavesParameters](#) [OceanWavesParameters](#)
Parameters that define the appearance of waves simulation.
- typedef struct [pxt::OceanMaterial](#) [OceanMaterial](#)
Material that describes ocean water surface in a 3D simulation.
- typedef uint32_t [TextureCabinetAttributes](#)
Cumulative rendering attributes that affect how the scene is filtered.
- typedef uint32_t [SceneAttributes](#)
Rendering attributes that define what type of resources are used and how rendering is performed.
- typedef struct [SceneLights_t](#) [SceneLights_t](#)
Container for storing 3D scene lights.
- typedef struct [ObjectMaterials_t](#) [ObjectMaterials_t](#)
Container for storing 3D object materials.
- typedef struct [TextureCabinet_t](#) [TextureCabinet_t](#)
A collection of drawable textures used to render the scene.
- typedef struct [ShadowCaster_t](#) [ShadowCaster_t](#)
A source that casts shadows that can be shared among multiple light sources.
- typedef struct [ShadowCastingAtlas_t](#) [ShadowCastingAtlas_t](#)
Module that manages shadow maps produced by one or more shadow casters.
- typedef struct [Scene_t](#) [Scene_t](#)
3D rendering scene.
- typedef struct [OceanSimulation_t](#) [OceanSimulation_t](#)
3D ocean semi-infinite wave field rendering module.
- typedef uintptr_t [ObjectModelID](#)
Data type for storing unique object identification numbers.
- typedef struct [pxt::MeshAligns](#) [MeshAligns](#)
Alignment for all three axes that determine the placement of mesh around its model.
- typedef struct [ObjectModel_t](#) [ObjectModel_t](#)
Object model represented in a working 3D environment.
- typedef struct [ObjectModels_t](#) [ObjectModels_t](#)
Container and manager for working with multiple 3D scene objects.
- typedef struct [ObjectModelsIterator_t](#) [ObjectModelsIterator_t](#)
Iterator for retrieving individual objects from an object model's container.
- typedef struct [ObjectModelView_t](#) [ObjectModelView_t](#)
A container that represents a particular view that watches objects in the world.
- typedef int32_t(* [ObjectModelCompareFunc](#)) ([ObjectModel_t](#) const *object1, [ObjectModel_t](#) const *object2, void *user)
Comparison function for a pair of objects in a 3D scene.
- typedef struct [MeshVoxel_t](#) [MeshVoxel_t](#)
Voxel representation of a 3D mesh model.
- typedef void(* [MeshVoxelVisualizeFunc](#)) ([Vector](#) const *position, [Vector](#) const *size, [FloatColor](#) const *color, void *user)
Mesh voxel rendering function.
- typedef struct [pxt::SceneMeshMaterialShading](#) [SceneMeshMaterialShading](#)
Shading parameters for scene mesh material.

- typedef struct [pxt::SceneMeshMaterialRange](#) [SceneMeshMaterialRange](#)
A range of indices in the mesh that a material applies to.
- typedef struct [SceneMeshMaterial_t](#) [SceneMeshMaterial_t](#)
Material that describes how a portion of a scene mesh geometry should look like.
- typedef struct [SceneMeshMaterials_t](#) [SceneMeshMaterials_t](#)
Container for scene mesh material that describe the appearance of scene mesh geometry.
- typedef struct [pxt::SceneMeshLatch](#) [SceneMeshLatch](#)
A latch in 3D scene mesh, which describes a bone joint connection or a movement waypoint.
- typedef struct [SceneMeshLatches_t](#) [SceneMeshLatches_t](#)
A collection of marks in a 3D mesh to denote bone joint connections and movement waypoints.
- typedef struct [SceneMesh_t](#) [SceneMesh_t](#)
High-level wrapper around a rendereable and pickable mesh in 3D scene.
- typedef struct [SceneMeshes_t](#) [SceneMeshes_t](#)
Container for high-level wrappers of rendereable and pickable objects in 3D scene.
- typedef struct [pxt::ApplicationConfiguration](#) [ApplicationConfiguration](#)
Structure that contains all the parameters necessary to create a new application and its window.
- typedef struct [Application_t](#) [Application_t](#)
Application helper module.
- typedef void(* [EventFunc](#)) ([Application_t](#) *application, void *user)
Basic application event.
- typedef [LibraryBool](#)(* [BoolFunc](#)) ([Application_t](#) *application, void *user)
Application creation event.
- typedef void(* [MouseEvent](#)) ([Application_t](#) *application, [MouseEvent](#) event, [MouseButton](#) button, [Point](#) const *position, void *user)
Application mouse handling event.
- typedef void(* [KeyboardFunc](#)) ([Application_t](#) *application, [KeyEvent](#) event, int32_t virtualKey, uint16_t key↔ Code, void *user)
Application keyboard handling event.
- typedef void(* [StatusFunc](#)) ([Application](#) *application, [ApplicationEvent](#) event, void *user)
Event function for application status change events.
- typedef [LibraryBool](#)(* [EventHookFunc](#)) ([Application_t](#) *application, void const *event, void *user)
Custom application message handling event.
- typedef struct [pxt::ApplicationEvents](#) [ApplicationEvents](#)
- typedef struct [pxt::CameraConstraints](#) [CameraConstraints](#)
- typedef struct [ActorCamera_t](#) [ActorCamera_t](#)
- typedef struct [Timer_t](#) [Timer_t](#)
Multimedia timer that can accurately calculate latency, frame rate and provide time-based processing.
- typedef union [pxt::WidgetPropertyData](#) [WidgetPropertyData](#)
A flexible structure for storing property data.
- typedef struct [pxt::WidgetProperty](#) [WidgetProperty](#)
Information about a widget property.
- typedef struct [Widget_t](#) [Widget_t](#)
A widget that represents a basic building block of user interface.
- typedef [LibraryBool](#)(* [WidgetExternalEvent](#)) (struct [Widget_t](#) *widget, char const *event, void *user)
Widget's external callback event type.

Functions

- uint32_t [deviceAttributeExtensions](#) (uint8_t const extensionLevel)

5.1.1 Typedef Documentation

5.1.1.1 ActorCamera_t

```
typedef struct ActorCamera_t ActorCamera_t
```

Camera module that can represent both first-person and third-person views for 3D world exploration and object manipulation.

5.1.1.2 AmbientOcclusionParameters

```
typedef struct pxt::AmbientOcclusionParameters AmbientOcclusionParameters
```

Parameters that define how ambient occlusion is performed.

5.1.1.3 Application_t

```
typedef struct Application_t Application_t
```

Application helper module.

5.1.1.4 ApplicationConfiguration

```
typedef struct pxt::ApplicationConfiguration ApplicationConfiguration
```

Structure that contains all the parameters necessary to create a new application and its window.

5.1.1.5 ApplicationEvents

```
typedef struct pxt::ApplicationEvents ApplicationEvents
```

Events that are invoked by the application. Each of these fields is optional, so if a particular event is set to NULL, it will not be invoked and default handler will be used.

5.1.1.6 BoolFunc

```
typedef LibraryBool(* BoolFunc) (Application_t *application, void *user)
```

Application creation event.

5.1.1.7 Buffer_t

```
typedef struct Buffer_t Buffer_t
```

Buffer that is typically stored on graphics hardware.

5.1.1.8 CameraConstraints

```
typedef struct pxt::CameraConstraints CameraConstraints
```

Camera position, rotation and distance constraints. If the difference between min and max constraint on a particular axis is almost zero or less, then no constraint is applied.

5.1.1.9 Canvas_t

```
typedef struct Canvas_t Canvas_t
```

Canvas that can render geometry either in 2D or on a particular plane in 3D world.

5.1.1.10 CanvasAttributes

```
typedef uint32_t CanvasAttributes
```

One or more canvas attributes that define rendering behavior.

5.1.1.11 CanvasSamplerState

```
typedef struct pxt::CanvasSamplerState CanvasSamplerState
```

Sampler parameters that can be easily configured by the canvas.

5.1.1.12 ColorDithering_t

```
typedef struct ColorDithering_t ColorDithering_t
```

Color dithering module.

5.1.1.13 ComputeBindTextureFormat

```
typedef struct pxt::ComputeBindTextureFormat ComputeBindTextureFormat
```

Compute texture parameters and characteristics.

5.1.1.14 ComputeProgram_t

```
typedef struct ComputeProgram_t ComputeProgram_t
```

Compute shader program for general processing on GPU (GPGPU).

5.1.1.15 DeviceAttributes

```
typedef uint32_t DeviceAttributes
```

One or more device attribute flags combined (ORed) together.

5.1.1.16 DeviceBehavior

```
typedef uint32_t DeviceBehavior
```

Device behavior flags that affect the rendering code path.

5.1.1.17 DeviceLegacyBits

```
typedef uint32_t DeviceLegacyBits
```

Legacy feature bits that define older hardware features.

5.1.1.18 EventFunc

```
typedef void(* EventFunc) (Application_t *application, void *user)
```

Basic application event.

5.1.1.19 EventHookFunc

```
typedef LibraryBool(* EventHookFunc) (Application_t *application, void const *event, void *user)
```

Custom application message handling event.

5.1.1.20 FogParameters

```
typedef struct pxt::FogParameters FogParameters
```

Structure containing spatial fog characteristics.

5.1.1.21 FontEffect

```
typedef struct pxt::FontEffect FontEffect
```

Attributes and characteristics that define how font glyphs are rasterized.

5.1.1.22 FontParameters

```
typedef struct pxt::FontParameters FontParameters
```

Parameters that define the appearance and important characteristics of the font.

5.1.1.23 GaussianBlur_t

```
typedef struct GaussianBlur_t GaussianBlur_t
```

Gaussian Blur module.

5.1.1.24 Grapher_t

```
typedef struct Grapher_t Grapher_t
```

3D graph plotting module.

5.1.1.25 ImageAtlas_t

```
typedef struct ImageAtlas_t ImageAtlas_t
```

Image atlas, containing sub-images that can be rendered with canvas.

5.1.1.26 ImageRegion

```
typedef struct pxt::ImageRegion ImageRegion
```

An image region defined by its texture number and bounding rectangle on that texture.

5.1.1.27 KawaseBlur_t

```
typedef struct KawaseBlur_t KawaseBlur_t
```

Kawase Blur module.

5.1.1.28 KeyboardFunc

```
typedef void(* KeyboardFunc) (Application_t *application, KeyEvent event, int32_t virtualKey,  
uint16_t keyCode, void *user)
```

Application keyboard handling event.

5.1.1.29 LibraryBool

```
typedef uint8_t LibraryBool
```

Boolean type returned by the library.

5.1.1.30 MeshAligns

```
typedef struct pxt::MeshAligns MeshAligns
```

Alignment for all three axes that determine the placement of mesh around its model.

5.1.1.31 MeshBuffer_t

```
typedef struct MeshBuffer_t MeshBuffer_t
```

Buffer that contains 3D mesh information.

5.1.1.32 MeshBufferEntry

```
typedef struct pxt::MeshBufferEntry MeshBufferEntry
```

Calculated values for a single mesh vertex that can be modified by callback.

5.1.1.33 MeshLoadSaveFeedback

```
typedef int32_t(* MeshLoadSaveFeedback) (char const *message, int32_t progress, void *user)
```

Notification event regarding mesh loading or saving process. The notification callback must return 0 to indicate that the process should continue; a non-zero value means that the loading process should be interrupted.

5.1.1.34 MeshMetaTag_t

```
typedef struct MeshMetaTag_t MeshMetaTag_t
```

Meta-tag, which describes a certain important axis-aligned bounding area inside a mesh.

5.1.1.35 MeshMetaTagPortion

```
typedef struct pxt::MeshMetaTagPortion MeshMetaTagPortion
```

A portion of the mesh corresponding to a particular tag.

5.1.1.36 MeshMetaTags_t

```
typedef struct MeshMetaTags_t MeshMetaTags_t
```

List of meta-tags that describe important axis-aligned bounding areas inside a mesh.

5.1.1.37 MeshModel_t

```
typedef struct MeshModel_t MeshModel_t
```

3D mesh model that has vertex and optionally index buffers for rendering.

5.1.1.38 MeshVoxel_t

```
typedef struct MeshVoxel_t MeshVoxel_t
```

Voxel representation of a 3D mesh model.

5.1.1.39 MeshVoxelVisualizeFunc

```
typedef void(* MeshVoxelVisualizeFunc) (Vector const *position, Vector const *size, FloatColor  
const *color, void *user)
```

Mesh voxel rendering function.

5.1.1.40 MouseFunc

```
typedef void(* MouseFunc) (Application_t *application, MouseEvent event, MouseButton button,  
Point const *position, void *user)
```

Application mouse handling event.

5.1.1.41 ObjectMaterial

```
typedef struct pxt::ObjectMaterial ObjectMaterial
```

Material that describes a particular object in 3D world.

5.1.1.42 ObjectMaterials_t

```
typedef struct ObjectMaterials_t ObjectMaterials_t
```

Container for storing 3D object materials.

5.1.1.43 ObjectModel_t

```
typedef struct ObjectModel_t ObjectModel_t
```

Object model represented in a working 3D environment.

5.1.1.44 ObjectModelCompareFunc

```
typedef int32_t(* ObjectModelCompareFunc) (ObjectModel_t const *object1, ObjectModel_t const *object2, void *user)
```

Comparison function for a pair of objects in a 3D scene.

5.1.1.45 ObjectModelID

```
typedef uintptr_t ObjectModelID
```

Data type for storing unique object identification numbers.

5.1.1.46 ObjectModels_t

```
typedef struct ObjectModels_t ObjectModels_t
```

Container and manager for working with multiple 3D scene objects.

5.1.1.47 ObjectModelsIterator_t

```
typedef struct ObjectModelsIterator_t ObjectModelsIterator_t
```

Iterator for retrieving individual objects from an object model's container.

5.1.1.48 ObjectModelView_t

```
typedef struct ObjectModelView_t ObjectModelView_t
```

A container that represents a particular view that watches objects in the world.

5.1.1.49 ObjectPayload

```
typedef void* ObjectPayload
```

Custom object's payload.

5.1.1.50 OceanMaterial

```
typedef struct pxt::OceanMaterial OceanMaterial
```

Material that describes ocean water surface in a 3D simulation.

5.1.1.51 OceanSimulation_t

```
typedef struct OceanSimulation_t OceanSimulation_t
```

3D ocean semi-infinite wave field rendering module.

5.1.1.52 OceanWavesParameters

```
typedef struct pxt::OceanWavesParameters OceanWavesParameters
```

Parameters that define the appearance of waves simulation.

5.1.1.53 ParallaxMappingParameters

```
typedef struct pxt::ParallaxMappingParameters ParallaxMappingParameters
```

Parameters that define how parallax mapping is performed.

5.1.1.54 Program_t

```
typedef struct Program_t Program_t
```

Shader program that is typically executed on graphics hardware.

5.1.1.55 ProgramElement

```
typedef struct pxt::ProgramElement ProgramElement
```

Structure that describes a single physical element of shader program.

5.1.1.56 ProgramVariable

```
typedef struct pxt::ProgramVariable ProgramVariable
```

Structure that describes a single (uniform) variable in a shader program.

5.1.1.57 RenderingState

```
typedef struct pxt::RenderingState RenderingState
```

Parameters that define depth, stencil, rasterizer and blending operations.

5.1.1.58 Sampler_t

```
typedef struct Sampler_t Sampler_t
```

Sampler object that defines how texture is read by the shaders.

5.1.1.59 SamplerState

```
typedef struct pxt::SamplerState SamplerState
```

Sampler parameters that are associated with a particular texture unit.

5.1.1.60 Scene_t

```
typedef struct Scene_t Scene_t
```

3D rendering scene.

5.1.1.61 SceneAttributes

```
typedef uint32_t SceneAttributes
```

Rendering attributes that define what type of resources are used and how rendering is performed.

5.1.1.62 SceneLight

```
typedef struct pxt::SceneLight SceneLight
```

Light source parameters in 3D scene.

5.1.1.63 SceneLights_t

```
typedef struct SceneLights_t SceneLights_t
```

Container for storing 3D scene lights.

5.1.1.64 SceneMesh_t

```
typedef struct SceneMesh_t SceneMesh_t
```

High-level wrapper around a rendereable and pickable mesh in 3D scene.

5.1.1.65 SceneMeshes_t

```
typedef struct SceneMeshes_t SceneMeshes_t
```

Container for high-level wrappers of rendereable and pickable objects in 3D scene.

5.1.1.66 SceneMeshLatch

```
typedef struct pxt::SceneMeshLatch SceneMeshLatch
```

A latch in 3D scene mesh, which describes a bone joint connection or a movement waypoint.

5.1.1.67 SceneMeshLatches_t

```
typedef struct SceneMeshLatches_t SceneMeshLatches_t
```

A collection of marks in a 3D mesh to denote bone joint connections and movement waypoints.

5.1.1.68 SceneMeshMaterial_t

```
typedef struct SceneMeshMaterial_t SceneMeshMaterial_t
```

Material that describes how a portion of a scene mesh geometry should look like.

5.1.1.69 SceneMeshMaterialRange

```
typedef struct pxt::SceneMeshMaterialRange SceneMeshMaterialRange
```

A range of indices in the mesh that a material applies to.

5.1.1.70 SceneMeshMaterials_t

```
typedef struct SceneMeshMaterials_t SceneMeshMaterials_t
```

Container for scene mesh material that describe the appearance of scene mesh geometry.

5.1.1.71 SceneMeshMaterialShading

```
typedef struct pxt::SceneMeshMaterialShading SceneMeshMaterialShading
```

Shading parameters for scene mesh material.

5.1.1.72 SelectionHighlight_t

```
typedef struct SelectionHighlight_t SelectionHighlight_t
```

Selection highlight module.

5.1.1.73 SelectionHighlightParameters

```
typedef struct pxt::SelectionHighlightParameters SelectionHighlightParameters
```

Parameters that define how selection highlight technique is performed.

5.1.1.74 ShadowCaster_t

```
typedef struct ShadowCaster_t ShadowCaster_t
```

A source that casts shadows that can be shared among multiple light sources.

5.1.1.75 ShadowCastingAtlas_t

```
typedef struct ShadowCastingAtlas_t ShadowCastingAtlas_t
```

Module that manages shadow maps produced by one or more shadow casters.

5.1.1.76 ShadowParameters

```
typedef struct pxt::ShadowParameters ShadowParameters
```

Parameters that define how shadows are rendered.

5.1.1.77 SignedDistanceField

```
typedef struct pxt::SignedDistanceField SignedDistanceField
```

Signed Distance Field (SDF) parameters.

5.1.1.78 SpatialFog_t

```
typedef struct SpatialFog_t SpatialFog_t
```

Module that performs both ground fog and distance-based fog simultaneously.

5.1.1.79 StatusFunc

```
typedef void(* StatusFunc) (Application *application, ApplicationEvent event, void *user)
```

Event function for application status change events.

5.1.1.80 Surface_t

```
typedef struct Surface_t Surface_t
```

Raster surface that contains image in memory for pixel manipulation.

5.1.1.81 TextEntryRect

```
typedef struct pxt::TextEntryRect TextEntryRect
```

Individual text entry that will appear as rendered.

5.1.1.82 TextModeller_t

```
typedef struct TextModeller_t TextModeller_t
```

2D and 3D text rendering module.

5.1.1.83 TextRenderer_t

```
typedef struct TextRenderer_t TextRenderer_t
```

Text renderer utility that can draw text on the canvas.

5.1.1.84 TextRenderModifiers

```
typedef struct pxt::TextRenderModifiers TextRenderModifiers
```

Modifier attributes that can be applied to the rendered text.

5.1.1.85 Texture_t

```
typedef struct Texture_t Texture_t
```

Texture that contains image data typically stored in GPU memory.

5.1.1.86 TextureAttributes

```
typedef uint32_t TextureAttributes
```

One or more texture attribute flags combined together.

5.1.1.87 TextureCabinet_t

```
typedef struct TextureCabinet_t TextureCabinet_t
```

A collection of drawable textures used to render the scene.

5.1.1.88 TextureCabinetAttributes

```
typedef uint32_t TextureCabinetAttributes
```

Cumulative rendering attributes that affect how the scene is filtered.

5.1.1.89 TextureParameters

```
typedef struct pxt::TextureParameters TextureParameters
```

Texture parameters and characteristics.

5.1.1.90 Timer_t

```
typedef struct Timer_t Timer_t
```

Multimedia timer that can accurately calculate latency, frame rate and provide time-based processing.

5.1.1.91 ToneMappingBloom

```
typedef struct pxt::ToneMappingBloom ToneMappingBloom
```

Parameters that define behavior of tone-mapping and bloom effects.

5.1.1.92 UnicodeChar

```
typedef uint32_t UnicodeChar
```

32-bit Unicode character type

5.1.1.93 UntypedHandle

```
typedef void* UntypedHandle
```

Generic untyped handle.

5.1.1.94 VertexElement

```
typedef struct pxt::VertexElement VertexElement
```

Structure that describes the layout of a single buffer element.

5.1.1.95 Widget_t

```
typedef struct Widget_t Widget_t
```

A widget that represents a basic building block of user interface.

5.1.1.96 WidgetExternalEvent

```
typedef LibraryBool(* WidgetExternalEvent) (struct Widget_t *widget, char const *event, void *user)
```

Widget's external callback event type.

5.1.1.97 WidgetProperty

```
typedef struct pxt::WidgetProperty WidgetProperty
```

Information about a widget property.

5.1.1.98 WidgetPropertyData

```
typedef union pxt::WidgetPropertyData WidgetPropertyData
```

A flexible structure for storing property data.

5.1.2 Function Documentation

5.1.2.1 deviceAttributeExtensions()

```
uint32_t deviceAttributeExtensions (  
    uint8_t const extensionLevel ) [inline]
```

Returns device attribute bits that define level of extensions to be queried. This value can vary from zero (0), which means no extensions (other than debug context) should be used to three (3), which means all extensions. Note that a value of one is added to the given extension level before returning - this means that extension level bits are present. If all extension level bits are unset (default), this is the same as maximum level, meaning that all available extensions should be used.

Chapter 6

Class Documentation

6.1 AmbientOcclusionParameters Struct Reference

Parameters that define how ambient occlusion is performed.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- float [radius](#)
Radius of the occlusion.
- int32_t [samples](#)
Number of kernel samples.
- float [attenuation](#)
Linear attenuation.
- float [contrast](#)
Contrast of the occlusion.
- float [depthBias](#)
Bias value to resolve depth precision issues.
- float [kernelBias](#)
Bias value to prevent sampling close to tangent surface.
- float [blurSigma](#)
Sigma coefficient for blur filter.
- float [blurFallOff](#)
Fall-off coefficient for detecting depth discontinuities in blur filter.
- float [strength](#)
Strength of the effect.

6.1.1 Detailed Description

Parameters that define how ambient occlusion is performed.

6.1.2 Member Data Documentation

6.1.2.1 attenuation

`float attenuation`

Linear attenuation.

6.1.2.2 blurFallOff

`float blurFallOff`

Fall-off coefficient for detecting depth discontinuities in blur filter.

6.1.2.3 blurSigma

`float blurSigma`

Sigma coefficient for blur filter.

6.1.2.4 contrast

`float contrast`

Contrast of the occlusion.

6.1.2.5 depthBias

`float depthBias`

Bias value to resolve depth precision issues.

6.1.2.6 kernelBias

`float kernelBias`

Bias value to prevent sampling close to tangent surface.

6.1.2.7 radius

`float radius`

Radius of the occlusion.

6.1.2.8 samples

```
int32_t samples
```

Number of kernel samples.

6.1.2.9 strength

```
float strength
```

Strength of the effect.

The documentation for this struct was generated from the following file:

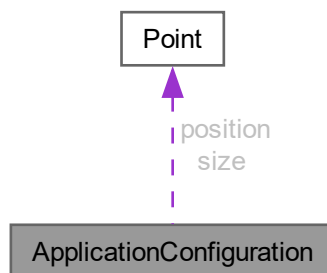
- [Afterwarp.Structs.h](#)

6.2 ApplicationConfiguration Struct Reference

Structure that contains all the parameters necessary to create a new application and its window.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for ApplicationConfiguration:



Public Attributes

- [Point size](#)
Initial size of the application window.
- [ApplicationWindowState state](#)
Initial state of application window.
- [Point position](#)
- union {
 - struct {
 - [UntypedHandle handleInstance](#)
Handle of application's instance.
 - [UntypedHandle handleIconSmall](#)
Handle of application's "small" icon.
 - [UntypedHandle handleIconBig](#)
Handle of application's "big" icon.
 - char [windowClassName](#) [32]
Name of application window class.
 - } [win](#)
Fields specific to Windows platform.
 - struct {
 - uint8_t [renderingType](#)
Type of rendering used with the application.
 - int32_t [parameterCount](#)
Number of command-line arguments.
 - char ** [parameterStrings](#)
List of command-line arguments.
 - } [nx](#)
Fields specific to Linux platform.
- } [startup](#)

Platform-specific fields that is required for startup.

6.2.1 Detailed Description

Structure that contains all the parameters necessary to create a new application and its window.

6.2.2 Member Data Documentation

6.2.2.1 handleIconBig

[UntypedHandle](#) handleIconBig

Handle of application's "big" icon.

6.2.2.2 handleIconSmall

[UntypedHandle](#) handleIconSmall

Handle of application's "small" icon.

6.2.2.3 handleInstance

`UntypedHandle` handleInstance

Handle of application's instance.

6.2.2.4 [struct]

```
struct { ... } nx
```

Fields specific to Linux platform.

6.2.2.5 parameterCount

`int32_t` parameterCount

Number of command-line arguments.

6.2.2.6 parameterStrings

`char**` parameterStrings

List of command-line arguments.

6.2.2.7 position

`Point` position

Initial position of application window. If both X and Y are set to INT32_MAX, then a default position will be used.

6.2.2.8 renderingType

`uint8_t` renderingType

Type of rendering used with the application.

6.2.2.9 size

`Point` size

Initial size of the application window.

6.2.2.10 [union]

```
union { ... } startup
```

Platform-specific fields that is required for startup.

6.2.2.11 state

`ApplicationWindowState` state

Initial state of application window.

6.2.2.12 [struct]

```
struct { ... } win
```

Fields specific to Windows platform.

6.2.2.13 windowClassName

```
char windowClassName[32]
```

Name of application window class.

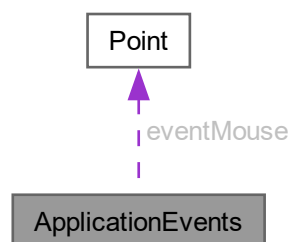
The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.3 ApplicationEvents Struct Reference

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for ApplicationEvents:



Public Attributes

- [BoolFunc eventCreate](#)
- [EventFunc eventDestroy](#)
Event invoked during application shutdown to release the resources.
- [EventFunc eventRender](#)
Event invoked when application needs to render contents of the window.
- [MouseFunc eventMouse](#)
Event invoked when a mouse event has been registered.
- [KeyboardFunc eventKey](#)
Event invoked when a key event has been registered.
- [StatusFunc eventStatus](#)
Event invoked when an application status has changed.
- [BoolFunc eventIdle](#)
Event invoked every time the application is idle.
- [EventFunc eventResize](#)
Event invoked when application window is resized.
- [EventHookFunc eventHook](#)
Event hook invoked to process application events.

6.3.1 Detailed Description

Events that are invoked by the application. Each of these fields is optional, so if a particular event is set to NULL, it will not be invoked and default handler will be used.

6.3.2 Member Data Documentation

6.3.2.1 eventCreate

[BoolFunc](#) eventCreate

Event invoked during startup to load application resources. If this function returns `false`, application creation will fail.

6.3.2.2 eventDestroy

[EventFunc](#) eventDestroy

Event invoked during application shutdown to release the resources.

6.3.2.3 eventHook

[EventHookFunc](#) eventHook

Event hook invoked to process application events.

6.3.2.4 eventIdle

`BoolFunc eventIdle`

Event invoked every time the application is idle.

6.3.2.5 eventKey

`KeyboardFunc eventKey`

Event invoked when a key event has been registered.

6.3.2.6 eventMouse

`MouseFunc eventMouse`

Event invoked when a mouse event has been registered.

6.3.2.7 eventRender

`EventFunc eventRender`

Event invoked when application needs to render contents of the window.

6.3.2.8 eventResize

`EventFunc eventResize`

Event invoked when application window is resized.

6.3.2.9 eventStatus

`StatusFunc eventStatus`

Event invoked when an application status has changed.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.4 Blend Struct Reference

Blending parameters that are specific to a certain component, or a portion of color.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- [BlendFactor source](#)
Blending factor used for the source component.
- [BlendFactor dest](#)
Blending factor used for the destination component.
- [BlendOp op](#)
Blending operation used between source and destination components.

6.4.1 Detailed Description

Blending parameters that are specific to a certain component, or a portion of color.

6.4.2 Member Data Documentation

6.4.2.1 dest

[BlendFactor](#) dest

Blending factor used for the destination component.

6.4.2.2 op

[BlendOp](#) op

Blending operation used between source and destination components.

6.4.2.3 source

[BlendFactor](#) source

Blending factor used for the source component.

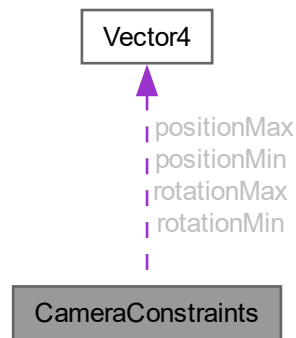
The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.5 CameraConstraints Struct Reference

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for CameraConstraints:



Public Attributes

- [Vector4 positionMin](#)
Minimum position allowed for camera positioning. "w" component is a minimum distance constraint.
- [Vector4 positionMax](#)
Maximum position allowed for camera positioning. "w" component is a maximum distance constraint.
- [Vector4 rotationMin](#)
Minimum camera rotation angles. "w" component is a minimum angle for Lenticular printing rotation.
- [Vector4 rotationMax](#)
Maximum camera rotation angles. "w" component is a maximum angle for Lenticular printing rotation.

6.5.1 Detailed Description

Camera position, rotation and distance constraints. If the difference between min and max constraint on a particular axis is almost zero or less, then no constraint is applied.

6.5.2 Member Data Documentation

6.5.2.1 positionMax

[Vector4](#) positionMax

Maximum position allowed for camera positioning. "w" component is a maximum distance constraint.

6.5.2.2 positionMin

`Vector4` positionMin

Minimum position allowed for camera positioning. "w" component is a minimum distance constraint.

6.5.2.3 rotationMax

`Vector4` rotationMax

Maximum camera rotation angles. "w" component is a maximum angle for Lenticular printing rotation.

6.5.2.4 rotationMin

`Vector4` rotationMin

Minimum camera rotation angles. "w" component is a minimum angle for Lenticular printing rotation.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.6 CanvasSamplerState Struct Reference

Sampler parameters that can be easily configured by the canvas.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- [TextureFilter filterMin](#)
Minification texture filtering.
- [TextureFilter filterMag](#)
Magnification texture filtering.
- [TextureFilter filterMip](#)
Mipmapping texture filtering.
- [TextureAddress addressU](#)
Horizontal texture coordinate addressing.
- [TextureAddress addressV](#)
Vertical texture coordinate addressing.
- [Color borderColor](#)
Special color constant to be used for TextureAddress::Border addressing mode.

6.6.1 Detailed Description

Sampler parameters that can be easily configured by the canvas.

6.6.2 Member Data Documentation

6.6.2.1 addressU

`TextureAddress` addressU

Horizontal texture coordinate addressing.

6.6.2.2 addressV

`TextureAddress` addressV

Vertical texture coordinate addressing.

6.6.2.3 borderColor

`Color` borderColor

Special color constant to be used for `TextureAddress::Border` addressing mode.

6.6.2.4 filterMag

`TextureFilter` filterMag

Magnification texture filtering.

6.6.2.5 filterMin

`TextureFilter` filterMin

Minification texture filtering.

6.6.2.6 filterMip

`TextureFilter` filterMip

Mipmapping texture filtering.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.7 ColorPair Struct Reference

```
#include <Afterwarp.Types.h>
```

Public Member Functions

- [ColorPair](#) ([Color](#) first, [Color](#) second)
Creates two 32-bit ARGB color gradient from specified pair of values.
- [ColorPair](#) ([Color](#) color)
Creates two 32-bit ARGB color gradient with both values set to specified color.

Public Attributes

- [Color](#) first
First color entry, which can be reinterpreted as top or left color depending on context.
- [Color](#) second
Second color entry, which can be reinterpreted as bottom or right color depending on context.

6.7.1 Detailed Description

A combination of two colors, primarily used for displaying text with the first color being on top and the second being on bottom. The format for specifying colors is defined as `ARGB8`).

6.7.2 Constructor & Destructor Documentation

6.7.2.1 [ColorPair\(\)](#) [1/2]

```
ColorPair (  
    Color first,  
    Color second ) [inline]
```

Creates two 32-bit ARGB color gradient from specified pair of values.

6.7.2.2 [ColorPair\(\)](#) [2/2]

```
ColorPair (  
    Color color ) [inline]
```

Creates two 32-bit ARGB color gradient with both values set to specified color.

6.7.3 Member Data Documentation

6.7.3.1 first

```
Color first
```

First color entry, which can be reinterpreted as top or left color depending on context.

6.7.3.2 second

`Color second`

Second color entry, which can be reinterpreted as bottom or right color depending on context.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.8 ColorRect Struct Reference

```
#include <Afterwarp.Types.h>
```

Public Attributes

- [Color topLeft](#)
Color corresponding to top / left corner.
- [Color topRight](#)
Color corresponding to top / right corner.
- [Color bottomRight](#)
Color corresponding to bottom / right corner.
- [Color bottomLeft](#)
Color corresponding to bottom / left corner.

6.8.1 Detailed Description

A rectangle of four colors, primarily used for displaying colored quads, where each color corresponds to top/left, top/right, bottom/right and bottom/left accordingly (clockwise). The format for specifying colors is defined as `ARGB8`.

6.8.2 Member Data Documentation

6.8.2.1 bottomLeft

`Color bottomLeft`

Color corresponding to bottom / left corner.

6.8.2.2 bottomRight

`Color bottomRight`

Color corresponding to bottom / right corner.

6.8.2.3 topLeft

`Color` topLeft

Color corresponding to top / left corner.

6.8.2.4 topRight

`Color` topRight

Color corresponding to top / right corner.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.9 ComputeBindTextureFormat Struct Reference

Compute texture parameters and characteristics.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- `uint32_t` [channel](#)
Channel to which the texture is bound to.
- `int32_t` [mipLevel](#)
Number of mipmap level that should be attached.
- `int32_t` [layer](#)
Layer number from a volume or array texture to bound (-1 means an entire texture should be bound).
- [ComputeTextureAccess](#) [access](#)
Type of access that will be used with the texture.
- [PixelFormat](#) [format](#)
Format of the texture's pixels.

6.9.1 Detailed Description

Compute texture parameters and characteristics.

6.9.2 Member Data Documentation

6.9.2.1 access

[ComputeTextureAccess](#) access

Type of access that will be used with the texture.

6.9.2.2 channel

```
uint32_t channel
```

Channel to which the texture is bound to.

6.9.2.3 format

```
PixelFormat format
```

Format of the texture's pixels.

6.9.2.4 layer

```
int32_t layer
```

Layer number from a volume or array texture to bound (-1 means an entire texture should be bound).

6.9.2.5 mipLevel

```
int32_t mipLevel
```

Number of mipmap level that should be attached.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.10 FloatColor Struct Reference

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float [red](#)
Red value ranging from 0.0 (no intensity) to 1.0 (fully intense).
- float [green](#)
Green value ranging from 0.0 (no intensity) to 1.0 (fully intense).
- float [blue](#)
Blue value ranging from 0.0 (no intensity) to 1.0 (fully intense).
- float [alpha](#)
Alpha-channel value ranging from 0.0 (translucent) to 1.0 (opaque).

6.10.1 Detailed Description

A special high-precision color value that has each individual component represented as 32-bit floating-point value in range of [0, 1]. Although components may have values outside of aforementioned range, such colors cannot be reliably displayed on the screen.

6.10.2 Member Data Documentation

6.10.2.1 alpha

```
float alpha
```

Alpha-channel value ranging from 0.0 (translucent) to 1.0 (opaque).

6.10.2.2 blue

```
float blue
```

Blue value ranging from 0.0 (no intensity) to 1.0 (fully intense).

6.10.2.3 green

```
float green
```

Green value ranging from 0.0 (no intensity) to 1.0 (fully intense).

6.10.2.4 red

```
float red
```

Red value ranging from 0.0 (no intensity) to 1.0 (fully intense).

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.11 FloatColorRGB Struct Reference

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float [red](#)
Red value ranging from 0.0 (no intensity) to 1.0 (fully intense).
- float [green](#)
Green value ranging from 0.0 (no intensity) to 1.0 (fully intense).
- float [blue](#)
Blue value ranging from 0.0 (no intensity) to 1.0 (fully intense).

6.11.1 Detailed Description

A three-component special high-precision color value that has each individual component represented as 32-bit floating-point value in range of [0, 1]. Although components may have values outside of aforementioned range, such colors cannot be reliably displayed on the screen.

6.11.2 Member Data Documentation

6.11.2.1 blue

```
float blue
```

Blue value ranging from 0.0 (no intensity) to 1.0 (fully intense).

6.11.2.2 green

```
float green
```

Green value ranging from 0.0 (no intensity) to 1.0 (fully intense).

6.11.2.3 red

```
float red
```

Red value ranging from 0.0 (no intensity) to 1.0 (fully intense).

The documentation for this struct was generated from the following file:

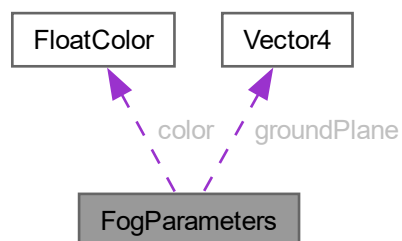
- [Afterwarp.Types.h](#)

6.12 FogParameters Struct Reference

Structure containing spatial fog characteristics.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for FogParameters:



Public Attributes

- [FloatColor](#) `color`
Fog color.
- `float` [opacity](#)
Fog opacity.
- [Vector4](#) `groundPlane`
Equation of the plane for ground fog.
- `float` [densityGround](#)
Ground fog density factor.
- `union` {
 `float` [distance](#)
 `float` [scattering](#)
};

Either a distance (far plane) for linear fog or an in-scattering parameter for exponential fog.
- `union` {
 `float` [bias](#)
 `float` [extinction](#)
};

Either a bias for linear fog or an extinction parameter for exponential fog.

6.12.1 Detailed Description

Structure containing spatial fog characteristics.

6.12.2 Member Data Documentation

6.12.2.1 [union]

```
union { ... }
```

Either a distance (far plane) for linear fog or an in-scattering parameter for exponential fog.

6.12.2.2 [union]

```
union { ... }
```

Either a bias for linear fog or an extinction parameter for exponential fog.

6.12.2.3 bias

```
float bias
```

6.12.2.4 color

`FloatColor` color

Fog color.

6.12.2.5 densityGround

`float` densityGround

Ground fog density factor.

6.12.2.6 distance

`float` distance

6.12.2.7 extinction

`float` extinction

6.12.2.8 groundPlane

`Vector4` groundPlane

Equation of the plane for ground fog.

6.12.2.9 opacity

`float` opacity

Fog opacity.

6.12.2.10 scattering

`float` scattering

The documentation for this struct was generated from the following file:

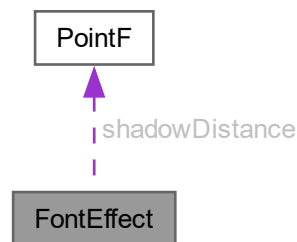
- [Afterwarp.Structs.h](#)

6.13 FontEffect Struct Reference

Attributes and characteristics that define how font glyphs are rasterized.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for FontEffect:



Public Attributes

- float [fillBrightness](#)
- float [fillOpacity](#)
- [FontBorder](#) [borderType](#)
Type of border that will appear around the text.
- float [borderThickness](#)
Thickness of the font's border in pixels.
- float [borderBrightness](#)
- float [borderOpacity](#)
- float [shadowSmoothness](#)
How strong the blur should be applied to letter's shadow (number of steps).
- [PointF](#) [shadowDistance](#)
Distance between letter and its shadow in pixels.
- float [shadowBrightness](#)
- float [shadowOpacity](#)
- float [signedFieldDistance](#)

6.13.1 Detailed Description

Attributes and characteristics that define how font glyphs are rasterized.

6.13.2 Member Data Documentation

6.13.2.1 [borderBrightness](#)

```
float borderBrightness
```

Brightness of the font's border in range of [0, 1], where zero means the border will appear completely dark and one means that the border will appear completely white.

6.13.2.2 borderOpacity

```
float borderOpacity
```

Opacity of the font's border in range of [0, 1], where zero means the border will appear completely transparent (that is, not appear at all) and one means that the border will appear completely opaque.

6.13.2.3 borderThickness

```
float borderThickness
```

Thickness of the font's border in pixels.

6.13.2.4 borderType

```
FontBorder borderType
```

Type of border that will appear around the text.

6.13.2.5 fillBrightness

```
float fillBrightness
```

Brightness of the letter's fill in range of [0, 1], where zero means the letter will appear completely dark and one means that letter will appear completely white.

6.13.2.6 fillOpacity

```
float fillOpacity
```

Opacity of the letter's fill in range of [0, 1], where zero means the letter will appear completely transparent (that is, not appear at all) and one means that the letter will appear completely opaque.

6.13.2.7 shadowBrightness

```
float shadowBrightness
```

Brightness of the letter's shadow in range of [0, 1], where zero means the shadow will appear completely dark and one means that the shadow will appear completely white.

6.13.2.8 shadowDistance

```
PointF shadowDistance
```

Distance between letter and its shadow in pixels.

6.13.2.9 shadowOpacity

```
float shadowOpacity
```

Opacity of the letter's shadow in range of [0, 1], where zero means the shadow will appear completely transparent (that is, not appear at all) and one means that the shadow will appear completely opaque.

6.13.2.10 shadowSmoothness

```
float shadowSmoothness
```

How strong the blur should be applied to letter's shadow (number of steps).

6.13.2.11 signedFieldDistance

```
float signedFieldDistance
```

Distance in pixels for generation of Signed Distance Field (SDF) font glyphs. This parameter must be greater or equal to zero. If the value is not zero, then the font is considered SDF.

The documentation for this struct was generated from the following file:

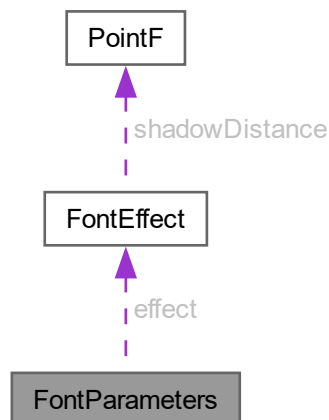
- [Afterwarp.Structs.h](#)

6.14 FontParameters Struct Reference

Parameters that define the appearance and important characteristics of the font.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for FontParameters:



Public Member Functions

- [FontSettings](#) ()=default
Creates font settings uninitialized.
- [FontParameters](#) (char const *family, float size, [FontWeight](#) weight=[FontWeight::Normal](#), [FontStretch](#) stretch=[FontStretch::Normal](#), [FontSlant](#) slant=[FontSlant::None](#))
Creates font settings with the given parameters.

Public Attributes

- char [family](#) [128]
Font family (e.g. "Helvetica").
- float [size](#)
Size of the font.
- [FontWeight](#) [weight](#)
Font letter weight.
- [FontStretch](#) [stretch](#)
Font letter stretch.
- [FontSlant](#) [slant](#)
Font letter slant.
- [FontAttributes](#) [attributes](#)
Additional font attributes.
- [FontEffect](#) [effect](#)
Effect parameters that contribute to font rasterization.

6.14.1 Detailed Description

Parameters that define the appearance and important characteristics of the font.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 FontParameters()

```
FontParameters (
    char const * family,
    float size,
    FontWeight weight = FontWeight::Normal,
    FontStretch stretch = FontStretch::Normal,
    FontSlant slant = FontSlant::None ) [inline]
```

Creates font settings with the given parameters.

6.14.3 Member Function Documentation

6.14.3.1 FontSettings()

```
FontSettings ( ) [default]
```

Creates font settings uninitialized.

6.14.4 Member Data Documentation

6.14.4.1 attributes

`FontAttributes` attributes

Additional font attributes.

6.14.4.2 effect

`FontEffect` effect

Effect parameters that contribute to font rasterization.

6.14.4.3 family

`char family[128]`

Font family (e.g. "Helvetica").

6.14.4.4 size

`float size`

Size of the font.

6.14.4.5 slant

`FontSlant` slant

Font letter slant.

6.14.4.6 stretch

`FontStretch` stretch

Font letter stretch.

6.14.4.7 weight

`FontWeight` weight

Font letter weight.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.15 ImageRegion Struct Reference

An image region defined by its texture number and bounding rectangle on that texture.

```
#include <Afterwarp.Structs.h>
```

Public Types

- enum { [Flip](#) = 0x01 , [Mirror](#) = 0x02 , [Rotate](#) = 0x04 }
Modifier flags that can be combined together.

Public Attributes

- uint16_t [left](#)
Left position of the region.
- uint16_t [top](#)
Top position of the region.
- uint16_t [width](#)
Width of the region.
- uint16_t [height](#)
Height of the region.
- uint16_t [index](#)
Texture index associated with this region.

6.15.1 Detailed Description

An image region defined by its texture number and bounding rectangle on that texture.

6.15.2 Member Enumeration Documentation

6.15.2.1 anonymous enum

anonymous enum

Modifier flags that can be combined together.

Enumerator

Flip	Flip region vertically.
Mirror	Mirror region horizontally.
Rotate	Rotate region by 90 degrees clockwise.

6.15.3 Member Data Documentation

6.15.3.1 height

`uint16_t height`

Height of the region.

6.15.3.2 index

`uint16_t index`

Texture index associated with this region.

6.15.3.3 left

`uint16_t left`

Left position of the region.

6.15.3.4 top

`uint16_t top`

Top position of the region.

6.15.3.5 width

`uint16_t width`

Width of the region.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.16 Margins Struct Reference

[Margins](#) defined by top, left, right and bottom edge paddings.

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float [left](#)
Left edge padding.
- float [top](#)
Top edge padding.
- float [right](#)
Right edge padding.
- float [bottom](#)
Bottom edge padding.

6.16.1 Detailed Description

[Margins](#) defined by top, left, right and bottom edge paddings.

6.16.2 Member Data Documentation

6.16.2.1 [bottom](#)

```
float bottom
```

Bottom edge padding.

6.16.2.2 [left](#)

```
float left
```

Left edge padding.

6.16.2.3 [right](#)

```
float right
```

Right edge padding.

6.16.2.4 [top](#)

```
float top
```

Top edge padding.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.17 Matrix Struct Reference

4x4 matrix representing transformation information in context of 3D graphics.

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float [data](#) [4][4]
Individual matrix values.

6.17.1 Detailed Description

4x4 matrix representing transformation information in context of 3D graphics.

6.17.2 Member Data Documentation

6.17.2.1 data

```
float data[4][4]
```

Individual matrix values.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.18 Matrix3x2 Struct Reference

3x2 matrix representing rotation and translation in context of 2D graphics.

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float [data](#) [3][2]
Individual matrix values.

6.18.1 Detailed Description

3x2 matrix representing rotation and translation in context of 2D graphics.

6.18.2 Member Data Documentation

6.18.2.1 data

```
float data[3][2]
```

Individual matrix values.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.19 MeshAligns Struct Reference

Alignment for all three axes that determine the placement of mesh around its model.

```
#include <Afterwarp.Structs.h>
```

Public Member Functions

- [MeshAligns](#) ()=default
Creates mesh alignments uninitialized.
- [MeshAligns](#) ([MeshAlign](#) x=[MeshAlign::Origin](#), [MeshAlign](#) y=[MeshAlign::Positive](#), [MeshAlign](#) z=[MeshAlign::↔](#)
[Origin](#), float [bias](#)=0.1f)
Creates new alignments with the given values.

Public Attributes

- [MeshAlign](#) x
Alignment for X axis.
- [MeshAlign](#) y
Alignment for Y axis.
- [MeshAlign](#) z
Alignment for Z axis.
- float [bias](#)
Bias value used for volume size adjustment.

6.19.1 Detailed Description

Alignment for all three axes that determine the placement of mesh around its model.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 MeshAligns() [1/2]

```
MeshAligns ( ) [default]
```

Creates mesh alignments uninitialized.

6.19.2.2 MeshAligns() [2/2]

```
MeshAligns (
    MeshAlign x = MeshAlign::Origin,
    MeshAlign y = MeshAlign::Positive,
    MeshAlign z = MeshAlign::Origin,
    float bias = 0.1f ) [inline]
```

Creates new alignments with the given values.

6.19.3 Member Data Documentation

6.19.3.1 bias

```
float bias
```

Bias value used for volume size adjustment.

6.19.3.2 x

```
MeshAlign x
```

Alignment for X axis.

6.19.3.3 y

```
MeshAlign y
```

Alignment for Y axis.

6.19.3.4 z

```
MeshAlign z
```

Alignment for Z axis.

The documentation for this struct was generated from the following file:

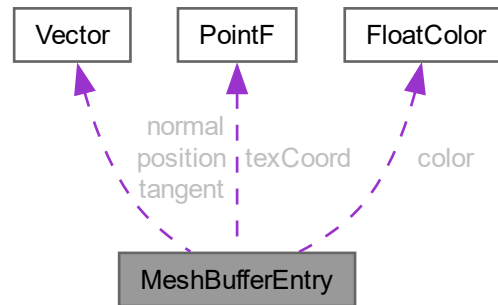
- [Afterwarp.Structs.h](#)

6.20 MeshBufferEntry Struct Reference

Calculated values for a single mesh vertex that can be modified by callback.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for MeshBufferEntry:



Public Attributes

- [Vector position](#)
Vertex 3D position.
- [Vector normal](#)
Vertex 3D normal (normalized).
- [Vector tangent](#)
Vertex 3D tangent (normalized).
- [PointF texCoord](#)
Vertex 2D texture coordinates.
- [FloatColor color](#)
Vertex RGBA color.

6.20.1 Detailed Description

Calculated values for a single mesh vertex that can be modified by callback.

6.20.2 Member Data Documentation

6.20.2.1 color

[FloatColor](#) color

Vertex RGBA color.

6.20.2.2 normal

`Vector` normal

Vertex 3D normal (normalized).

6.20.2.3 position

`Vector` position

Vertex 3D position.

6.20.2.4 tangent

`Vector` tangent

Vertex 3D tangent (normalized).

6.20.2.5 texCoord

`PointF` texCoord

Vertex 2D texture coordinates.

The documentation for this struct was generated from the following file:

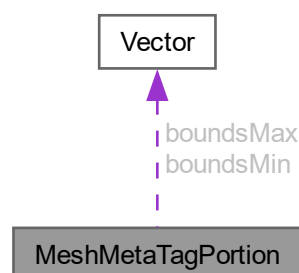
- [Afterwarp.Structs.h](#)

6.21 MeshMetaTagPortion Struct Reference

A portion of the mesh corresponding to a particular tag.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for MeshMetaTagPortion:



Public Attributes

- `int32_t firstIndex`
Starting index of the tag.
- `int32_t indexCount`
Number of indices of the tag.
- `int32_t firstVertex`
Starting vertex of the tag.
- `int32_t vertexCount`
Number of vertices of the tag.
- `Vector boundsMin`
Tag's minimum boundaries.
- `Vector boundsMax`
Tag's maximum boundaries.

6.21.1 Detailed Description

A portion of the mesh corresponding to a particular tag.

6.21.2 Member Data Documentation

6.21.2.1 boundsMax

`Vector boundsMax`

Tag's maximum boundaries.

6.21.2.2 boundsMin

`Vector boundsMin`

Tag's minimum boundaries.

6.21.2.3 firstIndex

`int32_t firstIndex`

Starting index of the tag.

6.21.2.4 firstVertex

`int32_t firstVertex`

Starting vertex of the tag.

6.21.2.5 indexCount

```
int32_t indexCount
```

Number of indices of the tag.

6.21.2.6 vertexCount

```
int32_t vertexCount
```

Number of vertices of the tag.

The documentation for this struct was generated from the following file:

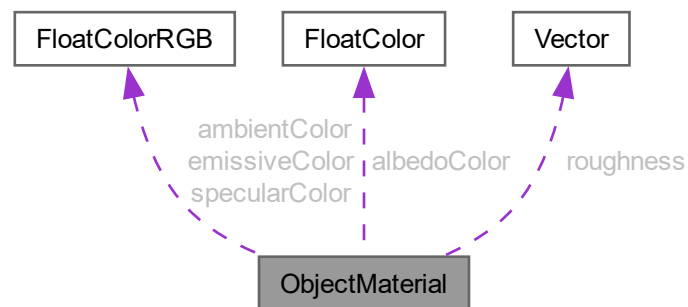
- [Afterwarp.Structs.h](#)

6.22 ObjectMaterial Struct Reference

Material that describes a particular object in 3D world.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for ObjectMaterial:



Public Attributes

- [TechniqueLighting technique](#)
Technique that describes how light is reflected off the object's surface.
- [FloatColorRGB ambientColor](#)
Ambient color that is always received by the object regardless the position of light.
- float [shadows](#)
Strength of received shadows. A value of zero means no shadows will be received.
- [FloatColor albedoColor](#)
Albedo color, where alpha-channel is typically a Bloom Factor (glare from strong reflections).
- [FloatColorRGB specularColor](#)
Specular color, which is added depending on reflected vector from material's surface.
- float [specularExponent](#)
Specular Exponent that defines how short or wide specular reflection appears.
- [FloatColorRGB emissiveColor](#)
Emissive color, which is always added before any lights.
- float [occlusion](#)
Strength of ambient occlusion. A value of zero means no occlusion will be received.
- [Vector roughness](#)
Roughness of the object's surface that defines how light is reflected.
- uint32_t [bitmask](#)
- float [frostedGlass](#)
Strength of Frosted Glass effect when rendering order-independent transparency.
- [ObjectPayload payload](#)
Custom payload value.

6.22.1 Detailed Description

Material that describes a particular object in 3D world.

6.22.2 Member Data Documentation

6.22.2.1 albedoColor

`FloatColor albedoColor`

Albedo color, where alpha-channel is typically a Bloom Factor (glare from strong reflections).

6.22.2.2 ambientColor

`FloatColorRGB ambientColor`

Ambient color that is always received by the object regardless the position of light.

6.22.2.3 bitmask

`uint32_t bitmask`

A bitmask that defines what lights affect this object. A value of zero means that this object is affected by all lights.

6.22.2.4 emissiveColor

`FloatColorRGB emissiveColor`

Emissive color, which is always added before any lights.

6.22.2.5 frostedGlass

`float frostedGlass`

Strength of Frosted Glass effect when rendering order-independent transparency.

6.22.2.6 occlusion

`float occlusion`

Strength of ambient occlusion. A value of zero means no occlusion will be received.

6.22.2.7 payload

`ObjectPayload payload`

Custom payload value.

6.22.2.8 roughness

`Vector roughness`

Roughness of the object's surface that defines how light is reflected.

6.22.2.9 shadows

`float shadows`

Strength of received shadows. A value of zero means no shadows will be received.

6.22.2.10 specularColor

`FloatColorRGB specularColor`

Specular color, which is added depending on reflected vector from material's surface.

6.22.2.11 specularExponent

`float specularExponent`

Specular Exponent that defines how short or wide specular reflection appears.

6.22.2.12 technique

`TechniqueLighting` technique

Technique that describes how light is reflected off the object's surface.

The documentation for this struct was generated from the following file:

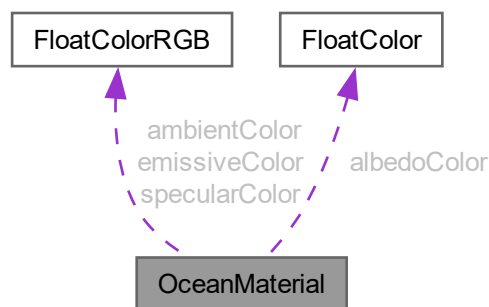
- [Afterwarp.Structs.h](#)

6.23 OceanMaterial Struct Reference

Material that describes ocean water surface in a 3D simulation.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for OceanMaterial:



Public Attributes

- [FloatColorRGB](#) `ambientColor`
Ambient color.
- float `fresnel`
Fresnel Coefficient (reflectance at normal incidence).
- [FloatColor](#) `albedoColor`
Ambient color that is always received by the ocean regardless the position of light.
- [FloatColorRGB](#) `specularColor`
Specular color, which is added depending on reflected vector from ocean's surface.
- float `specularExponent`
Specular color, which is added depending on reflected vector from ocean's surface.
- [FloatColorRGB](#) `emissiveColor`
Emissive color, which is always added before any lights.
- float `extinction`
Extinction of the ocean's surface, which determines its transparency.
- float `shadows`
- uint32_t `bitmask`

6.23.1 Detailed Description

Material that describes ocean water surface in a 3D simulation.

6.23.2 Member Data Documentation

6.23.2.1 albedoColor

`FloatColor` albedoColor

Ambient color that is always received by the ocean regardless the position of light.

6.23.2.2 ambientColor

`FloatColorRGB` ambientColor

Ambient color.

6.23.2.3 bitmask

`uint32_t` bitmask

A bitmask that defines what lights affect this object. A value of zero means that this object is affected by all lights.

6.23.2.4 emissiveColor

`FloatColorRGB` emissiveColor

Emissive color, which is always added before any lights.

6.23.2.5 extinction

`float` extinction

Extinction of the ocean's surface, which determines its transparency.

6.23.2.6 fresnel

`float` fresnel

Fresnel Coefficient (reflectance at normal incidence).

6.23.2.7 shadows

```
float shadows
```

Strength of a shadows when projected to the object's surface. A value of zero means shadows are not rendered and/or ignored.

6.23.2.8 specularColor

```
FloatColorRGB specularColor
```

Specular color, which is added depending on reflected vector from ocean's surface.

6.23.2.9 specularExponent

```
float specularExponent
```

Specular color, which is added depending on reflected vector from ocean's surface.

The documentation for this struct was generated from the following file:

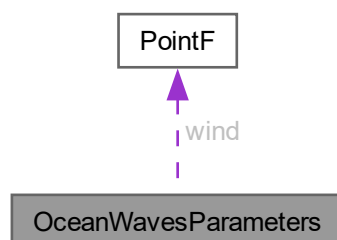
- [Afterwarp.Structs.h](#)

6.24 OceanWavesParameters Struct Reference

Parameters that define the appearance of waves simulation.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for OceanWavesParameters:



Public Attributes

- `int32_t` [resolution](#)
Resolution of the simulation textures.
- `PointF` [wind](#)
General direction in which the waves propagate.
- `float` [waveSize](#)
Size of the generated waves.
- `float` [choppiness](#)
Choppiness of the generated waves.

6.24.1 Detailed Description

Parameters that define the appearance of waves simulation.

6.24.2 Member Data Documentation

6.24.2.1 `choppiness`

```
float choppiness
```

Choppiness of the generated waves.

6.24.2.2 `resolution`

```
int32_t resolution
```

Resolution of the simulation textures.

6.24.2.3 `waveSize`

```
float waveSize
```

Size of the generated waves.

6.24.2.4 `wind`

```
PointF wind
```

General direction in which the waves propagate.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.25 ParallaxMappingParameters Struct Reference

Parameters that define how parallax mapping is performed.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- float [scale](#)
Scale that defines how height value from parallax map translates into 3D depth.
- int32_t [samplesMin](#)
Minimum number of samples to read per pixel.
- int32_t [samplesMax](#)
Maximum number of samples to read per pixel.
- float [occlusion](#)
Occlusion strength applied from Occlusion Map (alpha-channel of Normal Map).

6.25.1 Detailed Description

Parameters that define how parallax mapping is performed.

6.25.2 Member Data Documentation

6.25.2.1 occlusion

```
float occlusion
```

Occlusion strength applied from Occlusion Map (alpha-channel of Normal Map).

6.25.2.2 samplesMax

```
int32_t samplesMax
```

Maximum number of samples to read per pixel.

6.25.2.3 samplesMin

```
int32_t samplesMin
```

Minimum number of samples to read per pixel.

6.25.2.4 scale

float scale

Scale that defines how height value from parallax map translates into 3D depth.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.26 PathElement Union Reference

A single element in path declaration.

```
#include <Afterwarp.Structs.h>
```

Public Member Functions

- [PathElement](#) ()=default
Creates a new path element uninitialized.
- [PathElement](#) (uint8_t [command](#), uint16_t [length](#)=0u)
Creates a new path element with the given command and length.
- [PathElement](#) (float [value](#))
Creates a new path element with the given data value.

Public Attributes

- struct {
 - uint8_t [command](#)
Path command enumeration combined with one of its attribute bits.
 - uint8_t [reserved](#)
Field reserved for future use.
 - uint16_t [length](#)
Number of "value" elements that follows this declaration.
- [tag](#)
Tag that defines a combination of command and number of values following such tag.
- float [value](#)
A single 32-bit floating-point value of the path coordinates.

6.26.1 Detailed Description

A single element in path declaration.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 PathElement() [1/3]

```
PathElement ( ) [default]
```

Creates a new path element uninitialized.

6.26.2.2 PathElement() [2/3]

```
PathElement (
    uint8_t command,
    uint16_t length = 0u ) [inline]
```

Creates a new path element with the given command and length.

6.26.2.3 PathElement() [3/3]

```
PathElement (
    float value ) [inline]
```

Creates a new path element with the given data value.

6.26.3 Member Data Documentation

6.26.3.1 command

```
uint8_t command
```

Path command enumeration combined with one of its attribute bits.

6.26.3.2 length

```
uint16_t length
```

Number of "value" elements that follows this declaration.

6.26.3.3 reserved

```
uint8_t reserved
```

Field reserved for future use.

6.26.3.4 [struct]

```
struct { ... } tag
```

Tag that defines a combination of command and number of values following such tag.

6.26.3.5 value

```
float value
```

A single 32-bit floating-point value of the path coordinates.

The documentation for this union was generated from the following file:

- [Afterwarp.Structs.h](#)

6.27 Point Struct Reference

2D integer point (or vector).

```
#include <Afterwarp.Types.h>
```

Public Attributes

- [int32_t x](#)
Horizontal coordinate in 2D space.
- [int32_t y](#)
Vertical coordinate in 2D space.

6.27.1 Detailed Description

2D integer point (or vector).

6.27.2 Member Data Documentation

6.27.2.1 x

```
int32_t x
```

Horizontal coordinate in 2D space.

6.27.2.2 y

`int32_t y`

Vertical coordinate in 2D space.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.28 PointF Struct Reference

2D floating-point vector.

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float [x](#)
Horizontal coordinate in 2D space.
- float [y](#)
Vertical coordinate in 2D space.

6.28.1 Detailed Description

2D floating-point vector.

6.28.2 Member Data Documentation

6.28.2.1 x

`float x`

Horizontal coordinate in 2D space.

6.28.2.2 y

`float y`

Vertical coordinate in 2D space.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.29 ProgramElement Struct Reference

Structure that describes a single physical element of shader program.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- char [name](#) [64]
Name of the element (e.g. variable, buffer or texture name, null-terminated).
- [ShaderElement](#) [element](#)
Type of shader element.
- [ShaderType](#) [shader](#)
Shader to which this element belongs to.
- uint32_t [channel](#)
A virtual channel index, which will be associated with the appropriate buffer attached to that channel.
- int32_t [index](#)
- uint32_t [size](#)
Size of element in bytes.

6.29.1 Detailed Description

Structure that describes a single physical element of shader program.

6.29.2 Member Data Documentation

6.29.2.1 channel

```
uint32_t channel
```

A virtual channel index, which will be associated with the appropriate buffer attached to that channel.

6.29.2.2 element

```
ShaderElement element
```

Type of shader element.

6.29.2.3 index

```
int32_t index
```

Index of physical slot to which the element should be bound to. This can be either an actual binding point or a register number.

6.29.2.4 name

```
char name[64]
```

Name of the element (e.g. variable, buffer or texture name, null-terminated).

6.29.2.5 shader

```
ShaderType shader
```

Shader to which this element belongs to.

6.29.2.6 size

```
uint32_t size
```

Size of element in bytes.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.30 ProgramVariable Struct Reference

Structure that describes a single (uniform) variable in a shader program.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- char [name](#) [64]
Name of the element (e.g. variable, buffer or texture name, null-terminated).
- int32_t [index](#)
Index that uniquely identifies this variable.
- [ShaderType](#) [shader](#)
Shader to which this element belongs to.
- [ElementFormat](#) [format](#)
Data format of each value in this variable.
- int32_t [count](#)
- uint32_t [size](#)
Size of variable in bytes.
- uint32_t [offset](#)
Offset in a buffer where variable starts.

6.30.1 Detailed Description

Structure that describes a single (uniform) variable in a shader program.

6.30.2 Member Data Documentation

6.30.2.1 count

`int32_t count`

Number of values used by this variable. If "format" is "Undefined", this indicates number of bytes the element occupies.

6.30.2.2 format

`ElementFormat format`

Data format of each value in this variable.

6.30.2.3 index

`int32_t index`

Index that uniquely identifies this variable.

6.30.2.4 name

`char name[64]`

Name of the element (e.g. variable, buffer or texture name, null-terminated).

6.30.2.5 offset

`uint32_t offset`

Offset in a buffer where variable starts.

6.30.2.6 shader

`ShaderType shader`

Shader to which this element belongs to.

6.30.2.7 size

`uint32_t size`

Size of variable in bytes.

The documentation for this struct was generated from the following file:

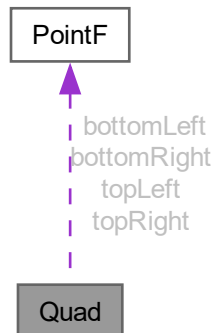
- [Afterwarp.Structs.h](#)

6.31 Quad Struct Reference

Floating-point quadrilateral defined by four vertices in clockwise order starting from top/left.

```
#include <Afterwarp.Types.h>
```

Collaboration diagram for Quad:



Public Attributes

- [PointF topLeft](#)
Top/left vertex position.
- [PointF topRight](#)
Top/right vertex position.
- [PointF bottomRight](#)
Bottom/right vertex position.
- [PointF bottomLeft](#)
Bottom/left vertex position.

6.31.1 Detailed Description

Floating-point quadrilateral defined by four vertices in clockwise order starting from top/left.

6.31.2 Member Data Documentation

6.31.2.1 bottomLeft

```
PointF bottomLeft
```

Bottom/left vertex position.

6.31.2.2 bottomRight

`PointF` `bottomRight`

Bottom/right vertex position.

6.31.2.3 topLeft

`PointF` `topLeft`

Top/left vertex position.

6.31.2.4 topRight

`PointF` `topRight`

Top/right vertex position.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.32 Quaternion Struct Reference

3D quaternion.

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float `x`
Quaternion X coordinate.
- float `y`
Quaternion Y coordinate.
- float `z`
Quaternion Z coordinate.
- float `w`
Quaternion W coordinate.

6.32.1 Detailed Description

3D quaternion.

6.32.2 Member Data Documentation

6.32.2.1 w

`float w`

[Quaternion](#) W coordinate.

6.32.2.2 x

`float x`

[Quaternion](#) X coordinate.

6.32.2.3 y

`float y`

[Quaternion](#) Y coordinate.

6.32.2.4 z

`float z`

[Quaternion](#) Z coordinate.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.33 Rect Struct Reference

Rectangle defined by integer top and left margins, width and height.

```
#include <Afterwarp.Types.h>
```

Public Attributes

- `int32_t left`
Left position of the rectangle.
- `int32_t top`
Top position of the rectangle.
- `int32_t width`
Width of the rectangle.
- `int32_t height`
Height of the rectangle.

6.33.1 Detailed Description

Rectangle defined by integer top and left margins, width and height.

6.33.2 Member Data Documentation

6.33.2.1 height

```
int32_t height
```

Height of the rectangle.

6.33.2.2 left

```
int32_t left
```

Left position of the rectangle.

6.33.2.3 top

```
int32_t top
```

Top position of the rectangle.

6.33.2.4 width

```
int32_t width
```

Width of the rectangle.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.34 RectF Struct Reference

Rectangle defined by floating-point top and left margins, width and height.

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float [left](#)
Left position of the rectangle.
- float [top](#)
Top position of the rectangle.
- float [width](#)
Width of the rectangle.
- float [height](#)
Height of the rectangle.

6.34.1 Detailed Description

Rectangle defined by floating-point top and left margins, width and height.

6.34.2 Member Data Documentation

6.34.2.1 height

`float height`

Height of the rectangle.

6.34.2.2 left

`float left`

Left position of the rectangle.

6.34.2.3 top

`float top`

Top position of the rectangle.

6.34.2.4 width

`float width`

Width of the rectangle.

The documentation for this struct was generated from the following file:

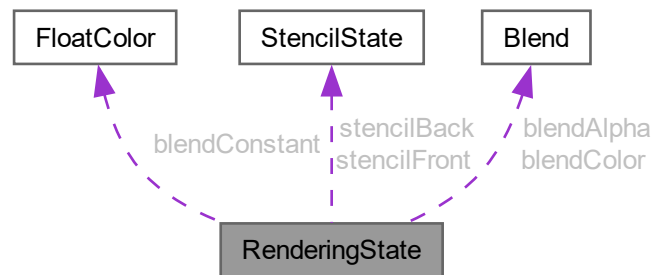
- [Afterwarp.Types.h](#)

6.35 RenderingState Struct Reference

Parameters that define `depth`, `stencil`, `rasterizer` and `blending` operations.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for `RenderingState`:



Public Attributes

- `uint32_t` [states](#)
Combination of state flags that define rendering operation.
- [TriangleFace](#) `cullFace`
How triangle faces should be culled.
- [ComparisonFunc](#) `depthFunc`
Depth buffer comparison function.
- `float` [depthBias](#)
Constant offset added to depth to alleviate z-fighting issues.
- `float` [slopeDepthBias](#)
Slope coefficient used for relative depth when calculating depth bias.
- `float` [clampDepthBias](#)
Value to clamp depth bias, preventing artifacts on very high slopes.
- `uint8_t` [stencilRefValue](#)
Reference value for the stencil test.
- `uint8_t` [stencilRefMask](#)
Mask that is applied to reference value and stored value after the test.
- `uint8_t` [stencilWriteMask](#)
Mask that is used to enable or disable writing of individual bits to stencil buffer.
- [FloatColor](#) `blendConstant`
Custom alpha-blending constant.
- `struct` [StencilState](#) `stencilFront`
Stencil function and operation that are performed for front-facing triangles.
- `struct` [StencilState](#) `stencilBack`
Stencil function and operation that are performed for back-facing triangles.
- `struct` [Blend](#) `blendColor`
Blending parameters that should be applied to color portion of components.
- `struct` [Blend](#) `blendAlpha`
Blending parameters that should be applied to alpha portion of components.

6.35.1 Detailed Description

Parameters that define `depth`, `stencil`, `rasterizer` and `blending` operations.

6.35.2 Member Data Documentation

6.35.2.1 `blendAlpha`

```
struct Blend blendAlpha
```

Blending parameters that should be applied to alpha portion of components.

6.35.2.2 `blendColor`

```
struct Blend blendColor
```

Blending parameters that should be applied to color portion of components.

6.35.2.3 `blendConstant`

```
FloatColor blendConstant
```

Custom alpha-blending constant.

6.35.2.4 `clampDepthBias`

```
float clampDepthBias
```

Value to clamp depth bias, preventing artifacts on very high slopes.

6.35.2.5 `cullFace`

```
TriangleFace cullFace
```

How triangle faces should be culled.

6.35.2.6 `depthBias`

```
float depthBias
```

Constant offset added to depth to alleviate z-fighting issues.

6.35.2.7 depthFunc

`ComparisonFunc` depthFunc

Depth buffer comparison function.

6.35.2.8 slopeDepthBias

`float` slopeDepthBias

Slope coefficient used for relative depth when calculating depth bias.

6.35.2.9 states

`uint32_t` states

Combination of state flags that define rendering operation.

6.35.2.10 stencilBack

`struct StencilState` stencilBack

Stencil function and operation that are performed for back-facing triangles.

6.35.2.11 stencilFront

`struct StencilState` stencilFront

Stencil function and operation that are performed for front-facing triangles.

6.35.2.12 stencilRefMask

`uint8_t` stencilRefMask

Mask that is applied to reference value and stored value after the test.

6.35.2.13 stencilRefValue

`uint8_t` stencilRefValue

Reference value for the stencil test.

6.35.2.14 stencilWriteMask

```
uint8_t stencilWriteMask
```

Mask that is used to enable or disable writing of individual bits to stencil buffer.

The documentation for this struct was generated from the following file:

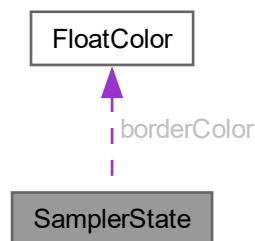
- [Afterwarp.Structs.h](#)

6.36 SamplerState Struct Reference

Sampler parameters that are associated with a particular texture unit.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for SamplerState:



Public Attributes

- [TextureFilter filterMin](#)
Minification texture filtering.
- [TextureFilter filterMag](#)
Magnification texture filtering.
- [TextureFilter filterMip](#)
Mipmapping texture filtering.
- [TextureAddress address](#) [3]
Texture coordinate addressing for S, T and R (U, V and W) coordinates.
- [FloatColor borderColor](#)
Special color constant to be used for TextureAddress::Border addressing mode.
- [int32_t anisotropy](#)
Total number of samples to be used with anisotropic filtering.
- [float minLOD](#)
Minimum level of detail for texture mipmap range.
- [float maxLOD](#)
Maximum level of detail for texture mipmap range.
- [float biasLOD](#)
Offset from the calculated mipmap level.
- [ComparisonFunc compareFunc](#)
Comparison function for texture values.
- [uint8_t compareToRef](#)
Indicates whether comparison should be made against a particular reference value or simply fetched.

6.36.1 Detailed Description

Sampler parameters that are associated with a particular texture unit.

6.36.2 Member Data Documentation

6.36.2.1 address

```
TextureAddress address[3]
```

Texture coordinate addressing for S, T and R (U, V and W) coordinates.

6.36.2.2 anisotropy

```
int32_t anisotropy
```

Total number of samples to be used with anisotropic filtering.

6.36.2.3 biasLOD

```
float biasLOD
```

Offset from the calculated mipmap level.

6.36.2.4 borderColor

```
FloatColor borderColor
```

Special color constant to be used for TextureAddress::Border addressing mode.

6.36.2.5 compareFunc

```
ComparisonFunc compareFunc
```

Comparison function for texture values.

6.36.2.6 compareToRef

```
uint8_t compareToRef
```

Indicates whether comparison should be made against a particular reference value or simply fetched.

6.36.2.7 filterMag

`TextureFilter filterMag`

Magnification texture filtering.

6.36.2.8 filterMin

`TextureFilter filterMin`

Minification texture filtering.

6.36.2.9 filterMip

`TextureFilter filterMip`

Mipmapping texture filtering.

6.36.2.10 maxLOD

`float maxLOD`

Maximum level of detail for texture mipmap range.

6.36.2.11 minLOD

`float minLOD`

Minimum level of detail for texture mipmap range.

The documentation for this struct was generated from the following file:

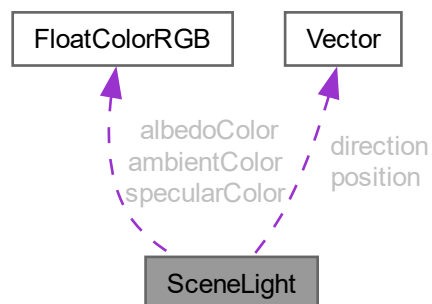
- [Afterwarp.Structs.h](#)

6.37 SceneLight Struct Reference

Light source parameters in 3D scene.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for SceneLight:



Public Attributes

- [FloatColorRGB ambientColor](#)
Ambient color.
- float [attenuationStart](#)
Range at which light attenuation starts (FLT_MAX means no attenuation should be applied).
- [FloatColorRGB albedoColor](#)
Albedo color.
- float [attenuationEnd](#)
Range at which light attenuation ends. This must be equal to or higher than `attenuationStart`.
- [FloatColorRGB specularColor](#)
Specular color.
- float [specularExponent](#)
Exponent for specular light reflections.
- [Vector position](#)
Position of the light source.
- float [intensity](#)
Intensity of the light.
- [Vector direction](#)
Direction of the light source (non-zero value indicates a spotlight, zero values means omni-light).
- float [angle](#)
Angle at which spotlight starts to fade in range of $[0, \pi]$.
- float [angleCutoff](#)
Angle after which spotlight is completely dark. This should be equal to or higher than `angle`.
- int32_t [shadowCaster](#)
- uint32_t [bitmask](#)
- uint32_t [reserved](#)
Reserved, must be set to zero.
- [ObjectPayload payload](#)
Custom payload value.

6.37.1 Detailed Description

Light source parameters in 3D scene.

6.37.2 Member Data Documentation

6.37.2.1 albedoColor

[FloatColorRGB](#) `albedoColor`

Albedo color.

6.37.2.2 ambientColor

[FloatColorRGB](#) `ambientColor`

Ambient color.

6.37.2.3 angle

```
float angle
```

Angle at which spotlight starts to fade in range of [0, PI).

6.37.2.4 angleCutoff

```
float angleCutoff
```

Angle after which spotlight is completely dark. This should be equal to or higher than `angle`.

6.37.2.5 attenuationEnd

```
float attenuationEnd
```

Range at which light attenuation ends. This must be equal to or higher than `attenuationStart`.

6.37.2.6 attenuationStart

```
float attenuationStart
```

Range at which light attenuation starts (FLT_MAX means no attenuation should be applied).

6.37.2.7 bitmask

```
uint32_t bitmask
```

Bitmask that defines what objects are affected by this light. A value of zero means that light affects all objects.

6.37.2.8 direction

```
Vector direction
```

Direction of the light source (non-zero value indicates a spotlight, zero values means omni-light).

6.37.2.9 intensity

```
float intensity
```

Intensity of the light.

6.37.2.10 payload

```
ObjectPayload payload
```

Custom payload value.

6.37.2.11 position

`Vector` position

Position of the light source.

6.37.2.12 reserved

`uint32_t` reserved

Reserved, must be set to zero.

6.37.2.13 shadowCaster

`int32_t` shadowCaster

Index of the shadow caster associated with the light source. A value of -1 indicates light source does not cast shadows.

6.37.2.14 specularColor

`FloatColorRGB` specularColor

Specular color.

6.37.2.15 specularExponent

`float` specularExponent

Exponent for specular light reflections.

The documentation for this struct was generated from the following file:

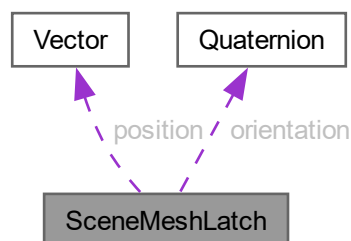
- [Afterwarp.Structs.h](#)

6.38 SceneMeshLatch Struct Reference

A latch in 3D scene mesh, which describes a bone joint connection or a movement waypoint.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for SceneMeshLatch:



Public Attributes

- char `name` [256]
Name of the latch.
- `Vector` `position`
Position of the latch.
- uint16_t `type`
Type of the latch.
- uint16_t `group`
Group of the latch.
- `Quaternion` `orientation`
Orientation of the latch.

6.38.1 Detailed Description

A latch in 3D scene mesh, which describes a bone joint connection or a movement waypoint.

6.38.2 Member Data Documentation

6.38.2.1 `group`

```
uint16_t group
```

Group of the latch.

6.38.2.2 `name`

```
char name[256]
```

Name of the latch.

6.38.2.3 `orientation`

```
Quaternion orientation
```

Orientation of the latch.

6.38.2.4 `position`

```
Vector position
```

Position of the latch.

6.38.2.5 type

uint16_t type

Type of the latch.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.39 SceneMeshMaterialRange Struct Reference

A range of indices in the mesh that a material applies to.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- int32_t [indexStart](#)
Starting index.
- int32_t [indexCount](#)
Number of indices.

6.39.1 Detailed Description

A range of indices in the mesh that a material applies to.

6.39.2 Member Data Documentation

6.39.2.1 indexCount

int32_t indexCount

Number of indices.

6.39.2.2 indexStart

int32_t indexStart

Starting index.

The documentation for this struct was generated from the following file:

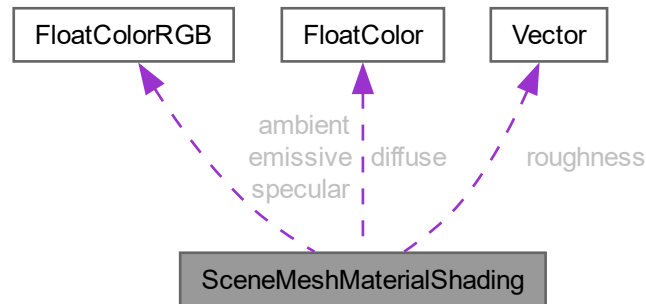
- [Afterwarp.Structs.h](#)

6.40 SceneMeshMaterialShading Struct Reference

Shading parameters for scene mesh material.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for SceneMeshMaterialShading:



Public Attributes

- [FloatColorRGB ambient](#)
Ambient color.
- float [bloom](#)
Bloom (glare) factor.
- [FloatColor diffuse](#)
Diffuse color (alpha-channel denotes transparency).
- [FloatColorRGB specular](#)
Specular color.
- float [specularExponent](#)
Specular exponent.
- [FloatColorRGB emissive](#)
Emissive color.
- int32_t [lighting](#)
Lighting model (0 - Phong, 1 - Minnaert, 2 - Cook-Torrance, 3 - Oren-Nayar)
- [Vector roughness](#)
Roughness values for non-Phong lighting models.

6.40.1 Detailed Description

Shading parameters for scene mesh material.

6.40.2 Member Data Documentation

6.40.2.1 ambient

`FloatColorRGB` ambient

Ambient color.

6.40.2.2 bloom

`float` bloom

Bloom (glare) factor.

6.40.2.3 diffuse

`FloatColor` diffuse

Diffuse color (alpha-channel denotes transparency).

6.40.2.4 emissive

`FloatColorRGB` emissive

Emissive color.

6.40.2.5 lighting

`int32_t` lighting

Lighting model (0 - Phong, 1 - Minnaert, 2 - Cook-Torrance, 3 - Oren-Nayar)

6.40.2.6 roughness

`Vector` roughness

Roughness values for non-Phong lighting models.

6.40.2.7 specular

`FloatColorRGB` specular

Specular color.

6.40.2.8 specularExponent

```
float specularExponent
```

Specular exponent.

The documentation for this struct was generated from the following file:

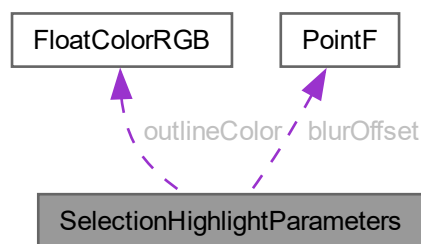
- [Afterwarp.Structs.h](#)

6.41 SelectionHighlightParameters Struct Reference

Parameters that define how selection highlight technique is performed.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for SelectionHighlightParameters:



Public Attributes

- [FloatColorRGB](#) **outlineColor**
Color of the selection glow's outline.
- float **outlineStart**
At which position outline begins (values of 1 or higher mean no outline).
- float **glowIntensity**
How intense should the glow appear.
- int32_t **blurPasses**
Number of passes for the blur effect.
- [PointF](#) **blurOffset**
Offset for the blur effect.

6.41.1 Detailed Description

Parameters that define how selection highlight technique is performed.

6.41.2 Member Data Documentation

6.41.2.1 blurOffset

`PointF blurOffset`

Offset for the blur effect.

6.41.2.2 blurPasses

`int32_t blurPasses`

Number of passes for the blur effect.

6.41.2.3 glowIntensity

`float glowIntensity`

How intense should the glow appear.

6.41.2.4 outlineColor

`FloatColorRGB outlineColor`

Color of the selection glow's outline.

6.41.2.5 outlineStart

`float outlineStart`

At which position outline begins (values of 1 or higher mean no outline).

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.42 ShadowParameters Struct Reference

Parameters that define how shadows are rendered.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- float [exponent1](#)
Primary exponent for ESM and EVSM techniques.
- float [exponent2](#)
Secondary exponent for ESM and EVSM techniques.
- float [variance](#)
Maximum variance delimiter used with VSM and EVSM techniques.
- float [bleedSigma](#)
Light bleeding sigma for reducing light bleeding artifacts with VSM and EVSM techniques.
- float [bias](#)
Bias parameter that is applied to depth derivatives in VSM technique.
- float [blurSigma](#)
Sigma value for shadow map's gaussian blur.
- int32_t [blurSamples](#)
Number of samples for shadow map's gaussian blur.

6.42.1 Detailed Description

Parameters that define how shadows are rendered.

6.42.2 Member Data Documentation

6.42.2.1 bias

```
float bias
```

Bias parameter that is applied to depth derivatives in VSM technique.

6.42.2.2 bleedSigma

```
float bleedSigma
```

Light bleeding sigma for reducing light bleeding artifacts with VSM and EVSM techniques.

6.42.2.3 blurSamples

```
int32_t blurSamples
```

Number of samples for shadow map's gaussian blur.

6.42.2.4 blurSigma

```
float blurSigma
```

Sigma value for shadow map's gaussian blur.

6.42.2.5 exponent1

```
float exponent1
```

Primary exponent for ESM and EVSM techniques.

6.42.2.6 exponent2

```
float exponent2
```

Secondary exponent for ESM and EVSM techniques.

6.42.2.7 variance

```
float variance
```

Maximum variance delimiter used with VSM and EVSM techniques.

The documentation for this struct was generated from the following file:

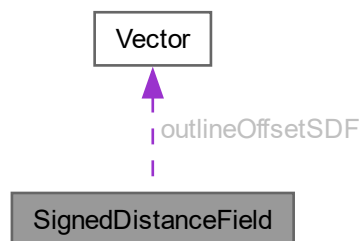
- [Afterwarp.Structs.h](#)

6.43 SignedDistanceField Struct Reference

Signed Distance Field (SDF) parameters.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for SignedDistanceField:



Public Attributes

- [SuperSampleSDF](#) [superSampleSDF](#)
Type of super-sampling for the field's rendering.
- float [signedFieldDistance](#)
Distance that was used for generating the field.
- [Vector](#) [outlineOffsetSDF](#)
Color offset in YCH color space for rendering the shape's outline.
- float [outlineDistanceMinSDF](#)
Minimum field distance for the shape's outline.
- float [outlineDistanceMaxSDF](#)
Maximum field distance for the shape's outline.

6.43.1 Detailed Description

Signed Distance Field (SDF) parameters.

6.43.2 Member Data Documentation

6.43.2.1 [outlineDistanceMaxSDF](#)

```
float outlineDistanceMaxSDF
```

Maximum field distance for the shape's outline.

6.43.2.2 [outlineDistanceMinSDF](#)

```
float outlineDistanceMinSDF
```

Minimum field distance for the shape's outline.

6.43.2.3 [outlineOffsetSDF](#)

```
Vector outlineOffsetSDF
```

Color offset in YCH color space for rendering the shape's outline.

6.43.2.4 [signedFieldDistance](#)

```
float signedFieldDistance
```

Distance that was used for generating the field.

6.43.2.5 superSampleSDF

`SuperSampleSDF` `superSampleSDF`

Type of super-sampling for the field's rendering.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.44 StencilState Struct Reference

Stencil state that is independent for each front and back facing.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- [ComparisonFunc](#) `func`
Stencil test function.
- [StencilOp](#) `failOp`
Action to take when stencil test fails.
- [StencilOp](#) `depthFailOp`
Action to take when stencil test passes but depth test fails.
- [StencilOp](#) `depthPassOp`

6.44.1 Detailed Description

Stencil state that is independent for each front and back facing.

6.44.2 Member Data Documentation

6.44.2.1 depthFailOp

[StencilOp](#) `depthFailOp`

Action to take when stencil test passes but depth test fails.

6.44.2.2 depthPassOp

[StencilOp](#) `depthPassOp`

Action to take when both stencil and depth tests pass, or when stencil test passes and either there is no depth buffer or depth testing is disabled.

6.44.2.3 failOp

`StencilOp failOp`

Action to take when stencil test fails.

6.44.2.4 func

`ComparisonFunc func`

Stencil test function.

The documentation for this struct was generated from the following file:

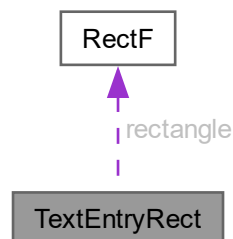
- [Afterwarp.Structs.h](#)

6.45 TextEntryRect Struct Reference

Individual text entry that will appear as rendered.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for TextEntryRect:



Public Attributes

- `intptr_t position`
Starting character position inside the text (for Unicode, first byte).
- `RectF rectangle`
Character's rectangle.

6.45.1 Detailed Description

Individual text entry that will appear as rendered.

6.45.2 Member Data Documentation

6.45.2.1 position

`intptr_t position`

Starting character position inside the text (for Unicode, first byte).

6.45.2.2 rectangle

`RectF rectangle`

Character's rectangle.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.46 TextRenderModifiers Struct Reference

Modifier attributes that can be applied to the rendered text.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- float [scale](#)
- float [interleave](#)
- float [verticalSpace](#)
Global spacing that will be added vertically when drawing multiple lines of text.
- [BlendingEffect](#) [effect](#)
Blending effect to use (if left undefined, normal blending effect will be used).

6.46.1 Detailed Description

Modifier attributes that can be applied to the rendered text.

6.46.2 Member Data Documentation

6.46.2.1 effect

`BlendingEffect effect`

Blending effect to use (if left undefined, normal blending effect will be used).

6.46.2.2 `interleave`

```
float interleave
```

Global spacing that will be added horizontally between text letters. This can be used to expand or shrink the text.

6.46.2.3 `scale`

```
float scale
```

Global font scale that is applied to the whole rendered text. A default value of zero means that font scale will not be modified. Using values other than zero or one would likely result in not pixel-perfect text rendering, appearing blurry. However, this can be used for real-time text animations.

6.46.2.4 `verticalSpace`

```
float verticalSpace
```

Global spacing that will be added vertically when drawing multiple lines of text.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.47 TextureParameters Struct Reference

Texture parameters and characteristics.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- `int32_t width`
Texture width (in pixels).
- `int32_t height`
Texture height (in pixels).
- `int32_t layers`
Number of texture layers in case of Texture Array or depth in case of 3D textures.
- `PixelFormat format`
Format of the texture's pixels.
- `TextureType type`
Texture type that defines how it is composed.
- `TextureAttributes attributes`
Attributes that define the characteristics of the texture.
- `int32_t multisamples`
Number of multisamples used in the render target.
- `PixelFormat depthStencil`
Depth/stencil format that will be used to create an associated buffer.

6.47.1 Detailed Description

Texture parameters and characteristics.

6.47.2 Member Data Documentation

6.47.2.1 attributes

`TextureAttributes` attributes

Attributes that define the characteristics of the texture.

6.47.2.2 depthStencil

`PixelFormat` depthStencil

Depth/stencil format that will be used to create an associated buffer.

6.47.2.3 format

`PixelFormat` format

Format of the texture's pixels.

6.47.2.4 height

`int32_t` height

Texture height (in pixels).

6.47.2.5 layers

`int32_t` layers

Number of texture layers in case of Texture Array or depth in case of 3D textures.

6.47.2.6 multisamples

`int32_t` multisamples

Number of multisamples used in the render target.

6.47.2.7 type

`TextureType` type

Texture type that defines how it is composed.

6.47.2.8 width

`int32_t` width

Texture width (in pixels).

The documentation for this struct was generated from the following file:

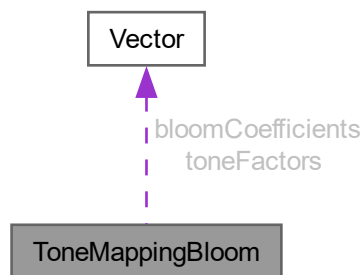
- [Afterwarp.Structs.h](#)

6.48 ToneMappingBloom Struct Reference

Parameters that define behavior of tone-mapping and bloom effects.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for ToneMappingBloom:



Public Attributes

- float [bloomThreshold](#)
Amount to be subtracted from initial HDR colors to determine bloom.
- float [bloomGamma](#)
Inverse gamma power to adjust the bloom color after threshold.
- [Vector](#) [bloomCoefficients](#)
Grayscale coefficients for red, green and blue components.
- float [bloomColorShift](#)
Amount of color to shift into grayscale.
- float [bloomBlurSigma](#)
Sigma value of Gaussian Blur for bloom pixels.
- int32_t [bloomBlurSamples](#)
Number of bloom pixels to sample during Gaussian blur.
- float [toneWhite](#)
Maximum allowed level of white for tone-mapping.
- [Vector](#) [toneFactors](#)
Conversion coefficients to calculate Luma from RGB during tone-mapping.
- float [frostedPower](#)
Power used to adjust the curve of frosted glass equation.
- float [glassyBuckets](#)
Number of buckets per pixel for OIT technique.

6.48.1 Detailed Description

Parameters that define behavior of tone-mapping and bloom effects.

6.48.2 Member Data Documentation

6.48.2.1 bloomBlurSamples

```
int32_t bloomBlurSamples
```

Number of bloom pixels to sample during Gaussian blur.

6.48.2.2 bloomBlurSigma

```
float bloomBlurSigma
```

Sigma value of Gaussian Blur for bloom pixels.

6.48.2.3 bloomCoefficients

```
Vector bloomCoefficients
```

Grayscale coefficients for red, green and blue components.

6.48.2.4 bloomColorShift

```
float bloomColorShift
```

Amount of color to shift into grayscale.

6.48.2.5 bloomGamma

```
float bloomGamma
```

Inverse gamma power to adjust the bloom color after threshold.

6.48.2.6 bloomThreshold

```
float bloomThreshold
```

Amount to be subtracted from initial HDR colors to determine bloom.

6.48.2.7 frostedPower

```
float frostedPower
```

Power used to adjust the curve of frosted glass equation.

6.48.2.8 glassyBuckets

```
float glassyBuckets
```

Number of buckets per pixel for OIT technique.

6.48.2.9 toneFactors

```
Vector toneFactors
```

Conversion coefficients to calculate Luma from RGB during tone-mapping.

6.48.2.10 toneWhite

```
float toneWhite
```

Maximum allowed level of white for tone-mapping.

The documentation for this struct was generated from the following file:

- [Afterwarp.Structs.h](#)

6.49 Vector Struct Reference

3D floating-point vector.

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float [x](#)
X coordinate in 3D space.
- float [y](#)
Y coordinate in 3D space.
- float [z](#)
Z coordinate in 3D space.

6.49.1 Detailed Description

3D floating-point vector.

6.49.2 Member Data Documentation

6.49.2.1 [x](#)

```
float x
```

X coordinate in 3D space.

6.49.2.2 [y](#)

```
float y
```

Y coordinate in 3D space.

6.49.2.3 [z](#)

```
float z
```

Z coordinate in 3D space.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.50 Vector4 Struct Reference

4-component (XYZ + W) floating-point vector.

```
#include <Afterwarp.Types.h>
```

Public Attributes

- float **x**
X coordinate in 3D space.
- float **y**
Y coordinate in 3D space.
- float **z**
Z coordinate in 3D space.
- float **w**
W coordinate in 3D space.

6.50.1 Detailed Description

4-component (XYZ + W) floating-point vector.

6.50.2 Member Data Documentation

6.50.2.1 w

```
float w
```

W coordinate in 3D space.

6.50.2.2 x

```
float x
```

X coordinate in 3D space.

6.50.2.3 y

```
float y
```

Y coordinate in 3D space.

6.50.2.4 z

```
float z
```

Z coordinate in 3D space.

The documentation for this struct was generated from the following file:

- [Afterwarp.Types.h](#)

6.51 VertexElement Struct Reference

Structure that describes the layout of a single buffer element.

```
#include <Afterwarp.Structs.h>
```

Public Attributes

- char [name](#) [64]
Name of the component (variable name in GLSL or semantic name in HLSL).
- [ElementFormat](#) [format](#)
Data format of each value in this element.
- int32_t [count](#)
- uint32_t [channel](#)
- uint32_t [offset](#)
Offset in bytes relative to origin of the first buffer element to the current one.

6.51.1 Detailed Description

Structure that describes the layout of a single buffer element.

6.51.2 Member Data Documentation

6.51.2.1 channel

```
uint32_t channel
```

Channel to which the values of this element will be bound to. The most significant bit of this field has a special purpose, meaning that the data values are normalized (applies only to integers), so the number of channel is stored in first 31 bits.

6.51.2.2 count

```
int32_t count
```

Number of values used by this element. If "format" is "Undefined", this indicates number of bytes the element occupies.

6.51.2.3 format

`ElementFormat` format

Data format of each value in this element.

6.51.2.4 name

`char` name[64]

Name of the component (variable name in GLSL or semantic name in HLSL).

6.51.2.5 offset

`uint32_t` offset

Offset in bytes relative to origin of the first buffer element to the current one.

The documentation for this struct was generated from the following file:

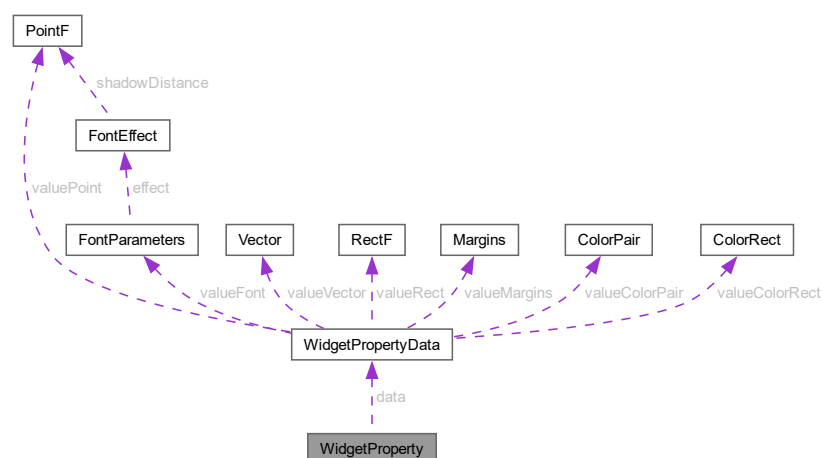
- [Afterwarp.Structs.h](#)

6.52 WidgetProperty Struct Reference

Information about a widget property.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for `WidgetProperty`:



Public Attributes

- char [name](#) [256]
Name of the property.
- [WidgetPropertyType](#) type
Data type of the property.
- [WidgetPropertyBehavior](#) behavior
Behavior of the property.
- uint16_t [attributes](#)
Additional attributes of the property.
- uint16_t [location](#)
- uint16_t [reserved](#)
Reserved bits, must be set to zero.
- [WidgetPropertyData](#) data
Contents of the property.

6.52.1 Detailed Description

Information about a widget property.

6.52.2 Member Data Documentation

6.52.2.1 attributes

uint16_t attributes

Additional attributes of the property.

6.52.2.2 behavior

[WidgetPropertyBehavior](#) behavior

Behavior of the property.

6.52.2.3 data

[WidgetPropertyData](#) data

Contents of the property.

6.52.2.4 location

uint16_t location

Location of the property in the list of existing properties. This is an opaque value that loosely determines next property to be iterated. The actual value is temporary and valid only during continuous iteration. Any change to properties invalidates all previous iteration values. A value of zero indicates an invalid or "undefined" location.

6.52.2.5 name

```
char name[256]
```

Name of the property.

6.52.2.6 reserved

```
uint16_t reserved
```

Reserved bits, must be set to zero.

6.52.2.7 type

```
WidgetPropertyType type
```

Data type of the property.

The documentation for this struct was generated from the following file:

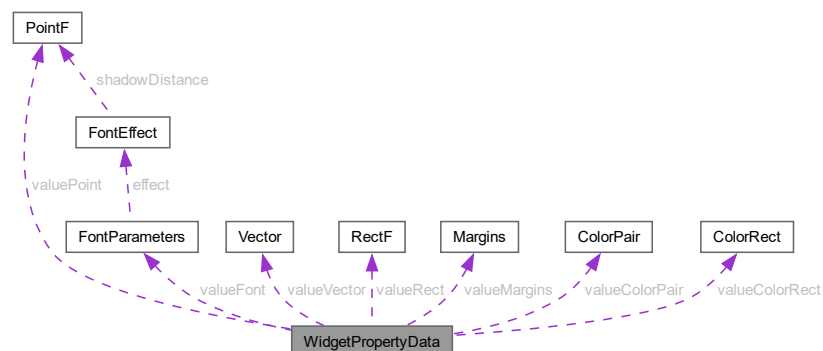
- [Afterwarp.Structs.h](#)

6.53 WidgetPropertyData Union Reference

A flexible structure for storing property data.

```
#include <Afterwarp.Structs.h>
```

Collaboration diagram for WidgetPropertyData:



Public Attributes

- `int32_t valueInteger`
Data represented as 32-bit signed integer.
- `int64_t valueInt64`
Data represented as 64-bit signed integer.
- `float valueFloat`
Data represented as 32-bit floating-point.
- `double valueReal`
Data represented as 64-bit floating-point.
- `bool valueBool`
Data represented as 8-bit boolean.
- `PointF valuePoint`
Data represented as 2-component 32-bit floating-point vector.
- `Vector valueVector`
Data represented as 3-component 32-bit floating-point vector.
- `RectF valueRect`
Data represented as a rectangle composed of four floating-point values.
- `Margins valueMargins`
Data represented as margins containing four floating-point values.
- `Color valueColor`
Data represented as 32-bit unsigned integer color.
- `ColorPair valueColorPair`
Data represented as two 32-bit unsigned integer colors.
- `ColorRect valueColorRect`
Data represented as four 32-bit unsigned integer colors.
- `uint8_t valueEnum`
Data represented as 8-bit unsigned integer enumeration.
- `char valueString [184]`
- `FontParameters valueFont`
Data represented as font parameters structure.

6.53.1 Detailed Description

A flexible structure for storing property data.

6.53.2 Member Data Documentation**6.53.2.1 valueBool**

```
bool valueBool
```

Data represented as 8-bit boolean.

6.53.2.2 valueColor

```
Color valueColor
```

Data represented as 32-bit unsigned integer color.

6.53.2.3 valueColorPair

`ColorPair` valueColorPair

Data represented as two 32-bit unsigned integer colors.

6.53.2.4 valueColorRect

`ColorRect` valueColorRect

Data represented as four 32-bit unsigned integer colors.

6.53.2.5 valueEnum

`uint8_t` valueEnum

Data represented as 8-bit unsigned integer enumeration.

6.53.2.6 valueFloat

`float` valueFloat

Data represented as 32-bit floating-point.

6.53.2.7 valueFont

`FontParameters` valueFont

Data represented as font parameters structure.

6.53.2.8 valueInt64

`int64_t` valueInt64

Data represented as 64-bit signed integer.

6.53.2.9 valueInteger

`int32_t` valueInteger

Data represented as 32-bit signed integer.

6.53.2.10 valueMargins

`Margins` valueMargins

Data represented as margins containing four floating-point values.

6.53.2.11 valuePoint

```
PointF valuePoint
```

Data represented as 2-component 32-bit floating-point vector.

6.53.2.12 valueReal

```
double valueReal
```

Data represented as 64-bit floating-point.

6.53.2.13 valueRect

```
RectF valueRect
```

Data represented as a rectangle composed of four floating-point values.

6.53.2.14 valueString

```
char valueString[184]
```

Data represented as 184-byte null-terminated string. The first byte is actually a string length; if this byte is set to 255, then the string does not fit and must be retrieved using other means.

6.53.2.15 valueVector

```
Vector valueVector
```

Data represented as 3-component 32-bit floating-point vector.

The documentation for this union was generated from the following file:

- [Afterwarp.Structs.h](#)

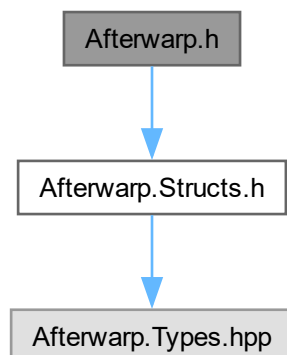
Chapter 7

File Documentation

7.1 Afterwarp.h File Reference

```
#include "Afterwarp.Structs.h"
```

Include dependency graph for Afterwarp.h:



Functions

- AFTERWARP_API [LibraryBool](#) [LibraryGetVersion](#) (int32_t *major, int32_t *minor, uint32_t *build)
Returns current version of the shared library.
- AFTERWARP_API void [LibrarySerialCode](#) (char const *serial, int32_t length)
Specifies a business serial key for the shared library.
- AFTERWARP_API struct SwapChain_t * [SwapChainCreate](#) (struct Device_t *device, [UntypedHandle](#) windowHandle, [Point](#) const *Size, [PixelFormat](#) format, [PixelFormat](#) depthStencil, int32_t multisamples, [LibraryBool](#) vsync)
- AFTERWARP_API void [SwapChainDestroy](#) (struct SwapChain_t *swapChain)
Releases given instance of rendering swap-chain.
- AFTERWARP_API struct Device_t * [SwapChainGetDevice](#) (struct SwapChain_t *swapChain)

- Returns device associated with the given swap-chain.*

 - AFTERWARP_API [UntypedHandle SwapChainGetWindowHandle](#) (struct SwapChain_t *swapChain)

Returns handle of the associated window.
- AFTERWARP_API int32_t [SwapChainGetWidth](#) (struct SwapChain_t *swapChain)

Returns swap-chain width.
- AFTERWARP_API int32_t [SwapChainGetHeight](#) (struct SwapChain_t *swapChain)

Returns swap-chain height.
- AFTERWARP_API [PixelFormat SwapChainGetFormat](#) (struct SwapChain_t *swapChain)

Returns pixel format of swap-chain rendering surface.
- AFTERWARP_API [PixelFormat SwapChainGetDepthStencil](#) (struct SwapChain_t *swapChain)

Returns pixel format of swap-chain depth/stencil surface.
- AFTERWARP_API int32_t [SwapChainGetMultisamples](#) (struct SwapChain_t *swapChain)

Returns number of samples of swap-chain surfaces.
- AFTERWARP_API [LibraryBool SwapChainGetVSync](#) (struct SwapChain_t *swapChain)

Returns whether to wait for a vertical retrace.
- AFTERWARP_API [LibraryBool SwapChainResize](#) (struct SwapChain_t *swapChain, [Point](#) const *size)

Resizes the swap-chain.
- AFTERWARP_API [LibraryBool SwapChainBegin](#) (struct SwapChain_t *swapChain)

Begins rendering on the swap-chain.
- AFTERWARP_API void [SwapChainEnd](#) (struct SwapChain_t *swapChain)

Finishes rendering on the swap-chain and flips the buffers.
- AFTERWARP_API struct Device_t * [DeviceCreate](#) (struct [Application_t](#) *application, [DeviceTechnology](#) technology, [DeviceAttributes](#) attributes)

Creates a new graphics device of the given technology and attributes.
- AFTERWARP_API struct Device_t * [DeviceCreateWrapped](#) (struct [Application_t](#) *application, [DeviceTechnology](#) technology, [DeviceAttributes](#) attributes, [UntypedHandle](#) platformDevice)
- AFTERWARP_API void [DeviceDestroy](#) (struct Device_t *device)

Releases given instance of device.
- AFTERWARP_API [UntypedHandle DeviceGetPlatformDevice](#) (struct Device_t *device)

Returns platform-specific device context.
- AFTERWARP_API [DeviceAttributes DeviceGetAttributes](#) (struct Device_t *device)

Returns attributes with which the device has been created.
- AFTERWARP_API [DeviceBehavior DeviceGetBehavior](#) (struct Device_t *device)

Returns device behavior attributes.
- AFTERWARP_API [DeviceLegacyBits DeviceGetLegacyBits](#) (struct Device_t *device)

Returns legacy feature bits that define older hardware features.
- AFTERWARP_API [DeviceTechnology DeviceGetTechnology](#) (struct Device_t *device)

Returns graphics technology type that is currently being used.
- AFTERWARP_API [DevicePlatform DeviceGetPlatform](#) (struct Device_t *device)

Returns OS platform the device application is currently running on.
- AFTERWARP_API uint32_t [DeviceGetTechVersion](#) (struct Device_t *device)
- AFTERWARP_API uint32_t [DeviceGetTechFeatureVersion](#) (struct Device_t *device)
- AFTERWARP_API void [DeviceResetCache](#) (struct Device_t *device)

Releases any cached device resources.
- AFTERWARP_API [LibraryBool DeviceClear](#) (struct Device_t *device, uint8_t clearLayers, [FloatColor](#) const *clearColor, float clearDepth, uint32_t clearStencil)

Clears rendering surface that is currently active.
- AFTERWARP_API [LibraryBool DeviceSetRenderingState](#) (struct Device_t *device, [RenderingState](#) const *state)

Sets new rendering parameters and operation characteristics.
- AFTERWARP_API void [DeviceGetRenderingState](#) (struct Device_t *device, [RenderingState](#) *state)

- Returns current rendering states.*
- AFTERWARP_API [LibraryBool DeviceSetViewport](#) (struct Device_t *device, [Rect](#) const *viewport)
Specifies new viewport parameters.
- AFTERWARP_API void [DeviceGetViewport](#) (struct Device_t *device, [Rect](#) *viewport)
Returns current viewport parameters.
- AFTERWARP_API [LibraryBool DeviceSetScissor](#) (struct Device_t *device, [Rect](#) const *scissor)
Specifies new scissor parameters.
- AFTERWARP_API void [DeviceGetScissor](#) (struct Device_t *device, [Rect](#) *scissor)
Returns current scissor parameters.
- AFTERWARP_API void [DeviceMemoryBarrier](#) (struct Device_t *device, uint32_t barrierBits)
Issues a memory barrier according to the specified bit flags.
- AFTERWARP_API struct [Buffer_t](#) * [BufferCreate](#) (struct Device_t *device, [BufferData](#) dataType, [BufferAccess](#) accessType, uint32_t size, uint32_t pitch, [PixelFormat](#) format, void const *initialData)
- AFTERWARP_API void [BufferDestroy](#) (struct [Buffer_t](#) *buffer)
Releases given instance of hardware buffer.
- AFTERWARP_API struct Device_t * [BufferGetDevice](#) (struct [Buffer_t](#) *buffer)
Returns device associated with the given buffer.
- AFTERWARP_API [UntypedHandle](#) [BufferGetPlatformHandle](#) (struct [Buffer_t](#) *buffer)
Returns platform-specific buffer handle.
- AFTERWARP_API [BufferData](#) [BufferGetDataType](#) (struct [Buffer_t](#) *buffer)
Returns the type of data that buffer contains.
- AFTERWARP_API [BufferAccess](#) [BufferGetAccessType](#) (struct [Buffer_t](#) *buffer)
Returns the access type of for buffer data.
- AFTERWARP_API uint32_t [BufferGetSize](#) (struct [Buffer_t](#) *buffer)
Returns buffer size in bytes.
- AFTERWARP_API uint32_t [BufferGetPitch](#) (struct [Buffer_t](#) *buffer)
Returns the number of bytes for offset to move for advancing the pointer from one element to another.
- AFTERWARP_API [PixelFormat](#) [BufferGetFormat](#) (struct [Buffer_t](#) *buffer)
Returns pixel format that represents the buffer contents.
- AFTERWARP_API [LibraryBool](#) [BufferUpdate](#) (struct [Buffer_t](#) *buffer, void const *data, uint32_t offset, uint32_t size)
- AFTERWARP_API [LibraryBool](#) [BufferRetrieve](#) (struct [Buffer_t](#) *buffer, void *data, uint32_t offset, uint32_t size)
- AFTERWARP_API [LibraryBool](#) [BufferCopy](#) (struct [Buffer_t](#) *buffer, struct [Buffer_t](#) *bufferSource, uint32_t offsetDest, uint32_t offsetSource, uint32_t size)
- AFTERWARP_API struct [Program_t](#) * [ProgramCreate](#) (struct Device_t *device, _In_opt_ [VertexElement](#) const vertexElements[], int32_t vertexElementCount, _In_opt_ [ProgramElement](#) const programElements[], int32_t programElementCount, _In_opt_ [ProgramVariable](#) const programVariables[], int32_t programVariableCount, _In_opt_ char const *shaderVertex, int32_t shaderVertexLength, _In_opt_ char const *shaderHull, int32_t shaderHullLength, _In_opt_ char const *shaderDomain, int32_t shaderDomainLength, _In_opt_ char const *shaderGeometry, int32_t shaderGeometryLength, _In_opt_ char const *shaderPixel, int32_t shaderPixelLength)
Creates a new instance of shader program.
- AFTERWARP_API void [ProgramDestroy](#) (struct [Program_t](#) *program)
Releases given instance of shader program.
- AFTERWARP_API struct Device_t * [ProgramGetDevice](#) (struct [Program_t](#) *program)
Returns device associated with the given program.
- AFTERWARP_API [LibraryBool](#) [ProgramUpdateByIndex](#) (struct [Program_t](#) *program, int32_t variableIndex, void const *variableData, uint32_t size)
- AFTERWARP_API [LibraryBool](#) [ProgramUpdateByName](#) (struct [Program_t](#) *program, char const *variableName, void const *variableData, uint32_t size)
- AFTERWARP_API [LibraryBool](#) [ProgramBind](#) (struct [Program_t](#) *program, struct [Buffer_t](#) *buffer, uint32_t channel, uint32_t offset)

- AFTERWARP_API void [ProgramUnbind](#) (struct [Program_t](#) *program, struct [Buffer_t](#) *buffer, uint32_t channel)
Removes association that was previously made with [ProgramBind](#) between the buffer and the given channel.
- AFTERWARP_API void [ProgramResetBindings](#) (struct [Program_t](#) *program)
Resets any bindings that were previously made as if [ProgramUnbind](#) would be called for each of them.
- AFTERWARP_API void [ProgramResetCache](#) (struct [Program_t](#) *program)
- AFTERWARP_API [LibraryBool](#) [ProgramCommit](#) (struct [Program_t](#) *program)
- AFTERWARP_API [LibraryBool](#) [ProgramBegin](#) (struct [Program_t](#) *program)
Activates the shader program and prepares for rendering.
- AFTERWARP_API void [ProgramEnd](#) (struct [Program_t](#) *program)
Deactivates the shader program, resets previously active input streams and bindings.
- AFTERWARP_API [LibraryBool](#) [ProgramDraw](#) (struct [Program_t](#) *program, [PrimitiveTopology](#) topology, int32_t vertexCount, int32_t baseVertex)
Renders primitives with the given number of vertices.
- AFTERWARP_API [LibraryBool](#) [ProgramDrawIndexed](#) (struct [Program_t](#) *program, [PrimitiveTopology](#) topology, int32_t indexCount, int32_t firstIndex, int32_t baseVertex)
Renders indexed primitives with the given number of indices.
- AFTERWARP_API [LibraryBool](#) [ProgramDrawInstances](#) (struct [Program_t](#) *program, [PrimitiveTopology](#) topology, int32_t vertexCount, int32_t instanceCount, int32_t baseVertex)
Renders multiple number of instances of the given primitives.
- AFTERWARP_API [LibraryBool](#) [ProgramDrawInstancesIndexed](#) (struct [Program_t](#) *program, [PrimitiveTopology](#) topology, int32_t indexCount, int32_t instanceCount, int32_t firstIndex, int32_t baseVertex)
Renders multiple number of instances of the given indexed primitives.
- AFTERWARP_API [LibraryBool](#) [ProgramSetPatchVertices](#) (struct [Program_t](#) *program, int32_t patchVertices)
Specifies number of vertices in each patch. This only applies when rendering patch topology.
- AFTERWARP_API struct [ComputeProgram_t](#) * [ComputeProgramCreate](#) (struct [Device_t](#) *device, [_In_opt_](#) [_ProgramElement](#) const programElements[], int32_t programElementCount, [_In_opt_](#) char const *shader, int32_t shaderLength)
Creates a new instance of compute shader program.
- AFTERWARP_API void [ComputeProgramDestroy](#) (struct [ComputeProgram_t](#) *program)
Releases given instance of compute shader program.
- AFTERWARP_API struct [Device_t](#) * [ComputeProgramGetDevice](#) (struct [ComputeProgram_t](#) *program)
Returns device associated with the given compute program.
- AFTERWARP_API [LibraryBool](#) [ComputeProgramBindBuffer](#) (struct [ComputeProgram_t](#) *program, struct [Buffer_t](#) *buffer, uint32_t channel, uint32_t offset)
- AFTERWARP_API void [ComputeProgramUnbindBuffer](#) (struct [ComputeProgram_t](#) *program, struct [Buffer_t](#) *buffer, uint32_t channel)
- AFTERWARP_API [LibraryBool](#) [ComputeProgramBindTexture](#) (struct [ComputeProgram_t](#) *program, struct [Texture_t](#) *texture, [ComputeBindTextureFormat](#) const *bindFormat)
Assigns texture to a particular channel.
- AFTERWARP_API void [ComputeProgramUnbindTexture](#) (struct [ComputeProgram_t](#) *program, struct [Texture_t](#) *texture, [ComputeBindTextureFormat](#) const *bindFormat)
Removes existing texture association with the particular channel.
- AFTERWARP_API void [ComputeProgramResetBindings](#) (struct [ComputeProgram_t](#) *program)
- AFTERWARP_API [LibraryBool](#) [ComputeProgramCommit](#) (struct [ComputeProgram_t](#) *program)
- AFTERWARP_API [LibraryBool](#) [ComputeProgramBegin](#) (struct [ComputeProgram_t](#) *program)
Activates the compute shader program.
- AFTERWARP_API void [ComputeProgramEnd](#) (struct [ComputeProgram_t](#) *program)
Deactivates the compute shader program.
- AFTERWARP_API [LibraryBool](#) [ComputeProgramDispatch](#) (struct [ComputeProgram_t](#) *program, int32_t groupsX, int32_t groupsY, int32_t groupsZ)
Launches one or more compute shader work groups.

- AFTERWARP_API struct [Sampler_t](#) * [SamplerCreate](#) (struct [Device_t](#) *device, [SamplerState](#) const *samplerState)
Creates a new instance of sampler object.
- AFTERWARP_API void [SamplerDestroy](#) (struct [Sampler_t](#) *sampler)
Releases given instance of sampler object.
- AFTERWARP_API struct [Device_t](#) * [SamplerGetDevice](#) (struct [Sampler_t](#) *sampler)
Returns device associated with the given sampler.
- AFTERWARP_API void [SamplerGetState](#) (struct [Sampler_t](#) *sampler, [SamplerState](#) *samplerState)
Retrieves parameters associated with the sampler object.
- AFTERWARP_API [LibraryBool](#) [SamplerSetState](#) (struct [Sampler_t](#) *sampler, [SamplerState](#) const *samplerState)
Updates parameters associated with the sampler object.
- AFTERWARP_API [LibraryBool](#) [SamplerBind](#) (struct [Sampler_t](#) *sampler, uint32_t channel)
Binds sampler object to a particular channel.
- AFTERWARP_API void [SamplerUnbind](#) (struct [Sampler_t](#) *sampler, uint32_t channel)
Unbinds sampler object from the particular channel.
- AFTERWARP_API struct [Texture_t](#) * [TextureCreate](#) (struct [Device_t](#) *device, [TextureParameters](#) const *parameters)
Creates a new instance of texture object.
- AFTERWARP_API struct [Texture_t](#) * [TextureCreateFromFile](#) (struct [Device_t](#) *device, char const *fileName, [PixelFormat](#) format, [TextureAttributes](#) attributes)
- AFTERWARP_API struct [Texture_t](#) * [TextureCreateFromFileParallax](#) (struct [Device_t](#) *device, char const *fileName, [PixelFormat](#) format, [TextureAttributes](#) attributes)
Creates texture containing displacement map for Parallax Mapping technique from an external file on disk.
- AFTERWARP_API struct [Texture_t](#) * [TextureCreateFromFileNormalsAndOcclusion](#) (struct [Device_t](#) *device, char const *fileNameNormalMap, char const *fileNameOcclusionMap, [TextureAttributes](#) attributes)
- AFTERWARP_API struct [Texture_t](#) * [TextureCreateFromFileInMemory](#) (struct [Device_t](#) *device, void const *fileContents, uint32_t contentSize, char const *extension, [PixelFormat](#) format, [TextureAttributes](#) attributes)
- AFTERWARP_API struct [Texture_t](#) * [TextureCreateFromSurface](#) (struct [Device_t](#) *device, struct [Surface_t](#) *surface, [TextureAttributes](#) attributes)
Creates a new instance of texture object loading its contents from an existing surface.
- AFTERWARP_API void [TextureDestroy](#) (struct [Texture_t](#) *texture)
Releases given instance of texture object.
- AFTERWARP_API struct [Device_t](#) * [TextureGetDevice](#) (struct [Texture_t](#) *texture)
Returns device associated with the given texture.
- AFTERWARP_API [UntypedHandle](#) [TextureGetPlatformHandle](#) (struct [Texture_t](#) *texture)
Returns platform-specific texture handle.
- AFTERWARP_API void [TextureGetParameters](#) (struct [Texture_t](#) *texture, [TextureParameters](#) *parameters)
Retrieves current texture parameters.
- AFTERWARP_API [LibraryBool](#) [TextureUpdate](#) (struct [Texture_t](#) *texture, void const *content, uint32_t pitch, int32_t layer, [Rect](#) const *rect, int32_t mipLevel)
- AFTERWARP_API [LibraryBool](#) [TextureRetrieve](#) (struct [Texture_t](#) *texture, void *content, uint32_t pitch, int32_t layer, [Rect](#) const *rect, int32_t mipLevel)
- AFTERWARP_API [LibraryBool](#) [TextureCopy](#) (struct [Texture_t](#) *texture, struct [Texture_t](#) *source, int32_t destLayer, [Point](#) const *destPos, int32_t srcLayer, [Rect](#) const *sourceRect, int32_t destMipLevel, int32_t srcMipLevel)
- AFTERWARP_API [LibraryBool](#) [TextureCopyFromSurface](#) (struct [Texture_t](#) *texture, struct [Surface_t](#) *surface, int32_t layer, [Point](#) const *destPos, [Rect](#) const *sourceRect, int32_t mipLevel)
- AFTERWARP_API [LibraryBool](#) [TextureCopyToSurface](#) (struct [Texture_t](#) *texture, struct [Surface_t](#) *surface, int32_t layer, [Point](#) const *destPos, [Rect](#) const *sourceRect, int32_t mipLevel)
- AFTERWARP_API [LibraryBool](#) [TextureLoadFromFile](#) (struct [Texture_t](#) *texture, char const *fileName, int32_t destLayer, [Point](#) const *destPos, [Rect](#) const *sourceRect, int32_t mipLevel)
Loads image from disk and copies it to the specified location and parameters.

- AFTERWARP_API [LibraryBool TextureSaveToFile](#) (struct [Texture_t](#) *texture, char const *fileName, int32_t quality, int32_t layer, [Rect](#) const *sourceRect, int32_t mipLevel)
Saves the appropriate portion of texture to external file.
- AFTERWARP_API [LibraryBool TextureClear](#) (struct [Texture_t](#) *texture)
Clears entire texture surface and fills pixels with zeros.
- AFTERWARP_API [LibraryBool TextureClearWith](#) (struct [Texture_t](#) *texture, [FloatColor](#) const *color, int32_t layer, int32_t mipLevel)
- AFTERWARP_API [LibraryBool TextureBind](#) (struct [Texture_t](#) *texture, uint32_t channel)
Binds texture to the specified rendering channel.
- AFTERWARP_API [LibraryBool TextureUnbind](#) (struct [Texture_t](#) *texture, uint32_t channel)
Unbinds texture from the specified rendering channel.
- AFTERWARP_API [LibraryBool TextureAttach](#) (struct [Texture_t](#) *texture, struct [Texture_t](#) *attachment, int32_t layer, int32_t mipLevel)
- AFTERWARP_API void [TextureDetach](#) (struct [Texture_t](#) *texture)
Removes all previous drawable texture attachments.
- AFTERWARP_API [LibraryBool TextureBegin](#) (struct [Texture_t](#) *texture, int32_t layer, int32_t mipLevel)
Starts rendering on the drawable texture.
- AFTERWARP_API [LibraryBool TextureEnd](#) (struct [Texture_t](#) *texture)
Finishes rendering on the drawable texture.
- AFTERWARP_API [LibraryBool TextureGenerateMipMaps](#) (struct [Texture_t](#) *texture)
Generates texture's mipmaps from its base (level = 0) image.
- AFTERWARP_API void [TextureResetCache](#) (struct [Texture_t](#) *texture)
- AFTERWARP_API struct [Surface_t](#) * [SurfaceCreate](#) (int32_t width, int32_t height, [PixelFormat](#) format)
Creates a new instance of raster surface.
- AFTERWARP_API struct [Surface_t](#) * [SurfaceCreateFromFile](#) (char const *fileName, [AlphaFormatRequest](#) formatRequest)
- AFTERWARP_API struct [Surface_t](#) * [SurfaceCreateFromFileInMemory](#) (void const *fileContents, uint32_t contentSize, char const *extension, [AlphaFormatRequest](#) formatRequest)
- AFTERWARP_API void [SurfaceDestroy](#) (struct [Surface_t](#) *surface)
Releases given instance of raster surface.
- AFTERWARP_API void * [SurfaceGetBits](#) (struct [Surface_t](#) *surface)
- AFTERWARP_API uint32_t [SurfaceGetPitch](#) (struct [Surface_t](#) *surface)
- AFTERWARP_API int32_t [SurfaceGetBytesPerPixel](#) (struct [Surface_t](#) *surface)
Returns number of bytes each pixel occupies.
- AFTERWARP_API int32_t [SurfaceGetWidth](#) (struct [Surface_t](#) *surface)
Returns surface width in pixels.
- AFTERWARP_API int32_t [SurfaceGetHeight](#) (struct [Surface_t](#) *surface)
Returns surface height in pixels.
- AFTERWARP_API uint32_t [SurfaceGetByteSize](#) (struct [Surface_t](#) *surface)
Returns surface size in bytes.
- AFTERWARP_API [PixelFormat](#) [SurfaceGetFormat](#) (struct [Surface_t](#) *surface)
Returns pixel format in which pixels are stored.
- AFTERWARP_API [LibraryBool SurfaceGetPremultipliedAlpha](#) (struct [Surface_t](#) *surface)
- AFTERWARP_API void [SurfaceSetPremultipliedAlpha](#) (struct [Surface_t](#) *surface, [LibraryBool](#) premultipliedAlpha)
- AFTERWARP_API void * [SurfaceGetScanline](#) (struct [Surface_t](#) *surface, int32_t index)
Provides pointer to a first pixel at the given scanline index (that is, row number).
- AFTERWARP_API void * [SurfaceGetPixelPtr](#) (struct [Surface_t](#) *surface, int32_t x, int32_t y)
Provides pointer to the pixel data at the given coordinates.
- AFTERWARP_API [LibraryBool SurfaceEmpty](#) (struct [Surface_t](#) *surface)
- AFTERWARP_API [LibraryBool SurfaceResize](#) (struct [Surface_t](#) *surface, int32_t width, int32_t height, [PixelFormat](#) format)

- Redefines surface size to the specified width, height and pixel format, discarding previous contents.*
- AFTERWARP_API [LibraryBool SurfaceConvertFormat](#) (struct [Surface_t](#) *surface, [PixelFormat](#) format)
- Converts surface from its current format to another one.*
- AFTERWARP_API [Color SurfaceGetPixel](#) (struct [Surface_t](#) *surface, int32_t x, int32_t y)
- Reads a single pixel from the surface.*
- AFTERWARP_API void [SurfaceSetPixel](#) (struct [Surface_t](#) *surface, int32_t x, int32_t y, [Color](#) color)
- Writes a single pixel to the surface.*
- AFTERWARP_API [Color SurfaceGetPixelBilinear](#) (struct [Surface_t](#) *surface, float x, float y)
- AFTERWARP_API [LibraryBool SurfaceCopyRect](#) (struct [Surface_t](#) *surface, struct [Surface_t](#) *source, [Rect](#) const *sourceRect, [Point](#) const *destPos)
- AFTERWARP_API [LibraryBool SurfaceCopyFrom](#) (struct [Surface_t](#) *surface, struct [Surface_t](#) *source)
- AFTERWARP_API void [SurfaceClear](#) (struct [Surface_t](#) *surface)
- Clears the entire surface with zeros.*
- AFTERWARP_API [LibraryBool SurfaceClearWith](#) (struct [Surface_t](#) *surface, [Color](#) color)
- Clears the entire surface with a given color. This does pixel format conversion when appropriate.*
- AFTERWARP_API [LibraryBool SurfaceResetAlpha](#) (struct [Surface_t](#) *surface, [LibraryBool](#) opaque)
- Processes surface pixels, setting alpha-channel to either fully translucent or fully opaque.*
- AFTERWARP_API [LibraryBool SurfaceHasAlphaChannel](#) (struct [Surface_t](#) *surface)
- AFTERWARP_API [LibraryBool SurfacePremultiplyAlpha](#) (struct [Surface_t](#) *surface)
- AFTERWARP_API [LibraryBool SurfaceUnpremultiplyAlpha](#) (struct [Surface_t](#) *surface)
- AFTERWARP_API [LibraryBool SurfaceMirror](#) (struct [Surface_t](#) *surface)
- Mirrors the visible image on surface horizontally.*
- AFTERWARP_API [LibraryBool SurfaceFlip](#) (struct [Surface_t](#) *surface)
- Flips the visible image on surface vertically.*
- AFTERWARP_API [LibraryBool SurfaceInvert](#) (struct [Surface_t](#) *surface)
- Inverts colors in the surface.*
- AFTERWARP_API [LibraryBool SurfaceShrinkFrom](#) (struct [Surface_t](#) *surface, struct [Surface_t](#) *source)
- AFTERWARP_API [LibraryBool SurfaceStretchRect](#) (struct [Surface_t](#) *surface, struct [Surface_t](#) *source, [RectF](#) const *destRect, [RectF](#) const *sourceRect)
- AFTERWARP_API [LibraryBool SurfaceStretchRectBilinear](#) (struct [Surface_t](#) *surface, struct [Surface_t](#) *source, [RectF](#) const *destRect, [RectF](#) const *sourceRect)
- AFTERWARP_API [LibraryBool SurfaceMakeSignedDistanceField](#) (struct [Surface_t](#) *surface, struct [Surface_t](#) *source, float spread, [Point](#) const *destPos, [Rect](#) const *sourceRect)
- Calculates signed distance field from alpha values of the source.*
- AFTERWARP_API [LibraryBool SurfaceSaveToFile](#) (struct [Surface_t](#) *surface, char const *fileName, int32_t quality)
- AFTERWARP_API uint32_t [SurfaceSaveToFileInMemory](#) (struct [Surface_t](#) *surface, void *fileContents, uint32_t fileContentSize, char const *extension, int32_t quality)
- AFTERWARP_API struct [CanvasBuffer_t](#) * [CanvasBufferCreate](#) (void)
- AFTERWARP_API void [CanvasBufferDestroy](#) (struct [CanvasBuffer_t](#) *buffer)
- Releases an existing instance of canvas buffer container.*
- AFTERWARP_API int32_t [CanvasBufferGetVertexCount](#) (struct [CanvasBuffer_t](#) *buffer)
- Returns existing number of vertices.*
- AFTERWARP_API [LibraryBool CanvasBufferSetVertexCount](#) (struct [CanvasBuffer_t](#) *buffer, int32_t vertexCount)
- Adjusts the existing number of vertices.*
- AFTERWARP_API int32_t [CanvasBufferGetIndexCount](#) (struct [CanvasBuffer_t](#) *buffer)
- Returns existing number of indices.*
- AFTERWARP_API [LibraryBool CanvasBufferSetIndexCount](#) (struct [CanvasBuffer_t](#) *buffer, int32_t indexCount)
- Adjusts the existing number of indices.*
- AFTERWARP_API void [CanvasBufferClear](#) (struct [CanvasBuffer_t](#) *buffer)
- Clears all buffers.*

- AFTERWARP_API void [CanvasBufferClearAndShrink](#) (struct CanvasBuffer_t *buffer)
Clears all buffers and releases their memory.
- AFTERWARP_API [PointF](#) * [CanvasBufferGetVertices](#) (struct CanvasBuffer_t *buffer)
Returns pointer to an existing array of vertices.
- AFTERWARP_API [Color](#) * [CanvasBufferGetColors](#) (struct CanvasBuffer_t *buffer)
Returns pointer to an existing array of colors.
- AFTERWARP_API int32_t * [CanvasBufferGetIndices](#) (struct CanvasBuffer_t *buffer)
Returns pointer to an existing array of vertex indices.
- AFTERWARP_API void [CanvasBufferGetExtents](#) (struct CanvasBuffer_t *buffer, [RectF](#) *extents)
Calculates boundaries for the existing vertices.
- AFTERWARP_API struct PathBroker_t * [PathBrokerCreate](#) (void)
Creates a new module for creating paths.
- AFTERWARP_API void [PathBrokerDestroy](#) (struct PathBroker_t *broker)
Releases an existing instance of path creation module.
- AFTERWARP_API [LibraryBool](#) [PathBrokerFill](#) (struct PathBroker_t *broker, struct CanvasBuffer_t *buffer, [PathElement](#) const path[], int32_t pathLength, [ColorRect](#) const *colors, [LibraryBool](#) quality)
- AFTERWARP_API [LibraryBool](#) [PathBrokerStroke](#) (struct PathBroker_t *broker, struct CanvasBuffer_t *buffer, [PathElement](#) const path[], int32_t pathLength, float thickness, [Color](#) color, [PathJoint](#) joints, [LineCaps](#) caps, float miterLimit, float smoothStep)
Pre-renders a stroke for the given path.
- AFTERWARP_API void [PathBrokerReset](#) (struct PathBroker_t *broker)
Resets internal cache and releases any allocated memory.
- AFTERWARP_API struct [Canvas_t](#) * [CanvasCreate](#) (struct Device_t *device)
Creates new 2D rendering canvas.
- AFTERWARP_API void [CanvasDestroy](#) (struct [Canvas_t](#) *canvas)
Releases given instance of 2D rendering canvas.
- AFTERWARP_API struct Device_t * [CanvasGetDevice](#) (struct [Canvas_t](#) *canvas)
Returns device associated with the given canvas.
- AFTERWARP_API [LibraryBool](#) [CanvasBegin](#) (struct [Canvas_t](#) *canvas)
Starts rendering with the canvas.
- AFTERWARP_API void [CanvasEnd](#) (struct [Canvas_t](#) *canvas)
Finishes rendering with the canvas.
- AFTERWARP_API void [CanvasPixels](#) (struct [Canvas_t](#) *canvas, [PointF](#) const positions[], [Color](#) const colors[], int32_t elementCount, [BlendingEffect](#) effect)
Draws pixels with the given positions and colors.
- AFTERWARP_API void [CanvasLines](#) (struct [Canvas_t](#) *canvas, [PointF](#) const vertices[], [Color](#) const colors[], int32_t const indices[], int32_t vertexCount, int32_t primitiveCount, [BlendingEffect](#) effect)
Draws lines with the given vertices, colors and indices.
- AFTERWARP_API void [CanvasTriangles](#) (struct [Canvas_t](#) *canvas, [PointF](#) const vertices[], [Color](#) const colors[], int32_t const indices[], int32_t vertexCount, int32_t primitiveCount, [BlendingEffect](#) effect)
Draws triangles with the given vertices, colors and indices.
- AFTERWARP_API void [CanvasTexturedTriangles](#) (struct [Canvas_t](#) *canvas, struct [Texture_t](#) *texture, [PointF](#) const vertices[], [PointF](#) const texCoords[], [Color](#) const colors[], int32_t const indices[], int32_t vertexCount, int32_t primitiveCount, [BlendingEffect](#) effect)
Draws triangles with the given vertices, texture coordinates, colors and indices.
- AFTERWARP_API void [CanvasPixel](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *position, [Color](#) color, [BlendingEffect](#) effect)
Draws a pixel with the given position and color.
- AFTERWARP_API void [CanvasLine](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *srcPos, [PointF](#) const *destPos, [ColorPair](#) const *colors, [BlendingEffect](#) effect)
Draws a line with the given positions and colors.

- AFTERWARP_API void [CanvasThickLine](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *srcPos, [PointF](#) const *destPos, [ColorPair](#) const *colors, float thickness, [BlendingEffect](#) effect)
Draws a line of varying thickness with the given positions and colors.
- AFTERWARP_API void [CanvasLineEllipse](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *radius, int32_t steps, [Color](#) color, [BlendingEffect](#) effect)
Draws ellipse with the given origin, radiuses, color and steps using "line" primitive.
- AFTERWARP_API void [CanvasLineCircle](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, float radius, int32_t steps, [Color](#) color, [BlendingEffect](#) effect)
Draws circle with the given origin, radius, color and steps using "line" primitive.
- AFTERWARP_API void [CanvasThickLineEllipse](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *radius, int32_t steps, [Color](#) color, float thickness, [BlendingEffect](#) effect)
Draws ellipse of varying thickness with the given origin, radiuses, color and steps.
- AFTERWARP_API void [CanvasThickLineCircle](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, float radius, int32_t steps, [Color](#) color, float thickness, [BlendingEffect](#) effect)
Draws circle of varying thickness with the given origin, radius, color and steps.
- AFTERWARP_API void [CanvasLineTriangle](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *vertex1, [PointF](#) const *vertex2, [PointF](#) const *vertex3, [Color](#) color1, [Color](#) color2, [Color](#) color3, [BlendingEffect](#) effect)
Draws lines between specified triangle vertices and colors.
- AFTERWARP_API void [CanvasThickLineTriangle](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *vertex1, [PointF](#) const *vertex2, [PointF](#) const *vertex3, [Color](#) color1, [Color](#) color2, [Color](#) color3, float thickness, [BlendingEffect](#) effect)
Draws line path of varying thickness between specified triangle vertices and colors.
- AFTERWARP_API void [CanvasLineQuad](#) (struct [Canvas_t](#) *canvas, [Quad](#) const *vertices, [ColorRect](#) const *colors, [BlendingEffect](#) effect)
Draws a wireframe quadrilateral with the given vertices and colors using "line" primitive.
- AFTERWARP_API void [CanvasThickLineQuad](#) (struct [Canvas_t](#) *canvas, [Quad](#) const *vertices, [ColorRect](#) const *colors, float thickness, [BlendingEffect](#) effect)
Draws a wireframe quadrilateral of varying thickness with the given vertices and colors.
- AFTERWARP_API void [CanvasLineHexagon](#) (struct [Canvas_t](#) *canvas, [Color](#) color1, [Color](#) color2, [Color](#) color3, [Color](#) color4, [Color](#) color5, [Color](#) color6, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasLineHexagonGrad](#) (struct [Canvas_t](#) *canvas, [ColorRect](#) const *colors, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasThickLineHexagon](#) (struct [Canvas_t](#) *canvas, [Color](#) color1, [Color](#) color2, [Color](#) color3, [Color](#) color4, [Color](#) color5, [Color](#) color6, float thickness, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasThickLineHexagonGrad](#) (struct [Canvas_t](#) *canvas, [ColorRect](#) const *colors, float thickness, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasTriangle](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *vertex1, [PointF](#) const *vertex2, [PointF](#) const *vertex3, [Color](#) color1, [Color](#) color2, [Color](#) color3, [BlendingEffect](#) effect)
Draws triangle filled with color gradient specified by given positions and colors.
- AFTERWARP_API void [CanvasQuad](#) (struct [Canvas_t](#) *canvas, [Quad](#) const *vertices, [ColorRect](#) const *colors, [BlendingEffect](#) effect)
Draws a filled quadrilateral with the given vertices and colors.
- AFTERWARP_API void [CanvasFillRect](#) (struct [Canvas_t](#) *canvas, [RectF](#) const *rect, [ColorRect](#) const *colors, [BlendingEffect](#) effect)
Draws a filled rectangle with the given vertices and colors.
- AFTERWARP_API void [CanvasFrameRect](#) (struct [Canvas_t](#) *canvas, [RectF](#) const *rect, [ColorRect](#) const *colors, float thickness, [BlendingEffect](#) effect)
Draws a rectangle of varying thickness with the given vertices and colors.
- AFTERWARP_API void [CanvasFillRoundRect](#) (struct [Canvas_t](#) *canvas, [RectF](#) const *rect, [ColorRect](#) const *colors, float radius, int32_t steps, [BlendingEffect](#) effect)
Draws a filled rectangle with the given gradient and round corners.
- AFTERWARP_API void [CanvasFrameRoundRect](#) (struct [Canvas_t](#) *canvas, [RectF](#) const *rect, [ColorRect](#) const *colors, float radius, float thickness, int32_t steps, [BlendingEffect](#) effect)

Draws a round frame for the given rectangle area with varying thickness and gradient.

- AFTERWARP_API void [CanvasFillRoundRectTop](#) (struct [Canvas_t](#) *canvas, [RectF](#) const *rect, [ColorRect](#) const *colors, float radius, int32_t steps, [BlendingEffect](#) effect)

Draws a filled rectangle with the given gradient and round corners on top.

- AFTERWARP_API void [CanvasFillRoundRectBottom](#) (struct [Canvas_t](#) *canvas, [RectF](#) const *rect, [ColorRect](#) const *colors, float radius, int32_t steps, [BlendingEffect](#) effect)

Draws a filled rectangle with the given gradient and round corners on bottom.

- AFTERWARP_API void [CanvasFillRoundRectTopInverse](#) (struct [Canvas_t](#) *canvas, [RectF](#) const *rect, [ColorRect](#) const *colors, float radius, int32_t steps, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasHighlight](#) (struct [Canvas_t](#) *canvas, [RectF](#) const *rect, float cornerRadius, float luma, float distance, int32_t steps)
- AFTERWARP_API void [CanvasHexagon](#) (struct [Canvas_t](#) *canvas, [Color](#) color1, [Color](#) color2, [Color](#) color3, [Color](#) color4, [Color](#) color5, [Color](#) color6, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasHexagonGrad](#) (struct [Canvas_t](#) *canvas, [ColorRect](#) const *colors, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasArc](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *radius, float initAngle, float endAngle, int32_t steps, [Color](#) colorInside, [Color](#) colorOutside, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasArcGrad](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *radius, float initAngle, float endAngle, int32_t steps, [ColorRect](#) const *colors, [BlendingEffect](#) effect)

Draws a filled arc with the given origin, radiuses, angles, steps and colors.

- AFTERWARP_API void [CanvasEllipse](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *radius, int32_t steps, [ColorRect](#) const *colors, [BlendingEffect](#) effect)

Draws a filled ellipse with the given origin, radiuses, steps and colors.

- AFTERWARP_API void [CanvasRibbonGrad](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *insideRadius, [PointF](#) const *outsideRadius, int32_t steps, [ColorRect](#) const *colors, [BlendingEffect](#) effect)

Draws a filled ribbon between inner and outer radiuses filled with four-color gradient.

- AFTERWARP_API void [CanvasRibbonTri](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *insideRadius, [PointF](#) const *outsideRadius, int32_t steps, [ColorPair](#) const *color1, [ColorPair](#) const *color2, [ColorPair](#) const *color3, [BlendingEffect](#) effect)

Draws a filled ribbon between inner and outer radiuses filled with continuous three-pair gradient.

- AFTERWARP_API void [CanvasRibbon](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *insideRadius, [PointF](#) const *outsideRadius, int32_t steps, [Color](#) color, [BlendingEffect](#) effect)

Draws a filled ribbon between inner and outer radiuses filled with a single color.

- AFTERWARP_API void [CanvasTapeGrad](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *insideRadius, [PointF](#) const *outsideRadius, float initAngle, float endAngle, int32_t steps, [ColorRect](#) const *colors, [BlendingEffect](#) effect)

Draws a filled tape between the given radiuses and angles, filled with four-color gradient.

- AFTERWARP_API void [CanvasTapeTri](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *insideRadius, [PointF](#) const *outsideRadius, float initAngle, float endAngle, int32_t steps, [ColorPair](#) const *color1, [ColorPair](#) const *color2, [ColorPair](#) const *color3, [BlendingEffect](#) effect)

Draws a filled tape between the given radiuses and angles, filled with continuous three-pair gradient.

- AFTERWARP_API void [CanvasTape](#) (struct [Canvas_t](#) *canvas, [PointF](#) const *origin, [PointF](#) const *insideRadius, [PointF](#) const *outsideRadius, float initAngle, float endAngle, int32_t steps, [Color](#) color, [BlendingEffect](#) effect)

Draws a filled tape between the given radiuses and angles, filled with a single color.

- AFTERWARP_API void [CanvasRectWithHole](#) (struct [Canvas_t](#) *canvas, [RectF](#) const *rect, [PointF](#) const *holeOrigin, [PointF](#) const *holeRadius, [ColorPair](#) const *colors, int32_t steps, [BlendingEffect](#) effect)

Draws a filled rectangle with a hole inside of it, filled with the given colors.

- AFTERWARP_API void [CanvasDrawBuffer](#) (struct [Canvas_t](#) *canvas, struct [CanvasBuffer_t](#) *buffer, [BlendingEffect](#) effect)

Draws triangles from an existing buffer.

- AFTERWARP_API void [CanvasTexturedTriangle](#) (struct [Canvas_t](#) *canvas, struct [Texture_t](#) *texture, [PointF](#) const *vertex1, [PointF](#) const *vertex2, [PointF](#) const *vertex3, [PointF](#) const *texCoord1, [PointF](#) const *texCoord2, [PointF](#) const *texCoord3, [Color](#) color1, [Color](#) color2, [Color](#) color3, [BlendingEffect](#) effect)

- AFTERWARP_API void [CanvasTexturedTriangleRegion](#) (struct [Canvas_t](#) *canvas, struct [Texture_t](#) *texture, [PointF](#) const *vertex1, [PointF](#) const *vertex2, [PointF](#) const *vertex3, [PointF](#) const *texCoord1, [PointF](#) const *texCoord2, [PointF](#) const *texCoord3, [Color](#) color1, [Color](#) color2, [Color](#) color3, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasTexturedQuad](#) (struct [Canvas_t](#) *canvas, struct [Texture_t](#) *texture, [Quad](#) const *vertices, [Quad](#) const *texCoords, [ColorRect](#) const *colors, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasTexturedQuadRegion](#) (struct [Canvas_t](#) *canvas, struct [Texture_t](#) *texture, [Quad](#) const *vertices, [Quad](#) const *texCoords, [ColorRect](#) const *colors, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasTexturedRoundRect](#) (struct [Canvas_t](#) *canvas, struct [Texture_t](#) *texture, [RectF](#) const *rect, [Quad](#) const *texCoords, [ColorRect](#) const *colors, float radius, int32_t steps, [BlendingEffect](#) effect)
Draws a textured rectangle with round corners and the given gradient and texture coordinates.
- AFTERWARP_API void [CanvasTexturedRoundRectRegion](#) (struct [Canvas_t](#) *canvas, struct [Texture_t](#) *texture, [RectF](#) const *rect, [Quad](#) const *texCoords, [ColorRect](#) const *colors, float radius, int32_t steps, [BlendingEffect](#) effect)
- AFTERWARP_API void [CanvasQuadImage](#) (struct [Canvas_t](#) *canvas, struct [ImageAtlas_t](#) *imageAtlas, [Quad](#) const *vertices, int32_t regionIndex, [ColorRect](#) const *colors, [RectF](#) const *sourceRect, uint32_t modifiers, [BlendingEffect](#) effect)
Draws a textured quadrilateral from source portion of image atlas, multiplied by a color gradient.
- AFTERWARP_API void [CanvasSetWorld](#) (struct [Canvas_t](#) *canvas, [Matrix](#) const *world)
Sets new 3D world transformation matrix.
- AFTERWARP_API void [CanvasGetWorld](#) (struct [Canvas_t](#) *canvas, [Matrix](#) *world)
Retrieves current 3D world transformation matrix.
- AFTERWARP_API void [CanvasSetViewProjection](#) (struct [Canvas_t](#) *canvas, [Matrix](#) const *viewProjection)
Sets new 3D view/projection transformation matrix.
- AFTERWARP_API void [CanvasGetViewProjection](#) (struct [Canvas_t](#) *canvas, [Matrix](#) *viewProjection)
Retrieves current 3D view/projection transformation matrix.
- AFTERWARP_API void [CanvasSetTransform](#) (struct [Canvas_t](#) *canvas, [Matrix3x2](#) const *transform)
Changes 2D transformation matrix.
- AFTERWARP_API void [CanvasGetTransform](#) (struct [Canvas_t](#) *canvas, [Matrix3x2](#) *transform)
Returns currently set 2D transformation matrix.
- AFTERWARP_API void [CanvasSetSignedDistanceField](#) (struct [Canvas_t](#) *canvas, [SignedDistanceField](#) const *signedDistanceField)
- AFTERWARP_API void [CanvasGetSignedDistanceField](#) (struct [Canvas_t](#) *canvas, [SignedDistanceField](#) *signedDistanceField)
Retrieves current 3D transformation (world/view/projection) matrix.
- AFTERWARP_API [CanvasContextState](#) [CanvasGetContextState](#) (struct [Canvas_t](#) *canvas)
Returns currently configured canvas context state.
- AFTERWARP_API [LibraryBool](#) [CanvasSetContextState](#) (struct [Canvas_t](#) *canvas, [CanvasContextState](#) contextState)
Configures device rendering state to one of pre-defined canvas states.
- AFTERWARP_API [LibraryBool](#) [CanvasSetSamplerState](#) (struct [Canvas_t](#) *canvas, [CanvasSamplerState](#) const *samplerState)
Sets new canvas sampler state.
- AFTERWARP_API [LibraryBool](#) [CanvasGetSamplerState](#) (struct [Canvas_t](#) *canvas, [CanvasSamplerState](#) *samplerState)
Retrieves current canvas sampler state (if such exists).
- AFTERWARP_API void [CanvasResetSamplerState](#) (struct [Canvas_t](#) *canvas)
Resets any canvas sampler state that is currently set.
- AFTERWARP_API void [CanvasSetAttributes](#) (struct [Canvas_t](#) *canvas, uint32_t attributes)
Sets new canvas rendering attributes.
- AFTERWARP_API uint32_t [CanvasGetAttributes](#) (struct [Canvas_t](#) *canvas)
Retrieves current canvas rendering attributes.
- AFTERWARP_API void [CanvasFlush](#) (struct [Canvas_t](#) *canvas)

- Flushes the canvas by rendering primitives that are still in batch buffers.*

 - AFTERWARP_API void [CanvasReset](#) (struct [Canvas_t](#) *canvas)

Clears canvas internal buffers and cached resources.

 - AFTERWARP_API int32_t [CanvasGetBatchCount](#) (struct [Canvas_t](#) *canvas)
 - AFTERWARP_API void [CanvasGetClipRect](#) (struct [Canvas_t](#) *canvas, [Rect](#) *clipRect)

Returns currently set clipping rectangle.

 - AFTERWARP_API [LibraryBool](#) [CanvasSetClipRect](#) (struct [Canvas_t](#) *canvas, [Rect](#) const *rect)

Sets new clipping rectangle.

 - AFTERWARP_API struct [ImageAtlas_t](#) * [ImageAtlasCreate](#) (struct [Device_t](#) *device)

Creates new instance of image atlas associated with a particular device.

 - AFTERWARP_API void [ImageAtlasDestroy](#) (struct [ImageAtlas_t](#) *atlas)

Releases given instance of image atlas.

 - AFTERWARP_API struct [Device_t](#) * [ImageAtlasGetDevice](#) (struct [ImageAtlas_t](#) *atlas)

Returns device associated with the given image atlas.

 - AFTERWARP_API int32_t [ImageAtlasTextureCount](#) (struct [ImageAtlas_t](#) *atlas)

Returns the number of textures contained within the atlas.

 - AFTERWARP_API struct [Texture_t](#) * [ImageAtlasTexture](#) (struct [ImageAtlas_t](#) *atlas, int32_t textureIndex)

Returns texture associated with a particular index or NULL if such is not available.

 - AFTERWARP_API int32_t [ImageAtlasRegionCount](#) (struct [ImageAtlas_t](#) *atlas)

Returns the number of regions contained within the atlas.

 - AFTERWARP_API [LibraryBool](#) [ImageAtlasRegion](#) (struct [ImageAtlas_t](#) *atlas, int32_t regionIndex, [ImageRegion](#) *region)

Returns region information for the given index.

 - AFTERWARP_API int32_t [ImageAtlasCreateRegion](#) (struct [ImageAtlas_t](#) *atlas, [Rect](#) const *rect, int32_t textureIndex)

Creates and adds a new region to the list.

 - AFTERWARP_API void [ImageAtlasRemoveRegion](#) (struct [ImageAtlas_t](#) *atlas, int32_t regionIndex)

Removes region with given index from the list.

 - AFTERWARP_API void [ImageAtlasClearRegions](#) (struct [ImageAtlas_t](#) *atlas)

Removes all regions.

 - AFTERWARP_API void [ImageAtlasMakeRegions](#) (struct [ImageAtlas_t](#) *atlas, [Point](#) const *patternSize, [Point](#) const *visibleSize, int32_t patternCount)

Creates list of regions based on repeated rectangular pattern dimensions.

 - AFTERWARP_API void [ImageAtlasClearTextures](#) (struct [ImageAtlas_t](#) *atlas)

Removes all existing textures.

 - AFTERWARP_API void [ImageAtlasRemoveTexture](#) (struct [ImageAtlas_t](#) *atlas, int32_t textureIndex)

Removes texture with the given index.

 - AFTERWARP_API int32_t [ImageAtlasCreateTexture](#) (struct [ImageAtlas_t](#) *atlas, [Point](#) const *size, [PixelFormat](#) format, [TextureAttributes](#) attributes)
 - AFTERWARP_API int32_t [ImageAtlasPackRegion](#) (struct [ImageAtlas_t](#) *atlas, [Point](#) const *size, int32_t padding)
 - AFTERWARP_API int32_t [ImageAtlasPackSurface](#) (struct [ImageAtlas_t](#) *atlas, struct [Surface_t](#) *surface, [Rect](#) const *sourceRect, int32_t padding)
 - AFTERWARP_API struct [TextRenderer_t](#) * [TextRendererCreate](#) (struct [Canvas_t](#) *canvas, [Point](#) const *textureSize, [PixelFormat](#) pixelFormat, [LibraryBool](#) mipMapping)

Creates new instance of text renderer.

 - AFTERWARP_API void [TextRendererDestroy](#) (struct [TextRenderer_t](#) *textRenderer)

Releases given instance of text renderer.

 - AFTERWARP_API struct [Canvas_t](#) * [TextRendererGetCanvas](#) (struct [TextRenderer_t](#) *textRenderer)

Returns canvas class associated with the text renderer.

 - AFTERWARP_API [LibraryBool](#) [TextRendererSetFontParameters](#) (struct [TextRenderer_t](#) *textRenderer, [FontParameters](#) const *parameters)

- Specifies new font settings for the text renderer.*
- AFTERWARP_API [LibraryBool](#) [TextRendererGetFontParameters](#) (struct [TextRenderer_t](#) *textRenderer, [FontParameters](#) *parameters)
- Retrieves current font settings from the text renderer.*
- AFTERWARP_API void [TextRendererExtent](#) (struct [TextRenderer_t](#) *textRenderer, char const *text, [PointF](#) *dimensions, [TextRenderModifiers](#) const *modifiers)
- Returns the logical dimensions that the given text will occupy when rendered.*
- AFTERWARP_API void [TextRendererExtentByPixels](#) (struct [TextRenderer_t](#) *textRenderer, char const *text, [RectF](#) *rect, [TextRenderModifiers](#) const *modifiers)
- Calculates the actual rectangle in pixels that the given text will occupy when rendered.*
- AFTERWARP_API [int32_t](#) [TextRendererRects](#) (struct [TextRenderer_t](#) *textRenderer, [PointF](#) *extent, [_Out_](#) [_opt_](#) [TextEntryRect](#) rects[], char const *text, [TextRenderModifiers](#) const *modifiers)
 - AFTERWARP_API void [TextRendererDraw](#) (struct [TextRenderer_t](#) *textRenderer, [PointF](#) const *position, char const *text, [ColorPair](#) const *colors, float alpha, [TextRenderModifiers](#) const *modifiers)
- Draws text at the given position.*
- AFTERWARP_API void [TextRendererDrawAligned](#) (struct [TextRenderer_t](#) *textRenderer, [PointF](#) const *position, char const *text, [ColorPair](#) const *colors, [TextAlignment](#) horizAlign, [TextAlignment](#) vertAlign, float alpha, [LibraryBool](#) alignToPixels, [TextRenderModifiers](#) const *modifiers)
- Draws text at the given position with specific alignment.*
- AFTERWARP_API void [TextRendererDrawCentered](#) (struct [TextRenderer_t](#) *textRenderer, [PointF](#) const *position, char const *text, [ColorPair](#) const *colors, float alpha, [LibraryBool](#) alignToPixels, [TextRenderModifiers](#) const *modifiers)
- Draws text centered around the specified position.*
- AFTERWARP_API void [TextRendererDrawAlignedByPixels](#) (struct [TextRenderer_t](#) *textRenderer, [PointF](#) const *position, char const *text, [ColorPair](#) const *colors, [TextAlignment](#) horizAlign, [TextAlignment](#) vertAlign, float alpha, [LibraryBool](#) alignToPixels, [TextRenderModifiers](#) const *modifiers)
- Draws text at the given position with specific alignment by actual visible pixels.*
- AFTERWARP_API void [TextRendererDrawCenteredByPixels](#) (struct [TextRenderer_t](#) *textRenderer, [PointF](#) const *position, char const *text, [ColorPair](#) const *colors, float alpha, [LibraryBool](#) alignToPixels, [TextRenderModifiers](#) const *modifiers)
- Draws text centered around the specified position by actual visible pixels.*
- AFTERWARP_API struct [TextModeller_t](#) * [TextModellerCreate](#) (struct [Device_t](#) *device)
- Creates new instance of vector-based text rendering module.*
- AFTERWARP_API void [TextModellerDestroy](#) (struct [TextModeller_t](#) *textModeller)
- Releases an existing instance of vector-based text rendering module.*
- AFTERWARP_API struct [Device_t](#) * [TextModellerGetDevice](#) (struct [TextModeller_t](#) *textModeller)
- Returns graphics device associated with the text rendering module.*
- AFTERWARP_API [UntypedHandle](#) [TextModellerSetFontParameters](#) (struct [TextModeller_t](#) *textModeller, [FontParameters](#) const *parameters)
 - AFTERWARP_API [UntypedHandle](#) [TextModellerGetFontProvider](#) (struct [TextModeller_t](#) *textModeller)
- Returns currently associated font provider.*
- AFTERWARP_API void [TextModellerSetFontProvider](#) (struct [TextModeller_t](#) *textModeller, [UntypedHandle](#) provider)
 - AFTERWARP_API void [TextModellerExtent](#) (struct [TextModeller_t](#) *textModeller, char const *text, [PointF](#) *extent, [TextRenderModifiers](#) const *modifiers)
- Returns the logical dimensions that the given text will occupy when rendered.*
- AFTERWARP_API void [TextModellerExtentByShape](#) (struct [TextModeller_t](#) *textModeller, char const *text, [RectF](#) *rect, [TextRenderModifiers](#) const *modifiers)
- Calculates a physical shape area that a given text occupies when rendered at zero position.*
- AFTERWARP_API [int32_t](#) [TextModellerRects](#) (struct [TextModeller_t](#) *textModeller, [PointF](#) *extent, [_Out_](#) [_opt_](#) [TextEntryRect](#) rects[], char const *text, [TextRenderModifiers](#) const *modifiers)
 - AFTERWARP_API [LibraryBool](#) [TextModellerDrawToCanvas](#) (struct [TextModeller_t](#) *textModeller, struct [Canvas_t](#) *canvas, [PointF](#) const *position, char const *text, [ColorPair](#) const *colors, [TextAlignment](#) alignHoriz, [TextAlignment](#) alignVert, [LibraryBool](#) alignByShape, [TextRenderModifiers](#) const *modifiers, [BlendingEffect](#) effect)

Draws 2D text at the given position with specific alignment directly to canvas.

- AFTERWARP_API [LibraryBool](#) [TextModellerDrawToCanvasBuffer](#) (struct [TextModeller_t](#) *textModeller, struct [CanvasBuffer_t](#) *buffer, [PointF](#) const *position, char const *text, [ColorPair](#) const *colors, [TextAlignment](#) alignHoriz, [TextAlignment](#) alignVert, [LibraryBool](#) alignByShape, [TextRenderModifiers](#) const *modifiers)

Draws 2D text at the given position with specific alignment to a canvas buffer.

- AFTERWARP_API [LibraryBool](#) [TextModellerDrawCurvedToCanvas](#) (struct [TextModeller_t](#) *textModeller, struct [Canvas_t](#) *canvas, [PointF](#) const *position, char const *text, float radius, float angle, [ColorPair](#) const *colors, [TextRenderModifiers](#) const *modifiers, [BlendingEffect](#) effect)
- AFTERWARP_API [LibraryBool](#) [TextModellerDrawCurvedToCanvasBuffer](#) (struct [TextModeller_t](#) *textModeller, struct [CanvasBuffer_t](#) *buffer, [PointF](#) const *position, char const *text, float radius, float angle, [ColorPair](#) const *colors, [TextRenderModifiers](#) const *modifiers)
- AFTERWARP_API [LibraryBool](#) [TextModellerDraw](#) (struct [TextModeller_t](#) *textModeller, [Vector](#) const *position, char const *text, float depth, [ColorPair](#) const *colors, [TextAlignment](#) alignHoriz, [TextAlignment](#) alignVert, [TextAlignment](#) alignDepth, [LibraryBool](#) alignByShape, [TextRenderModifiers](#) const *modifiers)
- AFTERWARP_API [LibraryBool](#) [TextModellerDrawCurved](#) (struct [TextModeller_t](#) *textModeller, [Vector](#) const *position, char const *text, float radius, float angle, float depth, [ColorPair](#) const *colors, [TextAlignment](#) alignDepth, [TextRenderModifiers](#) const *modifiers)
- AFTERWARP_API [LibraryBool](#) [TextModellerDrawDepthCurved](#) (struct [TextModeller_t](#) *textModeller, [Vector](#) const *position, char const *text, float radius, float angle, float depth, [ColorPair](#) const *colors, [TextRenderModifiers](#) const *modifiers)
- AFTERWARP_API void [TextModellerClear](#) (struct [TextModeller_t](#) *textModeller)

Clears GPU buffers for rendering 3D text.

- AFTERWARP_API [LibraryBool](#) [TextModellerPrepare](#) (struct [TextModeller_t](#) *textModeller)

Prepares and fills GPU buffers with an existing 3D text rendering data.

- AFTERWARP_API [LibraryBool](#) [TextModellerRender](#) (struct [TextModeller_t](#) *textModeller, struct [Program_t](#) *program, uint32_t channel)

Issues draw call with existing GPU buffers attached.

- AFTERWARP_API [LibraryBool](#) [TextModellerCopyToMeshBuffer](#) (struct [TextModeller_t](#) *textModeller, struct [MeshBuffer_t](#) *meshBuffer)

Copies the contents of existing 3D buffers into a mesh buffer.

- AFTERWARP_API void [TextModellerReset](#) (struct [TextModeller_t](#) *textModeller)

Releases existing resources, character records and buffers.

- AFTERWARP_API void [TextModellerGetTransform](#) (struct [TextModeller_t](#) *textModeller, [Matrix](#) *transform)

Returns current 3D transformation matrix.

- AFTERWARP_API void [TextModellerSetTransform](#) (struct [TextModeller_t](#) *textModeller, [Matrix](#) const *transform)

Changes current 3D transformation matrix.

- AFTERWARP_API struct [Grapher_t](#) * [GrapherCreate](#) (struct [Device_t](#) *device)

Creates new 3D plotting grapher.

- AFTERWARP_API void [GrapherDestroy](#) (struct [Grapher_t](#) *grapher)

Releases given instance of 3D plotting grapher.

- AFTERWARP_API struct [Device_t](#) * [GrapherGetDevice](#) (struct [Grapher_t](#) *grapher)

Returns device associated with the given grapher module.

- AFTERWARP_API [LibraryBool](#) [GrapherBegin](#) (struct [Grapher_t](#) *grapher)

Prepares grapher for rendering.

- AFTERWARP_API void [GrapherEnd](#) (struct [Grapher_t](#) *grapher)

Finishes grapher rendering.

- AFTERWARP_API void [GrapherPoints](#) (struct [Grapher_t](#) *grapher, [Vector4](#) const vertices[], [Color](#) const colors[], float const angles[], int32_t elementCount, [PointShape](#) shape)
- AFTERWARP_API void [GrapherLines](#) (struct [Grapher_t](#) *grapher, [Vector4](#) const vertices[], [Color](#) const colors[], int32_t const indices[], int32_t vertexCount, int32_t indexCount, [LineCaps](#) caps)

Draws lines at their given vertices, thicknesses ("w" component of each vertex), colors and indices.

- AFTERWARP_API void [GrapherPoint](#) (struct [Grapher_t](#) *grapher, [Vector](#) const *position, [Color](#) color, float size, [PointShape](#) shape, float angle)

Draws a single point with the given parameters.

- AFTERWARP_API void [GrapherLine](#) (struct [Grapher_t](#) *grapher, [Vector](#) const *position1, [Vector](#) const *position2, [Color](#) color1, [Color](#) color2, float thickness1, float thickness2, [LineCaps](#) caps)

Draws a single line with the given parameters.

- AFTERWARP_API void [GrapherArrow](#) (struct [Grapher_t](#) *grapher, [PointF](#) const *targetSize, [Vector](#) const *origin, [Vector](#) const *destination, [Color](#) color, float thickness, float size, [LineCaps](#) caps)
- AFTERWARP_API void [GrapherBoundingBox](#) (struct [Grapher_t](#) *grapher, [Matrix](#) const *volume, [Color](#) color, float thickness, float length, [LineCaps](#) caps)

Draws lines around a bounding box for the given volume matrix.

- AFTERWARP_API void [GrapherDottedLine](#) (struct [Grapher_t](#) *grapher, [Vector](#) const *position1, [Vector](#) const *position2, [Color](#) color1, [Color](#) color2, float thickness1, float thickness2, float sparsity, [PointShape](#) shape, float angle)

Draws a line of dots with the given parameters.

- AFTERWARP_API void [GrapherFlush](#) (struct [Grapher_t](#) *grapher)
- AFTERWARP_API void [GrapherReset](#) (struct [Grapher_t](#) *grapher)
- AFTERWARP_API void [GrapherGetTransform](#) (struct [Grapher_t](#) *grapher, [Matrix](#) *transform)

Returns currently set 3D transformation (world/view/projection) matrix.

- AFTERWARP_API void [GrapherSetTransform](#) (struct [Grapher_t](#) *grapher, [Matrix](#) const *transform)

Changes 3D transformation (world/view/projection) matrix.

- AFTERWARP_API int32_t [GrapherGetBatchCount](#) (struct [Grapher_t](#) *grapher)
- AFTERWARP_API struct [MeshModel_t](#) * [MeshModelCreate](#) (struct [Device_t](#) *device, int32_t vertexCount, uint32_t vertexElementPitch, int32_t indexCount, uint32_t channel, void const *initialVertexData, void const *initialIndexData)

Creates a new instance of 3D mesh model.

- AFTERWARP_API void [MeshModelDestroy](#) (struct [MeshModel_t](#) *meshModel)

Releases given instance of 3D mesh model.

- AFTERWARP_API [LibraryBool](#) [MeshModelRenderable](#) (struct [MeshModel_t](#) *meshModel)
- AFTERWARP_API void [MeshModelDismantle](#) (struct [MeshModel_t](#) *meshModel)
- AFTERWARP_API void [MeshModelTakeAway](#) (struct [MeshModel_t](#) *meshModel, struct [MeshModel_t](#) *meshModelAnother)

Takes away the internal contents from another 3D mesh model container without doing any copying.

- AFTERWARP_API int32_t [MeshModelGetVertexCount](#) (struct [MeshModel_t](#) *meshModel)

Returns total number of vertices stored in the model.

- AFTERWARP_API int32_t [MeshModelGetIndexCount](#) (struct [MeshModel_t](#) *meshModel)

Returns total number of indices stored in the model.

- AFTERWARP_API uint32_t [MeshModelGetChannel](#) (struct [MeshModel_t](#) *meshModel)

Returns channel associated with vertex buffer format.

- AFTERWARP_API struct [Buffer_t](#) * [MeshModelGetVertexBuffer](#) (struct [MeshModel_t](#) *meshModel)

Returns pointer to integrated vertex buffer.

- AFTERWARP_API struct [Buffer_t](#) * [MeshModelGetIndexBuffer](#) (struct [MeshModel_t](#) *meshModel)

Returns pointer to integrated index buffer.

- AFTERWARP_API [LibraryBool](#) [MeshModelDraw](#) (struct [MeshModel_t](#) *meshModel, struct [Program_t](#) *program, [PrimitiveTopology](#) topology, int32_t elementCount, int32_t firstIndex, int32_t baseVertex, [LibraryBool](#) postUnbind)

Renders the mesh model with the given program.

- AFTERWARP_API [LibraryBool](#) [MeshModelDrawInstances](#) (struct [MeshModel_t](#) *meshModel, struct [Program_t](#) *program, int32_t instanceCount, [PrimitiveTopology](#) topology, int32_t elementCount, int32_t firstIndex, int32_t baseVertex, [LibraryBool](#) postUnbind)

Renders multiple instances of the mesh model with a given program.

- AFTERWARP_API struct [MeshMetaTags_t](#) * [MeshMetaTagGetParent](#) (struct [MeshMetaTag_t](#) *tag)

Returns mesh meta tag's parent container.

- AFTERWARP_API void [MeshMetaTagGetName](#) (struct [MeshMetaTag_t](#) *tag, char *name, _Inout_ int32_t *nameLength)

- AFTERWARP_API uint8_t [MeshMetaTagGetType](#) (struct [MeshMetaTag_t](#) *tag)
Returns type of the tag.
- AFTERWARP_API void [MeshMetaTagGetBounds](#) (struct [MeshMetaTag_t](#) *tag, [Vector](#) *boundsMin, [Vector](#) *boundsMax)
Calculates and returns boundaries for all existing tag's portions, if such exist.
- AFTERWARP_API int32_t [MeshMetaTagGetPortionCount](#) (struct [MeshMetaTag_t](#) *tag)
Returns number of existing portions associated with the tag.
- AFTERWARP_API [LibraryBool](#) [MeshMetaTagPortionGet](#) (struct [MeshMetaTag_t](#) *tag, [MeshMetaTagPortion](#) *portion, int32_t index)
Returns a portion that corresponds to the tag with a given index.
- AFTERWARP_API [LibraryBool](#) [MeshMetaTagPortionAdd](#) (struct [MeshMetaTag_t](#) *tag, [MeshMetaTagPortion](#) const *portion)
Adds a new portion to the tag.
- AFTERWARP_API void [MeshMetaTagPortionErase](#) (struct [MeshMetaTag_t](#) *tag, int32_t index)
Erases a portion with the given index.
- AFTERWARP_API void [MeshMetaTagPortionsClear](#) (struct [MeshMetaTag_t](#) *tag)
Removes all existing portions.
- AFTERWARP_API [LibraryBool](#) [MeshMetaTagPortionsCopy](#) (struct [MeshMetaTag_t](#) *tag, struct [MeshMetaTag_t](#) *source)
Removes any existing portions and then copies them from an existing tag.
- AFTERWARP_API struct [MeshMetaTags_t](#) * [MeshMetaTagsCreate](#) (void)
Creates a new instance of 3D mesh tag container.
- AFTERWARP_API void [MeshMetaTagsDestroy](#) (struct [MeshMetaTags_t](#) *tags)
Releases the given 3D mesh tag container instance.
- AFTERWARP_API int32_t [MeshMetaTagsCount](#) (struct [MeshMetaTags_t](#) *tags)
Returns number of existing tags.
- AFTERWARP_API struct [MeshMetaTag_t](#) * [MeshMetaTagsGetByIndex](#) (struct [MeshMetaTags_t](#) *tags, int32_t index)
Returns a particular tag with the given index.
- AFTERWARP_API struct [MeshMetaTag_t](#) * [MeshMetaTagsGetByName](#) (struct [MeshMetaTags_t](#) *tags, char const *name, uint8_t type)
Returns a particular tag with the given name (case-sensitive) and type.
- AFTERWARP_API struct [MeshMetaTag_t](#) * [MeshMetaTagsAdd](#) (struct [MeshMetaTags_t](#) *tags, char const *name, uint8_t type)
Creates a new tag with the given name (case-sensitive) and type.
- AFTERWARP_API void [MeshMetaTagsErase](#) (struct [MeshMetaTags_t](#) *tags, int32_t index)
Removes an existing tag from container list.
- AFTERWARP_API void [MeshMetaTagsClear](#) (struct [MeshMetaTags_t](#) *tags)
Removes all existing tags from container list.
- AFTERWARP_API [LibraryBool](#) [MeshMetaTagsCopy](#) (struct [MeshMetaTags_t](#) *tags, struct [MeshMetaTags_t](#) *tagsAnother)
Copies tags from another container.
- AFTERWARP_API void [MeshMetaTagsTakeAway](#) (struct [MeshMetaTags_t](#) *tags, struct [MeshMetaTags_t](#) *tagsAnother)
Takes away the internal contents from another tags container without doing any copying.
- AFTERWARP_API struct [MeshBuffer_t](#) * [MeshBufferCreate](#) (void)
Creates a new instance of 3D mesh buffer.
- AFTERWARP_API void [MeshBufferDestroy](#) (struct [MeshBuffer_t](#) *meshBuffer)
Releases given instance of 3D mesh buffer.
- AFTERWARP_API [MeshBufferEntry](#) * [MeshBufferGetVertices](#) (struct [MeshBuffer_t](#) *meshBuffer)
Returns pointer to vertices stored in 3D mesh buffer.
- AFTERWARP_API int32_t [MeshBufferGetVertexCount](#) (struct [MeshBuffer_t](#) *meshBuffer)

- Returns total number of vertices in 3D mesh buffer.*
- AFTERWARP_API [LibraryBool](#) [MeshBufferSetVertexCount](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t vertexCount)
- Resizes 3D mesh buffer to accomodate new number of vertices.*
- AFTERWARP_API int32_t * [MeshBufferGetIndices](#) (struct [MeshBuffer_t](#) *meshBuffer)
- Returns pointer to indices stored in the 3D mesh buffer.*
- AFTERWARP_API int32_t [MeshBufferGetIndexCount](#) (struct [MeshBuffer_t](#) *meshBuffer)
- Returns total number of indices in 3D mesh buffer.*
- AFTERWARP_API [LibraryBool](#) [MeshBufferSetIndexCount](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t indexCount)
- Resizes 3D mesh buffer to accomodate new number of indices.*
- AFTERWARP_API struct [MeshModel_t](#) * [MeshBufferCreateModel](#) (struct [MeshBuffer_t](#) *meshBuffer, struct [Device_t](#) *device, [VertexElement](#) const vertexElements[], int32_t vertexElementCount, int32_t firstVertex, int32_t vertexCount, int32_t firstIndex, int32_t indexCount, uint32_t channel)
- Creates new instance of 3D mesh model with the given vertex elements and actual data from this buffer.*
- AFTERWARP_API [LibraryBool](#) [MeshBufferSuperEllipse](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t longitudeSections, int32_t latitudeSections, [Vector](#) const *origin, [Vector](#) const *radius, float initLongitudeAngle, float endLongitudeAngle, float longitudeShape, float initLatitudeAngle, float endLatitudeAngle, float latitudeShape, [PointF](#) const *initTexCoord, [PointF](#) const *endTexCoord, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- AFTERWARP_API [LibraryBool](#) [MeshBufferGeosphere](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t subdivisions, [FloatColor](#) const *color, [LibraryBool](#) flatNormals, [LibraryBool](#) indicesClockwise)
- Creates a 3D geosphere with the given parameters.*
- AFTERWARP_API [LibraryBool](#) [MeshBufferCylinder](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t radialSections, int32_t heightSections, [Vector](#) const *origin, [Vector](#) const *horizAxis, [Vector](#) const *vertAxis, [Vector](#) const *heightAxis, float initAngle, float endAngle, float shape, [PointF](#) const *initTexCoord, [PointF](#) const *endTexCoord, float bendAngle, float lateralAngle, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- AFTERWARP_API [LibraryBool](#) [MeshBufferConeSimple](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t sections, [Vector](#) const *origin, float radius, float height, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- Creates a simplistic cone with the given parameters in XZ plane with its base at origin.*
- AFTERWARP_API [LibraryBool](#) [MeshBufferDisc](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t radialSections, int32_t innerSections, [Vector](#) const *origin, [Vector](#) const *horizAxis, [Vector](#) const *vertAxis, [Vector](#) const *normal, float initAngle, float endAngle, float shape, float initRadius, [PointF](#) const *initTexCoord, [PointF](#) const *endTexCoord, float lateralAngle, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- AFTERWARP_API [LibraryBool](#) [MeshBufferPlane](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t horizSections, int32_t vertSections, [Vector](#) const *origin, [Vector](#) const *horizAxis, [Vector](#) const *vertAxis, [Vector](#) const *normal, [PointF](#) const *initTexCoord, [PointF](#) const *endTexCoord, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- Creates a 3D plane with the given parameters. Axis vectors define both orientation and size.*
- AFTERWARP_API [LibraryBool](#) [MeshBufferCube](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t horizSections, int32_t vertSections, int32_t depthSections, [Vector](#) const *origin, [Vector](#) const *horizAxis, [Vector](#) const *vertAxis, [Vector](#) const *depthAxis, [PointF](#) const *initTexCoord, [PointF](#) const *endTexCoord, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- Creates a 3D cube with the given parameters. Axis vectors define both orientation and size.*
- AFTERWARP_API [LibraryBool](#) [MeshBufferCubeMinimal](#) (struct [MeshBuffer_t](#) *meshBuffer, [Vector](#) const *origin, [Vector](#) const *size, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- AFTERWARP_API [LibraryBool](#) [MeshBufferCubeRound](#) (struct [MeshBuffer_t](#) *meshBuffer, [Vector](#) const *origin, [Vector](#) const *size, float roundness, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- Creates a 3D cube with round corners.*
- AFTERWARP_API [LibraryBool](#) [MeshBufferTorus](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t outerSections, int32_t innerSections, [Vector](#) const *origin, [Vector](#) const *horizAxis, [Vector](#) const *vertAxis, float initOuterAngle, float endOuterAngle, float outerShape, float initInnerAngle, float endInnerAngle, float innerShape, [PointF](#) const *innerRadius, [PointF](#) const *initTexCoord, [PointF](#) const *endTexCoord, float lateralAngle, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)

- AFTERWARP_API [LibraryBool MeshBufferTorusKnot](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t outerSections, int32_t innerSections, [Vector](#) const *origin, int32_t p, int32_t q, float initOuterAngle, float endOuterAngle, float outerRadius, float initInnerAngle, float endInnerAngle, float innerShape, [PointF](#) const *innerRadius, [PointF](#) const *initTexCoord, [PointF](#) const *endTexCoord, float lateralAngle, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- AFTERWARP_API [LibraryBool MeshBufferSupertoroid](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t outerSections, int32_t innerSections, [Vector](#) const *origin, float initOuterAngle, float endOuterAngle, float outerRadius, float outerShape, float initInnerAngle, float endInnerAngle, float innerRadius, float innerShape, [PointF](#) const *initTexCoord, [PointF](#) const *endTexCoord, [FloatColor](#) const *color, [LibraryBool](#) indicesClockwise)
- AFTERWARP_API [LibraryBool MeshBufferFrustumVolume](#) (struct [MeshBuffer_t](#) *meshBuffer, [Matrix](#) const *viewProjectionInverse, [LibraryBool](#) depthClipNegative)
Creates a 3D volume of a view frustum from the given view/projection matrix.
- AFTERWARP_API [LibraryBool MeshBufferExtrusion](#) (struct [MeshBuffer_t](#) *meshBuffer, [PathElement](#) const path[], int32_t pathLength, float depth, [PointF](#) const *texCoords, [FloatColor](#) const colors[], [LibraryBool](#) quality)
- AFTERWARP_API void [MeshBufferTransformVertices](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t firstVertex, int32_t vertexCount)
Applies current transformation to the vertices.
- AFTERWARP_API [LibraryBool MeshBufferCalculateFlatNormals](#) (struct [MeshBuffer_t](#) *meshBuffer, float epsilon)
- AFTERWARP_API void [MeshBufferCalculateNormals](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t firstVertex, int32_t vertexCount, int32_t firstIndex, int32_t indexCount)
- AFTERWARP_API void [MeshBufferCalculateNormalsWeld](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t firstVertex, int32_t vertexCount, int32_t firstIndex, int32_t indexCount, float weldEpsilon)
- AFTERWARP_API void [MeshBufferInvertNormals](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t firstVertex, int32_t vertexCount)
- AFTERWARP_API void [MeshBufferInvertIndexOrder](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t firstIndex, int32_t indexCount)
Inverts order of indices (e.g. from clockwise to counter-clockwise and vice-versa).
- AFTERWARP_API void [MeshBufferCalculateBounds](#) (struct [MeshBuffer_t](#) *meshBuffer, [Vector](#) *vertexMin, [Vector](#) *vertexMax, int32_t firstVertex, int32_t vertexCount)
Calculates minimum and maximum vertex coordinate boundaries in the given range of mesh.
- AFTERWARP_API void [MeshBufferCentralize](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t firstVertex, int32_t vertexCount)
Calculates coordinate boundaries and centralizes the vertices in the given range of mesh.
- AFTERWARP_API void [MeshBufferEliminateUnusedVertices](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t firstVertex, int32_t vertexCount)
- AFTERWARP_API [LibraryBool MeshBufferJoinDuplicateVertices](#) (struct [MeshBuffer_t](#) *meshBuffer, int32_t firstVertex, int32_t vertexCount, float threshold)
- AFTERWARP_API [LibraryBool MeshBufferCombine](#) (struct [MeshBuffer_t](#) *meshBuffer, struct [MeshBuffer_t](#) *meshBufferSource, int32_t firstVertex, int32_t vertexCount, int32_t firstIndex, int32_t indexCount)
Includes vertices and indices from source mesh buffer into the current one.
- AFTERWARP_API void [MeshBufferClear](#) (struct [MeshBuffer_t](#) *meshBuffer)
Removes all vertices and indices in the mesh buffer.
- AFTERWARP_API void [MeshBufferSetTransform](#) (struct [MeshBuffer_t](#) *meshBuffer, [Matrix](#) const *transform)
Sets new transformation matrix that is applied to generated vertices during construction.
- AFTERWARP_API void [MeshBufferGetTransform](#) (struct [MeshBuffer_t](#) *meshBuffer, [Matrix](#) *transform)
Returns currently set transformation matrix.
- AFTERWARP_API uint32_t [MeshBufferTransferVertexElements](#) (struct [MeshBuffer_t](#) *meshBuffer, void *buffer, [VertexElement](#) const vertexElements[], int32_t vertexElementCount, int32_t firstVertex, int32_t vertexCount, uint32_t channel, uint32_t semanticIndex)
- AFTERWARP_API uint32_t [MeshBufferTransferIndexElements](#) (struct [MeshBuffer_t](#) *meshBuffer, void *buffer, uint32_t pitch, int32_t firstVertex, int32_t firstIndex, int32_t indexCount)

- AFTERWARP_API [LibraryBool MeshBufferTransferVertices](#) (struct [MeshBuffer_t](#) *meshBuffer, struct [Buffer_t](#) *buffer, [VertexElement](#) const vertexElements[], int32_t vertexElementCount, int32_t firstVertex, int32_t vertexCount, uint32_t channel, uint32_t offset, uint32_t semanticIndex)
- AFTERWARP_API [LibraryBool MeshBufferTransferIndices](#) (struct [MeshBuffer_t](#) *meshBuffer, struct [Buffer_t](#) *buffer, int32_t firstVertex, int32_t firstIndex, int32_t indexCount, uint32_t offset)
Transfers mesh index elements to an index buffer object with the given format at the specified offset.
- AFTERWARP_API struct [MeshVoxel_t](#) * [MeshBufferVoxelize](#) (struct [MeshBuffer_t](#) *meshBuffer, uint8_t levels, [LibraryBool](#) colors)
- AFTERWARP_API [LibraryBool MeshBufferLoadFromFile](#) (struct [MeshBuffer_t](#) *meshBuffer, char const *fileName, struct [SceneMeshMaterials_t](#) *materials, struct [MeshMetaTags_t](#) *tags, _Inout_ uint32_t *options, [MeshLoadSaveFeedback](#) feedback, void *feedbackUser, char *debug, _Inout_ int32_t *debugLength)
- AFTERWARP_API [LibraryBool MeshBufferSaveToFile](#) (struct [MeshBuffer_t](#) *meshBuffer, char const *fileName, struct [SceneMeshMaterials_t](#) *materials, struct [MeshMetaTags_t](#) *tags, uint32_t options, [MeshLoadSaveFeedback](#) feedback, void *feedbackUser, char *debug, _Inout_ int32_t *debugLength)
- AFTERWARP_API [LibraryBool MeshBufferLoadFromFileEx](#) (struct [MeshBuffer_t](#) *meshBuffer, struct [MeshMetaTags_t](#) *tags, char const *fileName, [Vector](#) *minBounds, [Vector](#) *maxBounds, [LibraryBool](#) normals, char *debug, _Inout_ int32_t *debugLength)
- AFTERWARP_API [LibraryBool MeshBufferSaveToFileEx](#) (struct [MeshBuffer_t](#) *meshBuffer, char const *fileName, uint32_t const *options, char *debug, _Inout_ int32_t *debugLength)
- AFTERWARP_API struct [GaussianBlur_t](#) * [GaussianBlurCreate](#) (struct [Device_t](#) *device, float sigma, int32_t samples, [LibraryBool](#) hardwareFiltering)
Creates a new instance of gaussian blur module.
- AFTERWARP_API void [GaussianBlurDestroy](#) (struct [GaussianBlur_t](#) *gaussianBlur)
Releases given instance of gaussian blur module.
- AFTERWARP_API struct [Device_t](#) * [GaussianBlurGetDevice](#) (struct [GaussianBlur_t](#) *gaussianBlur)
Returns device associated with the given gaussian blur.
- AFTERWARP_API int32_t [GaussianBlurGetFixedSamples](#) (struct [GaussianBlur_t](#) *gaussianBlur)
Returns maximum number of samples supported by a fixed-sample blur, which enables higher performance.
- AFTERWARP_API int32_t [GaussianBlurGetSamples](#) (struct [GaussianBlur_t](#) *gaussianBlur)
Returns current number of samples in the kernel.
- AFTERWARP_API void [GaussianBlurSetSamples](#) (struct [GaussianBlur_t](#) *gaussianBlur, int32_t samples)
- AFTERWARP_API float [GaussianBlurGetSigma](#) (struct [GaussianBlur_t](#) *gaussianBlur)
Returns current value of sigma.
- AFTERWARP_API void [GaussianBlurSetSigma](#) (struct [GaussianBlur_t](#) *gaussianBlur, float sigma)
Sets new value of sigma. The value must be equal or higher than zero.
- AFTERWARP_API float [GaussianBlurGetChroma](#) (struct [GaussianBlur_t](#) *gaussianBlur)
Returns chroma offset for color adjustment.
- AFTERWARP_API void [GaussianBlurSetChroma](#) (struct [GaussianBlur_t](#) *gaussianBlur, float chroma)
Changes chroma offset in range of [0, inf) for color adjustment (1 means no adjustment is needed).
- AFTERWARP_API [LibraryBool GaussianBlurGetHardwareFiltering](#) (struct [GaussianBlur_t](#) *gaussianBlur)
Indicates whether hardware filtering is used to accelerate the blur.
- AFTERWARP_API void [GaussianBlurSetHardwareFiltering](#) (struct [GaussianBlur_t](#) *gaussianBlur, [LibraryBool](#) hardwareFiltering)
Adjusts whether hardware filtering should be used to accelerate the blur.
- AFTERWARP_API [LibraryBool GaussianBlurUpdate](#) (struct [GaussianBlur_t](#) *gaussianBlur, struct [Texture_t](#) *destination, struct [Texture_t](#) *intermediary, struct [Texture_t](#) *source)
- AFTERWARP_API [LibraryBool GaussianBlurUpdateAt](#) (struct [GaussianBlur_t](#) *gaussianBlur, struct [Texture_t](#) *destination, struct [Texture_t](#) *intermediary, struct [Texture_t](#) *source, [Point](#) const *destPos, [Rect](#) const *intermRect)
- AFTERWARP_API struct [KawaseBlur_t](#) * [KawaseBlurCreate](#) (struct [Device_t](#) *device)
Creates a new instance of kawase blur module.
- AFTERWARP_API void [KawaseBlurDestroy](#) (struct [KawaseBlur_t](#) *kawaseBlur)

- Releases given instance of kawase blur module.*

 - AFTERWARP_API struct Device_t * [KawaseBlurGetDevice](#) (struct [KawaseBlur_t](#) *kawaseBlur)

Returns device associated with the given kawase blur.
- AFTERWARP_API int32_t [KawaseBlurGetPasses](#) (struct [KawaseBlur_t](#) *kawaseBlur)

Returns number of blur passes (each pass includes two processing steps).
- AFTERWARP_API void [KawaseBlurSetPasses](#) (struct [KawaseBlur_t](#) *kawaseBlur, int32_t passes)

Updates number of blur passes (each pass includes two processing steps).
- AFTERWARP_API void [KawaseBlurGetOffset](#) (struct [KawaseBlur_t](#) *kawaseBlur, [PointF](#) *offset)

Returns pixel offset for each individual step.
- AFTERWARP_API void [KawaseBlurSetOffset](#) (struct [KawaseBlur_t](#) *kawaseBlur, [PointF](#) const *offset)

Changes pixel offset for each individual step.
- AFTERWARP_API [LibraryBool](#) [KawaseBlurUpdate](#) (struct [KawaseBlur_t](#) *kawaseBlur, struct [Texture_t](#) *destination, struct [Texture_t](#) *intermediary, struct [Texture_t](#) *source)
- AFTERWARP_API [LibraryBool](#) [KawaseBlurSinglePass](#) (struct [KawaseBlur_t](#) *kawaseBlur, struct [Texture_t](#) *destination, struct [Texture_t](#) *source, [PointF](#) const *offset)
- AFTERWARP_API struct [ColorDithering_t](#) * [ColorDitheringCreate](#) (struct Device_t *device)

Creates a new instance of color dithering module.
- AFTERWARP_API void [ColorDitheringDestroy](#) (struct [ColorDithering_t](#) *colorDithering)

Releases given instance of color dithering module.
- AFTERWARP_API struct Device_t * [ColorDitheringGetDevice](#) (struct [ColorDithering_t](#) *colorDithering)

Returns device associated with the given module.
- AFTERWARP_API [ColorDitheringFormat](#) [ColorDitheringGetFormat](#) (struct [ColorDithering_t](#) *colorDithering)

Returns target color quantization format for dithering.
- AFTERWARP_API void [ColorDitheringSetFormat](#) (struct [ColorDithering_t](#) *colorDithering, [ColorDitheringFormat](#) format)

Updates target color quantization format for dithering.
- AFTERWARP_API [LibraryBool](#) [ColorDitheringExecuteTo](#) (struct [ColorDithering_t](#) *colorDithering, struct [Texture_t](#) *destination, struct [Texture_t](#) *source)

Performs color dithering of the source texture rendering to destination texture.
- AFTERWARP_API [LibraryBool](#) [ColorDitheringExecute](#) (struct [ColorDithering_t](#) *colorDithering, struct [Texture_t](#) *source)

Performs color dithering of the source texture rendering a full-screen quad.
- AFTERWARP_API struct [SelectionHighlight_t](#) * [SelectionHighlightCreate](#) (struct Device_t *device, [Point](#) const *size, [PixelFormat](#) depthStencil, int32_t samples, [LibraryBool](#) grayscale)

Creates a new instance of selection highlight module.
- AFTERWARP_API void [SelectionHighlightDestroy](#) (struct [SelectionHighlight_t](#) *selectionHighlight)

Releases given instance of selection highlight module.
- AFTERWARP_API struct Device_t * [SelectionHighlightGetDevice](#) (struct [SelectionHighlight_t](#) *selectionHighlight)

Returns device associated with the given module.
- AFTERWARP_API [LibraryBool](#) [SelectionHighlightSetSize](#) (struct [SelectionHighlight_t](#) *selectionHighlight, [Point](#) const *size)

Updates size of the rendering surfaces.
- AFTERWARP_API void [SelectionHighlightGetSize](#) (struct [SelectionHighlight_t](#) *selectionHighlight, [Point](#) *size)

Returns size of the rendering surfaces.
- AFTERWARP_API [PixelFormat](#) [SelectionHighlightGetDepthStencil](#) (struct [SelectionHighlight_t](#) *selectionHighlight)

Returns pixel format used for depth/stencil buffer.
- AFTERWARP_API int32_t [SelectionHighlightGetSamples](#) (struct [SelectionHighlight_t](#) *selectionHighlight)

Returns number of samples used for multisampling.

- AFTERWARP_API [LibraryBool](#) [SelectionHighlightGetGrayscale](#) (struct [SelectionHighlight_t](#) *selectionHighlight)
Returns whether the technique is performed in a more optimal, grayscale mode, instead of full color.
- AFTERWARP_API void [SelectionHighlightGetParameters](#) (struct [SelectionHighlight_t](#) *selectionHighlight, [SelectionHighlightParameters](#) *parameters)
Returns parameters that define the behavior and characteristics of the technique.
- AFTERWARP_API [LibraryBool](#) [SelectionHighlightSetParameters](#) (struct [SelectionHighlight_t](#) *selectionHighlight, [SelectionHighlightParameters](#) const *parameters)
Updates parameters that define the behavior and characteristics of the technique.
- AFTERWARP_API [LibraryBool](#) [SelectionHighlightClear](#) (struct [SelectionHighlight_t](#) *selectionHighlight)
Clears the primary rendering textures.
- AFTERWARP_API [LibraryBool](#) [SelectionHighlightBegin](#) (struct [SelectionHighlight_t](#) *selectionHighlight)
Begins rendering the primary textures.
- AFTERWARP_API void [SelectionHighlightEnd](#) (struct [SelectionHighlight_t](#) *selectionHighlight)
Ends rendering to the primary textures.
- AFTERWARP_API [LibraryBool](#) [SelectionHighlightRendering](#) (struct [SelectionHighlight_t](#) *selectionHighlight)
Indicates that the rendering is currently performed to the primary textures.
- AFTERWARP_API [LibraryBool](#) [SelectionHighlightFilter](#) (struct [SelectionHighlight_t](#) *selectionHighlight)
Performs final post-filtering of the textures to produce the highlight selection effect.
- AFTERWARP_API struct [Texture_t](#) * [SelectionHighlightGetTexture](#) (struct [SelectionHighlight_t](#) *selectionHighlight, [SelectionHighlightTextureType](#) type)
Returns texture of the given type.
- AFTERWARP_API void [SelectionHighlightGetTextureCoordinates](#) (struct [SelectionHighlight_t](#) *selectionHighlight, [Quad](#) *coords)
Returns texture coordinates that can be passed to canvas for rendering the highlight.
- AFTERWARP_API struct [SpatialFog_t](#) * [SpatialFogCreate](#) (struct [Device_t](#) *device)
Creates a new instance of spatial fog module.
- AFTERWARP_API void [SpatialFogDestroy](#) (struct [SpatialFog_t](#) *spatialFog)
Releases given instance of spatial fog module.
- AFTERWARP_API struct [Device_t](#) * [SpatialFogGetDevice](#) (struct [SpatialFog_t](#) *spatialFog)
Returns device associated with the given module.
- AFTERWARP_API void [SpatialFogGetParameters](#) (struct [SpatialFog_t](#) *spatialFog, [FogParameters](#) *parameters)
Returns parameters that define spatial fog characteristics.
- AFTERWARP_API [LibraryBool](#) [SpatialFogSetParameters](#) (struct [SpatialFog_t](#) *spatialFog, [FogParameters](#) const *parameters)
Changes parameters that define spatial fog characteristics.
- AFTERWARP_API void [SpatialFogGetView](#) (struct [SpatialFog_t](#) *spatialFog, [Matrix](#) *view)
Returns view transformation matrix.
- AFTERWARP_API [LibraryBool](#) [SpatialFogSetView](#) (struct [SpatialFog_t](#) *spatialFog, [Matrix](#) const *view)
Updates view transformation matrix.
- AFTERWARP_API void [SpatialFogGetProjection](#) (struct [SpatialFog_t](#) *spatialFog, [Matrix](#) *projection)
Returns projection transformation matrix.
- AFTERWARP_API [LibraryBool](#) [SpatialFogSetProjection](#) (struct [SpatialFog_t](#) *spatialFog, [Matrix](#) const *projection)
Updates projection transformation matrix.
- AFTERWARP_API [FogFormula](#) [SpatialFogGetFormula](#) (struct [SpatialFog_t](#) *spatialFog)
Returns formula used for the fog.
- AFTERWARP_API [LibraryBool](#) [SpatialFogSetFormula](#) (struct [SpatialFog_t](#) *spatialFog, [FogFormula](#) formula)
Changes formula used for the fog.
- AFTERWARP_API [DepthFogDistance](#) [SpatialFogGetDepthDistance](#) (struct [SpatialFog_t](#) *spatialFog)
Returns distance calculation formula for the fog.

- AFTERWARP_API [LibraryBool SpatialFogSetDepthDistance](#) (struct [SpatialFog_t](#) *spatialFog, [DepthFogDistance](#) depthDistance)
Changes distance calculation formula for the fog.
- AFTERWARP_API [LibraryBool SpatialFogExecute](#) (struct [SpatialFog_t](#) *spatialFog, struct [Texture_t](#) *destination, struct [Texture_t](#) *linearDepths)
- AFTERWARP_API [LibraryBool SpatialFogExecuteGlassy](#) (struct [SpatialFog_t](#) *spatialFog, struct [Texture_t](#) *destination, struct [Texture_t](#) *linearDepths)
- AFTERWARP_API struct [SceneLights_t](#) * [SceneLightsCreate](#) (struct [Device_t](#) *device)
Creates a new instance of 3D scene light module.
- AFTERWARP_API void [SceneLightsDestroy](#) (struct [SceneLights_t](#) *sceneLights)
Releases given instance of 3D scene light module.
- AFTERWARP_API struct [Device_t](#) * [SceneLightsGetDevice](#) (struct [SceneLights_t](#) *sceneLights)
Returns device associated with 3D scene light module.
- AFTERWARP_API int32_t [SceneLightsGetCount](#) (struct [SceneLights_t](#) *sceneLights)
Returns total number of light sources.
- AFTERWARP_API [SceneLight](#) * [SceneLightsGetElement](#) (struct [SceneLights_t](#) *sceneLights, int32_t index)
Returns pointer to an existing 3D scene light parameters.
- AFTERWARP_API [SceneLight](#) * [SceneLightsAdd](#) (struct [SceneLights_t](#) *sceneLights)
Adds a new light with default parameters.
- AFTERWARP_API [LibraryBool SceneLightsErase](#) (struct [SceneLights_t](#) *sceneLights, int32_t index)
Erases light with the given index.
- AFTERWARP_API void [SceneLightsClear](#) (struct [SceneLights_t](#) *sceneLights)
Removes all existing lights.
- AFTERWARP_API void [SceneLightsGetViewSize](#) (struct [SceneLights_t](#) *sceneLights, [Point](#) *viewSize)
Returns size of the viewable area.
- AFTERWARP_API void [SceneLightsSetViewSize](#) (struct [SceneLights_t](#) *sceneLights, [Point](#) const *viewSize)
Returns size of the viewable area.
- AFTERWARP_API int32_t [SceneLightsGetClusterSize](#) (struct [SceneLights_t](#) *sceneLights)
Returns size of individual lighting clusters.
- AFTERWARP_API void [SceneLightsSetClusterSize](#) (struct [SceneLights_t](#) *sceneLights, int32_t clusterSize)
Changes size of individual lighting clusters.
- AFTERWARP_API int32_t [SceneLightsGetDepthSlices](#) (struct [SceneLights_t](#) *sceneLights)
Returns number of cluster slices per viewable depth.
- AFTERWARP_API void [SceneLightsSetDepthSlices](#) (struct [SceneLights_t](#) *sceneLights, int32_t depthSlices)
Updates number of cluster slices per viewable depth.
- AFTERWARP_API [ClustersCullingMode](#) [SceneLightsGetCullingMode](#) (struct [SceneLights_t](#) *sceneLights)
Returns mode used for packing culled lights in clusters.
- AFTERWARP_API void [SceneLightsSetCullingMode](#) (struct [SceneLights_t](#) *sceneLights, [ClustersCullingMode](#) mode)
Changes mode used for packing culled lights in clusters.
- AFTERWARP_API void [SceneLightsGetClusters](#) (struct [SceneLights_t](#) *sceneLights, [Point](#) *clusters)
Returns number of clusters per width and height of viewable area.
- AFTERWARP_API struct [Buffer_t](#) * [SceneLightsGetIndices](#) (struct [SceneLights_t](#) *sceneLights)
Returns light indices that have been calculated for each of the clusters.
- AFTERWARP_API [LibraryBool SceneLightsExecute](#) (struct [SceneLights_t](#) *sceneLights, [Matrix](#) const *view, [Matrix](#) const *projection)
- AFTERWARP_API [LibraryBool SceneLightsRenderDebug](#) (struct [SceneLights_t](#) *sceneLights, [LibraryBool](#) intensity)
Renders light culling debug information. A call to [SceneLightsExecute\(\)](#) must be performed before this.
- AFTERWARP_API struct [ObjectMaterials_t](#) * [ObjectMaterialsCreate](#) (void)
Creates a new instance of 3D object materials container.
- AFTERWARP_API void [ObjectMaterialsDestroy](#) (struct [ObjectMaterials_t](#) *objectMaterials)

- Releases given instance of 3D object materials container.*

 - AFTERWARP_API int32_t [ObjectMaterialsGetCount](#) (struct [ObjectMaterials_t](#) *objectMaterials)

Returns total number of materials.
- AFTERWARP_API [ObjectMaterial](#) * [ObjectMaterialsGetElement](#) (struct [ObjectMaterials_t](#) *objectMaterials, int32_t index)

Returns pointer to an existing 3D object material parameters.
- AFTERWARP_API [ObjectMaterial](#) * [ObjectMaterialsAdd](#) (struct [ObjectMaterials_t](#) *objectMaterials)

Adds a new material with default parameters.
- AFTERWARP_API [LibraryBool](#) [ObjectMaterialsErase](#) (struct [ObjectMaterials_t](#) *objectMaterials, int32_t index)

Erases material with the given index.
- AFTERWARP_API void [ObjectMaterialsClear](#) (struct [ObjectMaterials_t](#) *objectMaterials)

Removes all existing materials.
- AFTERWARP_API struct [TextureCabinet_t](#) * [TextureCabinetCreate](#) (struct [Device_t](#) *device, [Point](#) const *size, [TextureFidelity](#) fidelity, int32_t samples, [PixelFormat](#) depthStencil)

Creates a new instance of collection of drawable textures for performing 3D scene rendering.
- AFTERWARP_API void [TextureCabinetDestroy](#) (struct [TextureCabinet_t](#) *textureCabinet)

Releases given instance of drawable textures collection.
- AFTERWARP_API struct [Device_t](#) * [TextureCabinetGetDevice](#) (struct [TextureCabinet_t](#) *textureCabinet)

Returns device associated with the given textures collection.
- AFTERWARP_API void [TextureCabinetGetSize](#) (struct [TextureCabinet_t](#) *textureCabinet, [Point](#) *size)

Returns the size of the integrated textures.
- AFTERWARP_API [LibraryBool](#) [TextureCabinetSetSize](#) (struct [TextureCabinet_t](#) *textureCabinet, [Point](#) const *size)

Updates the size of integrated textures and buffers.
- AFTERWARP_API int32_t [TextureCabinetGetSamples](#) (struct [TextureCabinet_t](#) *textureCabinet)

Returns number of samples used in the textures.
- AFTERWARP_API [PixelFormat](#) [TextureCabinetGetDepthStencil](#) (struct [TextureCabinet_t](#) *textureCabinet)

Returns format used for depth/stencil buffers.
- AFTERWARP_API [TextureFidelity](#) [TextureCabinetGetFidelity](#) (struct [TextureCabinet_t](#) *textureCabinet)

Returns level of fidelity used for choosing integrated pixel formats.
- AFTERWARP_API [TextureCabinetAttributes](#) [TextureCabinetGetAttributes](#) (struct [TextureCabinet_t](#) *textureCabinet)

Returns rendering attributes of the collection.
- AFTERWARP_API [LibraryBool](#) [TextureCabinetSetAttributes](#) (struct [TextureCabinet_t](#) *textureCabinet, [TextureCabinetAttributes](#) attributes)
- AFTERWARP_API struct [Texture_t](#) * [TextureCabinetGetTexture](#) (struct [TextureCabinet_t](#) *textureCabinet, [TextureCabinetType](#) type)

Returns texture from the container of the given type.
- AFTERWARP_API struct [Texture_t](#) * [TextureCabinetGetFinalTexture](#) (struct [TextureCabinet_t](#) *textureCabinet)

Returns final color texture that can contains the scene.
- AFTERWARP_API [LibraryBool](#) [TextureCabinetClear](#) (struct [TextureCabinet_t](#) *textureCabinet, [TextureCabinetPass](#) pass, [In_opt_FloatColor](#) const *color)
- AFTERWARP_API [LibraryBool](#) [TextureCabinetBegin](#) (struct [TextureCabinet_t](#) *textureCabinet, [TextureCabinetPass](#) pass)

Begins rendering the given pass.
- AFTERWARP_API void [TextureCabinetEnd](#) (struct [TextureCabinet_t](#) *textureCabinet)

Ends rendering the pass that was previously started.
- AFTERWARP_API [LibraryBool](#) [TextureCabinetRendering](#) (struct [TextureCabinet_t](#) *textureCabinet)

Indicates that one of the passes is currently being rendered.

- AFTERWARP_API [LibraryBool TextureCabinetFilter](#) (struct [TextureCabinet_t](#) *textureCabinet, [TextureCabinetFilterType](#) filter, [Matrix](#) const *projection)
- AFTERWARP_API [LibraryBool TextureCabinetResolve](#) (struct [TextureCabinet_t](#) *textureCabinet)
Renders a multisample HDR color texture into a final color texture.
- AFTERWARP_API [LibraryBool TextureCabinetPresent](#) (struct [TextureCabinet_t](#) *textureCabinet)
Renders the final color texture on the current rendering surface.
- AFTERWARP_API void [TextureCabinetGetToneMappingBloom](#) (struct [TextureCabinet_t](#) *textureCabinet, [ToneMappingBloom](#) *parameters)
Returns tone-mapping and bloom parameters.
- AFTERWARP_API void [TextureCabinetSetToneMappingBloom](#) (struct [TextureCabinet_t](#) *textureCabinet, [ToneMappingBloom](#) const *parameters)
Updates tone-mapping and bloom parameters.
- AFTERWARP_API void [TextureCabinetGetAmbientOcclusionParameters](#) (struct [TextureCabinet_t](#) *textureCabinet, [AmbientOcclusionParameters](#) *parameters)
Returns ambient occlusion parameters.
- AFTERWARP_API void [TextureCabinetSetAmbientOcclusionParameters](#) (struct [TextureCabinet_t](#) *textureCabinet, [AmbientOcclusionParameters](#) const *parameters)
Updates ambient occlusion parameters.
- AFTERWARP_API [LibraryBool TextureCabinetRetrieveGlassyMetrics](#) (struct [TextureCabinet_t](#) *textureCabinet, [uint32_t](#) *values)
- AFTERWARP_API struct [Device_t](#) * [ShadowCasterGetDevice](#) (struct [ShadowCaster_t](#) *shadowCaster)
Returns device associated with the given shadow shadowCaster.
- AFTERWARP_API void [ShadowCasterGetPosition](#) (struct [ShadowCaster_t](#) *shadowCaster, [Point](#) *position)
Returns position of the shadow map in parent atlas.
- AFTERWARP_API void [ShadowCasterGetSize](#) (struct [ShadowCaster_t](#) *shadowCaster, [Point](#) *size)
Returns size of the shadow map.
- AFTERWARP_API void [ShadowCasterGetViewProjection](#) (struct [ShadowCaster_t](#) *shadowCaster, [Matrix](#) *viewProjection)
Returns a combined view and projection matrices of the shadow caster.
- AFTERWARP_API void [ShadowCasterSetViewProjection](#) (struct [ShadowCaster_t](#) *shadowCaster, [Matrix](#) const *viewProjection)
Updates a combined view and projection matrices of the shadow caster.
- AFTERWARP_API struct [ShadowCastingAtlas_t](#) * [ShadowCasterGetAtlas](#) (struct [ShadowCaster_t](#) *shadowCaster)
Returns pointer to an owner atlas.
- AFTERWARP_API [LibraryBool ShadowCasterClear](#) (struct [ShadowCaster_t](#) *shadowCaster)
Clears the integrated textures.
- AFTERWARP_API [LibraryBool ShadowCasterBegin](#) (struct [ShadowCaster_t](#) *shadowCaster)
Starts rendering to shadow caster's texture to build a shadow map.
- AFTERWARP_API void [ShadowCasterEnd](#) (struct [ShadowCaster_t](#) *shadowCaster)
Finishes rendering to shadow caster's texture.
- AFTERWARP_API [LibraryBool ShadowCasterFilter](#) (struct [ShadowCaster_t](#) *shadowCaster)
Performs filtering on the shadow map and accomodates it on the atlas.
- AFTERWARP_API [LibraryBool ShadowCasterRendering](#) (struct [ShadowCaster_t](#) *shadowCaster)
Indicates whether rendering is taking place.
- AFTERWARP_API struct [Texture_t](#) * [ShadowCasterGetTexture](#) (struct [ShadowCaster_t](#) *shadowCaster, [int32_t](#) index)
Returns one of the textures currently used by the shadow caster.
- AFTERWARP_API struct [ShadowCastingAtlas_t](#) * [ShadowCastingAtlasCreate](#) (struct [Device_t](#) *device, [Point](#) const *size, [TechniqueShadows](#) technique, [int32_t](#) samples, [int32_t](#) padding)
Creates a new instance of shadow caster and map management module.
- AFTERWARP_API void [ShadowCastingAtlasDestroy](#) (struct [ShadowCastingAtlas_t](#) *shadowCastingAtlas)

- Releases given instance of shadow caster and map management module.*
- AFTERWARP_API struct Device_t * ShadowCastingAtlasGetDevice (struct ShadowCastingAtlas_t *shadowCastingAtlas)
Returns device associated with the given textures collection.
 - AFTERWARP_API Technique Shadows ShadowCastingAtlasGetTechnique (struct ShadowCastingAtlas_t *shadowCastingAtlas)
Returns shadow rendering technique.
 - AFTERWARP_API void ShadowCastingAtlasGetParameters (struct ShadowCastingAtlas_t *shadowCastingAtlas, ShadowParameters *parameters)
Returns parameters that define how shadows are rendered.
 - AFTERWARP_API LibraryBool ShadowCastingAtlasSetParameters (struct ShadowCastingAtlas_t *shadowCastingAtlas, ShadowParameters const *parameters)
Updates shadow rendering parameters.
 - AFTERWARP_API void ShadowCastingAtlasGetSize (struct ShadowCastingAtlas_t *shadowCastingAtlas, Point *size)
Returns size of the shadow-casting shadowCastingAtlas.
 - AFTERWARP_API int32_t ShadowCastingAtlasGetSamples (struct ShadowCastingAtlas_t *shadowCastingAtlas)
Returns number of samples used for shadow map multisampling.
 - AFTERWARP_API int32_t ShadowCastingAtlasGetPadding (struct ShadowCastingAtlas_t *shadowCastingAtlas)
Returns padding used to accomodate shadow maps.
 - AFTERWARP_API struct Texture_t * ShadowCastingAtlasGetTexture (struct ShadowCastingAtlas_t *shadowCastingAtlas)
Returns shadow mapping atlas texture containing existing shadow maps.
 - AFTERWARP_API struct ShadowCaster_t * ShadowCastingAtlasAdd (struct ShadowCastingAtlas_t *shadowCastingAtlas, Point const *size, LibraryBool shared)
 - AFTERWARP_API void ShadowCastingAtlasErase (struct ShadowCastingAtlas_t *shadowCastingAtlas, struct ShadowCaster_t *caster)
Removes an existing shadow caster.
 - AFTERWARP_API void ShadowCastingAtlasClear (struct ShadowCastingAtlas_t *shadowCastingAtlas)
Removes all existing shadow casters.
 - AFTERWARP_API int32_t ShadowCastingAtlasGetCount (struct ShadowCastingAtlas_t *shadowCastingAtlas)
Returns number of existing shadow casters.
 - AFTERWARP_API struct ShadowCaster_t * ShadowCastingAtlasGetCaster (struct ShadowCastingAtlas_t *shadowCastingAtlas, int32_t index)
Returns shadow caster for the given index.
 - AFTERWARP_API LibraryBool ShadowCastingAtlasBorderFill (struct ShadowCastingAtlas_t *shadowCastingAtlas)
 - AFTERWARP_API struct Scene_t * SceneCreateDepthsNormals (struct Device_t *device)
Creates a new instance of depths/normals rendering scene.
 - AFTERWARP_API struct Scene_t * SceneCreateModeling (struct Device_t *device)
Creates a new instance of 3D rendering scene.
 - AFTERWARP_API void SceneDestroy (struct Scene_t *scene)
Releases an existing scene rendering instance.
 - AFTERWARP_API struct Device_t * SceneGetDevice (struct Scene_t *scene)
Returns device associated with the given scene.
 - AFTERWARP_API int32_t SceneGetVertexElementCount (struct Scene_t *scene, int32_t index)
Returns number of items in vertex element declaration associated with the given index.
 - AFTERWARP_API VertexElement const * SceneGetVertexElements (struct Scene_t *scene, int32_t index)
Returns items in vertex element declaration associated with the given index.

- AFTERWARP_API [LibraryBool](#) [SceneSetVertexElements](#) (struct [Scene_t](#) *scene, int32_t index, [VertexElement](#) const vertexElements[], int32_t vertexElementCount)
Changes vertex element declaration associated with the given index.
- AFTERWARP_API [LibraryBool](#) [SceneSetVertexElementsFromTextModeller](#) (struct [Scene_t](#) *scene, int32_t index)
Changes vertex element declaration associated with the given index.
- AFTERWARP_API int32_t [SceneGetActiveVertexElements](#) (struct [Scene_t](#) *scene)
Returns currently active index of vertex elements declaration.
- AFTERWARP_API [LibraryBool](#) [SceneSetActiveVertexElements](#) (struct [Scene_t](#) *scene, int32_t index)
Changes currently active index of vertex elements declaration.
- AFTERWARP_API [SceneAttributes](#) [SceneGetAttributes](#) (struct [Scene_t](#) *scene)
Returns attributes that define the rendering behavior of the scene.
- AFTERWARP_API [LibraryBool](#) [SceneSetAttributes](#) (struct [Scene_t](#) *scene, [SceneAttributes](#) attributes)
Updates attributes that define the rendering behavior of the scene.
- AFTERWARP_API void [SceneGetWorld](#) (struct [Scene_t](#) *scene, [Matrix](#) *world)
Returns object world matrix.
- AFTERWARP_API void [SceneSetWorld](#) (struct [Scene_t](#) *scene, [Matrix](#) const *world)
Updates object world matrix.
- AFTERWARP_API void [SceneGetView](#) (struct [Scene_t](#) *scene, [Matrix](#) *view)
Returns view matrix.
- AFTERWARP_API void [SceneSetView](#) (struct [Scene_t](#) *scene, [Matrix](#) const *view)
Updates view matrix.
- AFTERWARP_API void [SceneGetProjection](#) (struct [Scene_t](#) *scene, [Matrix](#) *projection)
Returns projection matrix.
- AFTERWARP_API void [SceneSetProjection](#) (struct [Scene_t](#) *scene, [Matrix](#) const *projection)
Updates projection matrix.
- AFTERWARP_API [LibraryBool](#) [SceneInstances](#) (struct [Scene_t](#) *scene, [Matrix](#) const transforms[], [FloatColor](#) const colors[], int32_t count)
- AFTERWARP_API int32_t [SceneGetInstancesCount](#) (struct [Scene_t](#) *scene)
Returns maximum number of supported instances.
- AFTERWARP_API struct [Program_t](#) * [SceneGetProgram](#) (struct [Scene_t](#) *scene)
Returns currently active shader program handle.
- AFTERWARP_API [LibraryBool](#) [SceneBegin](#) (struct [Scene_t](#) *scene)
Activates the appropriate shader program and begins rendering the scene.
- AFTERWARP_API void [SceneEnd](#) (struct [Scene_t](#) *scene)
Finishes rendering the scene and deactivates previously activated shader program.
- AFTERWARP_API [LibraryBool](#) [SceneRendering](#) (struct [Scene_t](#) *scene)
Indicates that the rendering is currently taking place (inside [SceneBegin\(\)](#) / [SceneEnd\(\)](#) block).
- AFTERWARP_API void [SceneResetCache](#) (struct [Scene_t](#) *scene)
Clears cached buffers and resources in the scene.
- AFTERWARP_API void [SceneGetMaterial](#) (struct [Scene_t](#) *scene, [ObjectMaterial](#) *material)
Returns scene object's material properties.
- AFTERWARP_API [LibraryBool](#) [SceneSetMaterial](#) (struct [Scene_t](#) *scene, [ObjectMaterial](#) const *material)
Updates scene object's material properties.
- AFTERWARP_API void [SceneGetToneMappingCoefficients](#) (struct [Scene_t](#) *scene, [Vector4](#) *coefficients)
- AFTERWARP_API [LibraryBool](#) [SceneSetToneMappingCoefficients](#) (struct [Scene_t](#) *scene, [Vector4](#) const *coefficients)
- AFTERWARP_API void [SceneGetParallaxMappingParameters](#) (struct [Scene_t](#) *scene, [ParallaxMappingParameters](#) *parameters)
Returns parameters that define Parallax Mapping technique is performed.
- AFTERWARP_API [LibraryBool](#) [SceneSetParallaxMappingParameters](#) (struct [Scene_t](#) *scene, [ParallaxMappingParameters](#) const *parameters)
Updates parameters that define Parallax Mapping technique is performed.

- AFTERWARP_API struct [Sampler_t](#) * [SceneGetSampler](#) (struct [Scene_t](#) *scene, [SceneSamplerType](#) type)
Returns sampler of the given type associated with the scene.
- AFTERWARP_API struct [Texture_t](#) * [SceneGetTexture](#) (struct [Scene_t](#) *scene, [SceneTextureType](#) type)
Returns texture of the given type associated with the scene.
- AFTERWARP_API [LibraryBool](#) [SceneSetTexture](#) (struct [Scene_t](#) *scene, struct [Texture_t](#) *texture, [SceneTextureType](#) type)
Updates texture of the given type associated with the scene.
- AFTERWARP_API struct [SceneLights_t](#) * [SceneGetLights](#) (struct [Scene_t](#) *scene)
Returns lights container module associated with the scene.
- AFTERWARP_API void [SceneSetLights](#) (struct [Scene_t](#) *scene, struct [SceneLights_t](#) *sceneLights)
- AFTERWARP_API struct [ShadowCastingAtlas_t](#) * [SceneGetShadowCastingAtlas](#) (struct [Scene_t](#) *scene)
Returns shadow casting atlas associated with the scene.
- AFTERWARP_API void [SceneSetShadowCastingAtlas](#) (struct [Scene_t](#) *scene, struct [ShadowCastingAtlas_t](#) *shadowCastingAtlas)
- AFTERWARP_API struct [TextureCabinet_t](#) * [SceneGetTextureCabinet](#) (struct [Scene_t](#) *scene)
Returns an associated texture cabinet.
- AFTERWARP_API void [SceneSetTextureCabinet](#) (struct [Scene_t](#) *scene, struct [TextureCabinet_t](#) *textureCabinet)
- AFTERWARP_API [LibraryBool](#) [ScenePrepare](#) (struct [Scene_t](#) *scene)
- AFTERWARP_API struct [OceanSimulation_t](#) * [OceanSimulationCreate](#) (struct [Device_t](#) *device)
Creates a new instance of module for rendering semi-infinite 3D ocean wave fields.
- AFTERWARP_API void [OceanSimulationDestroy](#) (struct [OceanSimulation_t](#) *oceanSimulation)
Releases an existing ocean rendering module.
- AFTERWARP_API struct [Device_t](#) * [OceanSimulationGetDevice](#) (struct [OceanSimulation_t](#) *oceanSimulation)
Returns device associated with the given module.
- AFTERWARP_API void [OceanSimulationGetWavesParameters](#) (struct [OceanSimulation_t](#) *oceanSimulation, [OceanWavesParameters](#) *parameters)
Returns ocean waves simulation parameters.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationSetWavesParameters](#) (struct [OceanSimulation_t](#) *oceanSimulation, [OceanWavesParameters](#) const *parameters)
Updates ocean waves simulation parameters.
- AFTERWARP_API struct [Texture_t](#) * [OceanSimulationGetWavesTexture](#) (struct [OceanSimulation_t](#) *oceanSimulation, [int32_t](#) index)
Returns an existing ocean waves simulation texture. This would be mostly useful for debug purposes.
- AFTERWARP_API void [OceanSimulationGetSections](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Point](#) *sections)
Returns number of sub-divisions in screen-space.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationSetSections](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Point](#) const *sections)
Changes number of sub-divisions in screen-space.
- AFTERWARP_API void [OceanSimulationGetView](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Matrix](#) *view)
Returns view transformation matrix.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationSetView](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Matrix](#) const *view)
Changes view transformation matrix.
- AFTERWARP_API void [OceanSimulationGetProjection](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Matrix](#) *projection)
Returns projection transformation matrix.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationSetProjection](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Matrix](#) const *projection)
Changes projection transformation matrix.

- AFTERWARP_API void [OceanSimulationGetScale](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Vector](#) *scale)
Returns scale of the displacement map.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationSetScale](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Vector](#) const *scale)
Changes scale of the displacement map.
- AFTERWARP_API void [OceanSimulationGetPlane](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Vector4](#) *plane)
Returns grid's plane equation.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationSetPlane](#) (struct [OceanSimulation_t](#) *oceanSimulation, [Vector4](#) const *plane)
Changes grid's plane equation.
- AFTERWARP_API float [OceanSimulationGetViewDistance](#) (struct [OceanSimulation_t](#) *oceanSimulation)
Returns maximum height-map viewing distance.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationSetViewDistance](#) (struct [OceanSimulation_t](#) *oceanSimulation, float viewDistance)
Changes maximum height-map viewing distance.
- AFTERWARP_API [SceneAttributes](#) [OceanSimulationGetAttributes](#) (struct [OceanSimulation_t](#) *oceanSimulation)
Returns currently set rendering attributes.
- AFTERWARP_API void [OceanSimulationSetAttributes](#) (struct [OceanSimulation_t](#) *oceanSimulation, [SceneAttributes](#) attributes)
- AFTERWARP_API void [OceanSimulationGetMaterial](#) (struct [OceanSimulation_t](#) *oceanSimulation, [OceanMaterial](#) *material)
Returns ocean's material.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationSetMaterial](#) (struct [OceanSimulation_t](#) *oceanSimulation, [OceanMaterial](#) const *material)
Updates ocean's material.
- AFTERWARP_API struct [Sampler_t](#) * [OceanSimulationGetSamplerShadow](#) (struct [OceanSimulation_t](#) *oceanSimulation)
Returns sampler for reading from shadow map.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationUpdate](#) (struct [OceanSimulation_t](#) *oceanSimulation, double latency)
Executes the simulation of ocean waves and updates internal textures.
- AFTERWARP_API [LibraryBool](#) [OceanSimulationRender](#) (struct [OceanSimulation_t](#) *oceanSimulation, struct [Texture_t](#) *linearDepths, struct [SceneLights_t](#) *sceneLights, struct [ShadowCastingAtlas_t](#) *atlas)
Renders the simulation to an existing active rendering surface.
- AFTERWARP_API struct [ObjectModels_t](#) * [ObjectModelGetOwner](#) (struct [ObjectModel_t](#) *objectModel)
Returns a container that owns the object.
- AFTERWARP_API [ObjectModelID](#) [ObjectModelGetID](#) (struct [ObjectModel_t](#) *objectModel)
Returns object's unique identification number.
- AFTERWARP_API void [ObjectModelGetName](#) (struct [ObjectModel_t](#) *objectModel, char *modelName, [_Inout_ int32_t](#) *modelNameLength)
Retrieves object's name.
- AFTERWARP_API [ObjectPayload](#) [ObjectModelGetPayload](#) (struct [ObjectModel_t](#) *objectModel)
Returns object's payload.
- AFTERWARP_API [LibraryBool](#) [ObjectModelSetName](#) (struct [ObjectModel_t](#) *objectModel, char const *name)
Changes object's name.
- AFTERWARP_API void [ObjectModelGetDescription](#) (struct [ObjectModel_t](#) *objectModel, char *description, [_Inout_ int32_t](#) *descriptionLength)
Retrieves object's description text.

- AFTERWARP_API [LibraryBool](#) [ObjectModelSetDescription](#) (struct [ObjectModel_t](#) *objectModel, char const *description)
Changes object's description text.
- AFTERWARP_API struct [MeshVoxel_t](#) * [ObjectModelGetVoxel](#) (struct [ObjectModel_t](#) *objectModel)
Returns object model's voxel representation.
- AFTERWARP_API void [ObjectModelSetVoxel](#) (struct [ObjectModel_t](#) *objectModel, struct [MeshVoxel_t](#) *meshVoxel)
Updates object model's voxel representation.
- AFTERWARP_API struct [SceneMesh_t](#) * [ObjectModelGetMesh](#) (struct [ObjectModel_t](#) *objectModel)
Returns object's renderable mesh.
- AFTERWARP_API [LibraryBool](#) [ObjectModelSetMesh](#) (struct [ObjectModel_t](#) *objectModel, struct [SceneMesh_t](#) *sceneMesh)
Changes object's renderable mesh.
- AFTERWARP_API void [ObjectModelGetMeshName](#) (struct [ObjectModel_t](#) *objectModel, char *meshName, [_Inout_](#) int32_t *meshNameLength)
Returns name of the currently associated mesh (or NULL, if no mesh is associated).
- AFTERWARP_API [LibraryBool](#) [ObjectModelSetMeshName](#) (struct [ObjectModel_t](#) *objectModel, char const *meshName)
Changes object's renderable mesh to one with the given name.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelGetParent](#) (struct [ObjectModel_t](#) *objectModel)
Returns object model's parent.
- AFTERWARP_API [LibraryBool](#) [ObjectModelSetParent](#) (struct [ObjectModel_t](#) *objectModel, struct [ObjectModel_t](#) *parent)
Changes object model's parent.
- AFTERWARP_API int32_t [ObjectModelGetChildCount](#) (struct [ObjectModel_t](#) *objectModel)
Returns number of object model's children.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelGetChild](#) (struct [ObjectModel_t](#) *objectModel, int32_t index)
Returns pointer to a particular object model's child.
- AFTERWARP_API void [ObjectModelGetTransform](#) (struct [ObjectModel_t](#) *objectModel, [ModelTransform](#) transform, [Matrix](#) *matrix)
Retrieves one of current object model's transforms.
- AFTERWARP_API [LibraryBool](#) [ObjectModelSetTransform](#) (struct [ObjectModel_t](#) *objectModel, [ModelTransform](#) transform, [Matrix](#) const *matrix)
Updates one of current object model's transforms.
- AFTERWARP_API void [ObjectModelGetPosition](#) (struct [ObjectModel_t](#) *objectModel, [Vector](#) *position)
Extracts and returns current object position.
- AFTERWARP_API float [ObjectModelGetDepthBias](#) (struct [ObjectModel_t](#) *objectModel)
Returns current depth bias, which is used during object selection.
- AFTERWARP_API void [ObjectModelSetDepthBias](#) (struct [ObjectModel_t](#) *objectModel, float depthBias)
Updates depth bias, which is used during object selection.
- AFTERWARP_API void [ObjectModelGetAABB](#) (struct [ObjectModel_t](#) *objectModel, [Vector](#) *boxMin, [Vector](#) *boxMax)
Updates and returns axis-aligned bounding box of the object.
- AFTERWARP_API uint32_t [ObjectModelGetAttributes](#) (struct [ObjectModel_t](#) *objectModel)
Returns current object model's attributes.
- AFTERWARP_API void [ObjectModelSetAttributes](#) (struct [ObjectModel_t](#) *objectModel, uint32_t attributes)
Updates current object's attributes.
- AFTERWARP_API int32_t [ObjectModelGetOrderIndex](#) (struct [ObjectModel_t](#) *objectModel)
Returns current priority order index.
- AFTERWARP_API void [ObjectModelSetOrderIndex](#) (struct [ObjectModel_t](#) *objectModel, int32_t orderIndex)
Updates object's priority order index.

- AFTERWARP_API uint64_t [ObjectModelGetLayers](#) (struct [ObjectModel_t](#) *objectModel)
Returns layers on which the object is visible.
- AFTERWARP_API void [ObjectModelSetLayers](#) (struct [ObjectModel_t](#) *objectModel, uint64_t layers)
- AFTERWARP_API void [ObjectModelGetSize](#) (struct [ObjectModel_t](#) *objectModel, [Vector](#) *size)
Returns the object's size.
- AFTERWARP_API void [ObjectModelSetSize](#) (struct [ObjectModel_t](#) *objectModel, [Vector](#) const *size)
Changes the object's size.
- AFTERWARP_API void [ObjectModelGetAlignments](#) (struct [ObjectModel_t](#) *objectModel, [MeshAligns](#) *aligns)
Returns mesh alignments.
- AFTERWARP_API void [ObjectModelSetAlignments](#) (struct [ObjectModel_t](#) *objectModel, [MeshAligns](#) const *aligns)
Changes mesh alignments.
- AFTERWARP_API int32_t [ObjectModelGetMaterial](#) (struct [ObjectModel_t](#) *objectModel)
Returns object's material.
- AFTERWARP_API void [ObjectModelSetMaterial](#) (struct [ObjectModel_t](#) *objectModel, int32_t material)
Updates object's material.
- AFTERWARP_API void [ObjectModelGetColor](#) (struct [ObjectModel_t](#) *objectModel, [FloatColor](#) *color)
Returns object's color.
- AFTERWARP_API void [ObjectModelSetColor](#) (struct [ObjectModel_t](#) *objectModel, [FloatColor](#) const *color)
Updates object's color.
- AFTERWARP_API void [ObjectModelGetHighlight](#) (struct [ObjectModel_t](#) *objectModel, [FloatColor](#) *highlight)
Returns object's highlight color.
- AFTERWARP_API void [ObjectModelSetHighlight](#) (struct [ObjectModel_t](#) *objectModel, [FloatColor](#) const *highlight)
Updates object's highlight color.
- AFTERWARP_API void [ObjectModelConnectLatches](#) (struct [ObjectModel_t](#) *objectModel, int32_t latch↔Parent, int32_t latchLocal)
Connects a local bone joint with the given index to a corresponding joint of the parent.
- AFTERWARP_API void [ObjectModelConnectLatchesByName](#) (struct [ObjectModel_t](#) *objectModel, _In_↔opt_ char const *latchParentName, _In_opt_ char const *latchLocalName)
Connects a local bone joint with the given name (case-insensitive) to a corresponding joint of the parent.
- AFTERWARP_API void [ObjectModelGetConnectedLatches](#) (struct [ObjectModel_t](#) *objectModel, int32_↔t *latchParent, int32_t *latchLocal)
Returns indexes of currently connected bone joints.
- AFTERWARP_API float [ObjectModelGetWaypointDistance](#) (struct [ObjectModel_t](#) *objectModel, int32_↔t group)
Returns calculated waypoint traveling distance for the given latch group.
- AFTERWARP_API void [ObjectModelGetLatchWaypointCouple](#) (struct [ObjectModel_t](#) *objectModel, int32_t group, float distance, [Vector](#) *position, [Quaternion](#) *orientation)
Calculates interpolated position and orientation based on the given distance for the given latch group.
- AFTERWARP_API void [ObjectModelGetLatchWaypointCoupleMatrix](#) (struct [ObjectModel_t](#) *objectModel, int32_t group, float distance, [Matrix](#) *transform)
- AFTERWARP_API void [ObjectModelGetLatchTransform](#) (struct [ObjectModel_t](#) *objectModel, [Matrix](#) *transform, int32_t latchIndex, [LibraryBool](#) local)
Retrieves either local or global transformation matrix for a given latch index.
- AFTERWARP_API void [ObjectModelGetLatchTransformByName](#) (struct [ObjectModel_t](#) *objectModel, [Matrix](#) *transform, char const *name, [LibraryBool](#) local)
Retrieves either local or global transformation matrix for a latch with the given name.
- AFTERWARP_API void [ObjectModelInvalidate](#) (struct [ObjectModel_t](#) *objectModel)
Marks local transform, volume and position as dirty, including children.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelGetNext](#) (struct [ObjectModel_t](#) *objectModel)

- AFTERWARP_API struct [ObjectModels_t](#) * [ObjectModelsCreate](#) (struct [SceneMeshes_t](#) *sceneMeshes)
Creates a new instance of the container associated with a particular list of meshes.
- AFTERWARP_API void [ObjectModelsDestroy](#) (struct [ObjectModels_t](#) *objectModels)
Releases current instance of the container.
- AFTERWARP_API struct [SceneMeshes_t](#) * [ObjectModelsGetMeshes](#) (struct [ObjectModels_t](#) *objectModels)
Returns the associated mesh container.
- AFTERWARP_API int32_t [ObjectModelsGetObjectCount](#) (struct [ObjectModels_t](#) *objectModels)
Returns number of existing objects.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelsGetFirst](#) (struct [ObjectModels_t](#) *objectModels)
Returns a first object from the collection of existing objects.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelsAddWithID](#) (struct [ObjectModels_t](#) *objectModels, [ObjectModelID](#) id, [_In_opt_](#) char const *name, [ObjectPayload](#) payload, uint32_t attributes)
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelsAdd](#) (struct [ObjectModels_t](#) *objectModels, [_In_](#)↔
[opt_](#) char const *name, [ObjectPayload](#) payload, uint32_t attributes)
Creates a new object with the given name and/or payload association.
- AFTERWARP_API void [ObjectModelsErase](#) (struct [ObjectModels_t](#) *objectModels, struct [ObjectModel_t](#) *objectModel)
Removes the given object.
- AFTERWARP_API void [ObjectModelsClear](#) (struct [ObjectModels_t](#) *objectModels)
Removes all objects.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelsGetObjectByID](#) (struct [ObjectModels_t](#) *object↔
Models, [ObjectModelID](#) id)
Returns an object with the given ID or NULL if such doesn't exist.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelsGetObjectByName](#) (struct [ObjectModels_t](#) *object↔
Models, char const *name)
Returns an object with the given name (case-insensitive) or NULL if such doesn't exist.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelsPayload](#) (struct [ObjectModels_t](#) *objectModels, [ObjectPayload](#) payload)
Returns object that corresponds to the given payload in 3D object model container.
- AFTERWARP_API struct [ObjectModelView_t](#) * [ObjectModelsCreateView](#) (struct [ObjectModels_t](#) *object↔
Models)
Creates a new view associated with this collection.
- AFTERWARP_API void [ObjectModelsEraseView](#) (struct [ObjectModels_t](#) *objectModels, struct [ObjectModelView_t](#) *objectModelView)
Erases an existing view from the collection.
- AFTERWARP_API void [ObjectModelsClearViews](#) (struct [ObjectModels_t](#) *objectModels)
Removes all existing views from the collection.
- AFTERWARP_API struct [ObjectModels_t](#) * [ObjectModelViewGetOwner](#) (struct [ObjectModelView_t](#) *object↔
ModelView)
Returns parent of the view container.
- AFTERWARP_API void [ObjectModelViewGetView](#) (struct [ObjectModelView_t](#) *objectModelView, [Matrix](#) *view)
Returns "View" transformation matrix of the container.
- AFTERWARP_API void [ObjectModelViewSetView](#) (struct [ObjectModelView_t](#) *objectModelView, [Matrix](#) const *view)
Changes "View" transformation matrix of the container.
- AFTERWARP_API void [ObjectModelViewGetProjection](#) (struct [ObjectModelView_t](#) *objectModelView, [Matrix](#) *projection, [LibraryBool](#) *depthClipNegative)
Returns "Projection" transformation matrix of the view container.
- AFTERWARP_API void [ObjectModelViewSetProjection](#) (struct [ObjectModelView_t](#) *objectModelView, [Matrix](#) const *projection, [LibraryBool](#) depthClipNegative)
Changes "Projection" transformation matrix of the view container.

- AFTERWARP_API void [ObjectModelViewGetViewProjection](#) (struct [ObjectModelView_t](#) *objectModelView, [Matrix](#) *viewProjection)
Returns combined "View/Projection" matrix of the container.
- AFTERWARP_API uint64_t [ObjectModelViewGetLayers](#) (struct [ObjectModelView_t](#) *objectModelView)
Returns layers that determine object visibility in the view.
- AFTERWARP_API void [ObjectModelViewSetLayers](#) (struct [ObjectModelView_t](#) *objectModelView, uint64_t layers)
Changes layers that determine object visibility in the view.
- AFTERWARP_API void [ObjectModelViewUpdateNeeded](#) (struct [ObjectModelView_t](#) *objectModelView)
Notifies the container that all visible objects must be recalculated.
- AFTERWARP_API void [ObjectModelViewInvalidate](#) (struct [ObjectModelView_t](#) *objectModelView)
Notifies the container that it needs to be repainted.
- AFTERWARP_API int32_t [ObjectModelViewGetObjectCount](#) (struct [ObjectModelView_t](#) *objectModelView)
Returns number of existing objects that are visible in the view.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelViewGetObject](#) (struct [ObjectModelView_t](#) *objectModelView, int32_t index)
Returns an object with the given index from the view container.
- AFTERWARP_API [LibraryBool](#) [ObjectModelViewUpdate](#) (struct [ObjectModelView_t](#) *objectModelView)
- AFTERWARP_API void [ObjectModelViewSort](#) (struct [ObjectModelView_t](#) *objectModelView, [ObjectModelViewCompare](#) compare)
Sorts the list of objects using a predefined comparison function.
- AFTERWARP_API void [ObjectModelViewSortWith](#) (struct [ObjectModelView_t](#) *objectModelView, [ObjectModelCompareFunc](#) compareFunc, void *user)
Sorts the list of objects using a custom comparison function.
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelViewSelect](#) (struct [ObjectModelView_t](#) *objectModelView, [Vector](#) const *origin, [Vector](#) const *direction, float *distance)
- AFTERWARP_API struct [ObjectModel_t](#) * [ObjectModelViewSelectAny](#) (struct [ObjectModelView_t](#) *objectModelView, [Vector](#) const *origin, [Vector](#) const *direction, float *distance)
- AFTERWARP_API int32_t [ObjectModelViewGetObjectsNotCulled](#) (struct [ObjectModelView_t](#) *objectModelView)
- AFTERWARP_API int32_t [ObjectModelViewGetIntersectedRays](#) (struct [ObjectModelView_t](#) *objectModelView)
Returns number of ray vs AABB intersections performed last time selection was performed.
- AFTERWARP_API int32_t [ObjectModelViewGetIntersectedObjects](#) (struct [ObjectModelView_t](#) *objectModelView)
Returns number of objects that passed intersection during last selection attempt.
- AFTERWARP_API [LibraryBool](#) [ObjectModelViewAutoDraw](#) (struct [ObjectModelView_t](#) *objectModelView, struct [Scene_t](#) *scene, struct [ObjectMaterials_t](#) *materials, uint32_t options, _Inout_ int32_t *drawCalls)
- AFTERWARP_API struct [MeshVoxel_t](#) * [MeshVoxelCreate](#) (void)
Creates an empty 3D mesh voxel representation.
- AFTERWARP_API void [MeshVoxelDestroy](#) (struct [MeshVoxel_t](#) *meshVoxel)
Releases the given 3D mesh voxel representation.
- AFTERWARP_API void [MeshVoxelTakeAway](#) (struct [MeshVoxel_t](#) *meshVoxel, struct [MeshVoxel_t](#) *meshVoxelAnother)
- AFTERWARP_API [LibraryBool](#) [MeshVoxelLoadFromFile](#) (struct [MeshVoxel_t](#) *meshVoxel, char const *fileName, int32_t dimensions[], uint8_t *levels, [LibraryBool](#) *colors)
Loads an existing 3D mesh voxel representation from a file on disk.
- AFTERWARP_API [LibraryBool](#) [MeshVoxelLoadFromFileInMemory](#) (struct [MeshVoxel_t](#) *meshVoxel, void *buffer, uint32_t bufferSize, int32_t dimensions[], uint8_t *levels, [LibraryBool](#) *colors)
Loads an existing 3D mesh voxel representation from a file in memory.
- AFTERWARP_API [LibraryBool](#) [MeshVoxelSaveToFile](#) (struct [MeshVoxel_t](#) *meshVoxel, char const *fileName, int32_t const dimensions[], uint8_t levels, [LibraryBool](#) colors)
Saves 3D mesh voxel representation to a file on disk.

- AFTERWARP_API void [MeshVoxelExtents](#) (struct [MeshVoxel_t](#) *meshVoxel, [Vector](#) *position, [Vector](#) *size)
Returns position and size of 3D mesh voxel representation.
- AFTERWARP_API void [MeshVoxelComputeParameters](#) (struct [MeshVoxel_t](#) *meshVoxel, int32_t dimensions[], uint8_t *levels, [LibraryBool](#) *colors)
Calculates existing voxel dimensions, levels and whether colors are present.
- AFTERWARP_API [LibraryBool](#) [MeshVoxelVisualize](#) (struct [MeshVoxel_t](#) *meshVoxel, uint8_t levelMax, [MeshVoxelVisualizeFunc](#) visualizeFunc, void *user)
Visualizes 3D mesh voxel representation by invoking callback function for each cube.
- AFTERWARP_API [LibraryBool](#) [MeshVoxelIntersect](#) (struct [MeshVoxel_t](#) *meshVoxel, [Vector](#) const *origin, [Vector](#) const *direction, [Matrix](#) const *world, [Matrix](#) const *view, float *distance, int32_t *testsCount)
Performs intersection between voxel representation and ray.
- AFTERWARP_API void [SceneMeshMaterialGetName](#) (struct [SceneMeshMaterial_t](#) *material, char *name, _Inout_ int32_t *nameLength)
- AFTERWARP_API void [SceneMeshMaterialGetShading](#) (struct [SceneMeshMaterial_t](#) *material, [SceneMeshMaterialShading](#) *shading)
Returns shading parameters of the material.
- AFTERWARP_API void [SceneMeshMaterialSetShading](#) (struct [SceneMeshMaterial_t](#) *material, [SceneMeshMaterialShading](#) const *shading)
Updates shading parameters of the material.
- AFTERWARP_API struct [Texture_t](#) * [SceneMeshMaterialGetTexture](#) (struct [SceneMeshMaterial_t](#) *material, uint8_t type)
Returns a texture associated with the given type or NULL if such does not exist.
- AFTERWARP_API [LibraryBool](#) [SceneMeshMaterialSetTexture](#) (struct [SceneMeshMaterial_t](#) *material, struct [Texture_t](#) *texture, uint8_t type)
- AFTERWARP_API void [SceneMeshMaterialReleaseTextures](#) (struct [SceneMeshMaterial_t](#) *material)
Releases existing textures.
- AFTERWARP_API int32_t [SceneMeshMaterialGetRangeCount](#) (struct [SceneMeshMaterial_t](#) *material)
Returns number of existing ranges.
- AFTERWARP_API [LibraryBool](#) [SceneMeshMaterialGetRange](#) (struct [SceneMeshMaterial_t](#) *material, int32_t index, _Out_ [SceneMeshMaterialRange](#) *range)
Returns a range with the given index.
- AFTERWARP_API [LibraryBool](#) [SceneMeshMaterialSetRange](#) (struct [SceneMeshMaterial_t](#) *material, int32_t index, [SceneMeshMaterialRange](#) const *range)
Updates a range with the given index.
- AFTERWARP_API int32_t [SceneMeshMaterialAddRange](#) (struct [SceneMeshMaterial_t](#) *material, [SceneMeshMaterialRange](#) const *range)
- AFTERWARP_API void [SceneMeshMaterialClearRanges](#) (struct [SceneMeshMaterial_t](#) *material)
Removes all existing ranges.
- AFTERWARP_API [LibraryBool](#) [SceneMeshMaterialCommit](#) (struct [SceneMeshMaterial_t](#) *material, struct [Device_t](#) *device, [TextureAttributes](#) attributes)
Converts existing "scratch" textures into hardware-based textures.
- AFTERWARP_API [LibraryBool](#) [SceneMeshMaterialCopy](#) (struct [SceneMeshMaterial_t](#) *material, struct [SceneMeshMaterial_t](#) *source)
- AFTERWARP_API struct [SceneMeshMaterials_t](#) * [SceneMeshMaterialsCreate](#) (void)
Creates a new 3D scene mesh material container.
- AFTERWARP_API void [SceneMeshMaterialsDestroy](#) (struct [SceneMeshMaterials_t](#) *materials)
Releases an existing instance of 3D scene mesh material container.
- AFTERWARP_API [LibraryBool](#) [SceneMeshMaterialsSetName](#) (struct [SceneMeshMaterials_t](#) *materials, char const *name)
Changes the name of the library.
- AFTERWARP_API void [SceneMeshMaterialsGetName](#) (struct [SceneMeshMaterials_t](#) *materials, char *name, _Inout_ int32_t *nameLength)
Returns name of the library.

- AFTERWARP_API int32_t [SceneMeshMaterialsGetCount](#) (struct [SceneMeshMaterials_t](#) *materials)
Returns total number of existing materials.
- AFTERWARP_API struct [SceneMeshMaterial_t](#) * [SceneMeshMaterialsGetMaterial](#) (struct [SceneMeshMaterials_t](#) *materials, int32_t index)
Returns an existing material with the given index, if such exists.
- AFTERWARP_API int32_t [SceneMeshMaterialsAdd](#) (struct [SceneMeshMaterials_t](#) *materials, char const *name)
Adds a new material to the collection.
- AFTERWARP_API void [SceneMeshMaterialsErase](#) (struct [SceneMeshMaterials_t](#) *materials, int32_t index)
Removes a material with the given index.
- AFTERWARP_API void [SceneMeshMaterialsClear](#) (struct [SceneMeshMaterials_t](#) *materials)
Removes and releases all existing materials.
- AFTERWARP_API [LibraryBool](#) [SceneMeshMaterialsCommit](#) (struct [SceneMeshMaterials_t](#) *materials, struct [Device_t](#) *device, [TextureAttributes](#) attributes)
Converts existing "scratch" textures into hardware-based textures.
- AFTERWARP_API [LibraryBool](#) [SceneMeshMaterialsCopy](#) (struct [SceneMeshMaterials_t](#) *materials, struct [SceneMeshMaterials_t](#) *materialsAnother)
Copies materials into the collection from another container.
- AFTERWARP_API void [SceneMeshMaterialsTakeAway](#) (struct [SceneMeshMaterials_t](#) *materials, struct [SceneMeshMaterials_t](#) *materialsAnother)
Takes away the internal contents from another material container without doing any copying.
- AFTERWARP_API [LibraryBool](#) [SceneMeshMaterialsTexturing](#) (struct [SceneMeshMaterials_t](#) *materials)
Reviews existing materials to determine whether any of them use texturing.
- AFTERWARP_API struct [SceneMeshLatches_t](#) * [SceneMeshLatchesCreate](#) (void)
Creates a new 3D scene mesh latches container.
- AFTERWARP_API void [SceneMeshLatchesDestroy](#) (struct [SceneMeshLatches_t](#) *latches)
Releases an existing instance of 3D scene mesh latches container.
- AFTERWARP_API int32_t [SceneMeshLatchesGetCount](#) (struct [SceneMeshLatches_t](#) *latches)
Returns total number of existing latches.
- AFTERWARP_API [LibraryBool](#) [SceneMeshLatchesGetLatch](#) (struct [SceneMeshLatches_t](#) *latches, int32_t index, [SceneMeshLatch](#) *latch)
Returns an existing latch with the given index.
- AFTERWARP_API [LibraryBool](#) [SceneMeshLatchesSetLatch](#) (struct [SceneMeshLatches_t](#) *latches, int32_t index, [SceneMeshLatch](#) const *latch)
Updates an existing latch with the given index.
- AFTERWARP_API int32_t [SceneMeshLatchesGetLatchIndex](#) (struct [SceneMeshLatches_t](#) *latches, char const *name)
Returns an index of a latch with the given name (case-insensitive) index or -1 if it doesn't exist.
- AFTERWARP_API int32_t [SceneMeshLatchesAdd](#) (struct [SceneMeshLatches_t](#) *latches, [_In_opt_](#) char const *name, int32_t type, int32_t group, [_In_opt_](#) [Vector](#) const *position, [_In_opt_](#) [Quaternion](#) const *orientation)
Adds a new latch to the collection.
- AFTERWARP_API void [SceneMeshLatchesErase](#) (struct [SceneMeshLatches_t](#) *latches, int32_t index)
Removes an existing latch with the given index.
- AFTERWARP_API void [SceneMeshLatchesClear](#) (struct [SceneMeshLatches_t](#) *latches)
Removes and releases all existing latches.
- AFTERWARP_API [LibraryBool](#) [SceneMeshLatchesCopy](#) (struct [SceneMeshLatches_t](#) *latches, struct [SceneMeshLatches_t](#) *latchesAnother)
Copies latches from another collection.
- AFTERWARP_API void [SceneMeshLatchesTakeAway](#) (struct [SceneMeshLatches_t](#) *latches, struct [SceneMeshLatches_t](#) *latchesAnother)
Takes away the internal contents from another collection without doing any copying.
- AFTERWARP_API [LibraryBool](#) [SceneMeshLatchesLoadFromFile](#) (struct [SceneMeshLatches_t](#) *latches, char const *fileName)
Loads latches from a file.

- Loads a collection of latches from file on disk.*
- AFTERWARP_API [LibraryBool](#) [SceneMeshLatchesLoadFromFileInMemory](#) (struct [SceneMeshLatches_t](#) *latches, void *buffer, uint32_t bufferSize)
- Loads a collection of latches from file in memory.*
- AFTERWARP_API [LibraryBool](#) [SceneMeshLatchesSaveToFile](#) (struct [SceneMeshLatches_t](#) *latches, char const *fileName)
- Saves a collection of latches to file on disk.*
- AFTERWARP_API float [SceneMeshLatchesGetWaypointDistance](#) (struct [SceneMeshLatches_t](#) *latches, int32_t group)
- Returns calculated waypoint traveling distance for the given latch group.*
- AFTERWARP_API void [SceneMeshLatchesGetWaypointCouple](#) (struct [SceneMeshLatches_t](#) *latches, int32_t group, float distance, [Vector](#) *position, [Quaternion](#) *orientation)
- Calculates interpolated position and orientation based on the given distance for the given latch group.*
- AFTERWARP_API void [SceneMeshLatchesInvalidateWaypoints](#) (struct [SceneMeshLatches_t](#) *latches)
- AFTERWARP_API void [SceneMeshGetName](#) (struct [SceneMesh_t](#) *sceneMesh, char *meshName, _Inout_ int32_t *meshNameLength)
- Returns name of the mesh.*
- AFTERWARP_API [ObjectPayload](#) [SceneMeshGetPayload](#) (struct [SceneMesh_t](#) *sceneMesh)
- Returns payload associated with the mesh.*
- AFTERWARP_API struct [MeshModel_t](#) * [SceneMeshGetModel](#) (struct [SceneMesh_t](#) *sceneMesh)
- Returns a renderable mesh model.*
- AFTERWARP_API struct [MeshVoxel_t](#) * [SceneMeshGetVoxel](#) (struct [SceneMesh_t](#) *sceneMesh)
- Returns voxel representation of the mesh.*
- AFTERWARP_API struct [SceneMeshMaterials_t](#) * [SceneMeshGetMaterials](#) (struct [SceneMesh_t](#) *sceneMesh)
- Returns materials associated with a 3D mesh.*
- AFTERWARP_API struct [MeshMetaTags_t](#) * [SceneMeshGetTags](#) (struct [SceneMesh_t](#) *sceneMesh)
- Returns meta-tags that represent sub-meshes within a 3D mesh.*
- AFTERWARP_API struct [SceneMeshLatches_t](#) * [SceneMeshGetLatches](#) (struct [SceneMesh_t](#) *sceneMesh)
- Returns an integrated collection of latches associated with the 3D mesh.*
- AFTERWARP_API void [SceneMeshGetBounds](#) (struct [SceneMesh_t](#) *sceneMesh, [Vector](#) *boundsMin, [Vector](#) *boundsMax)
- Returns minimum and maximum mesh boundaries.*
- AFTERWARP_API float [SceneMeshGetScale](#) (struct [SceneMesh_t](#) *sceneMesh)
- Returns scale of the mesh.*
- AFTERWARP_API void [SceneMeshGetSize](#) (struct [SceneMesh_t](#) *sceneMesh, [Vector](#) *size)
- Returns size of the mesh.*
- AFTERWARP_API uint8_t [SceneMeshGetVertexElementsIndex](#) (struct [SceneMesh_t](#) *sceneMesh)
- AFTERWARP_API void [SceneMeshSetVertexElementsIndex](#) (struct [SceneMesh_t](#) *sceneMesh, uint8_t vertexElementsIndex)
- AFTERWARP_API void [SceneMeshSetSlice](#) (struct [SceneMesh_t](#) *sceneMesh, struct [SceneMesh_t](#) *parent, struct [MeshMetaTag_t](#) *parentTag)
- Specify the parent mesh and the tag to which this slice would refer to.*
- AFTERWARP_API void [SceneMeshGetSlice](#) (struct [SceneMesh_t](#) *sceneMesh, _Out_ struct [SceneMesh_t](#) **parent, _Out_ struct [MeshMetaTag_t](#) **parentTag)
- AFTERWARP_API [LibraryBool](#) [SceneMeshAutoDraw](#) (struct [SceneMesh_t](#) *sceneMesh, struct [Scene_t](#) *scene, [ObjectMaterial](#) const *material, [FloatColor](#) const *color, [PrimitiveTopology](#) topology, int32_t instanceCount, int32_t elementCount, int32_t firstIndex, int32_t baseVertex, uint32_t options, _Inout_ int32_t *drawCalls)
- AFTERWARP_API [LibraryBool](#) [SceneMeshAutoDrawSliced](#) (struct [SceneMesh_t](#) *sceneMesh, struct [Scene_t](#) *scene, [ObjectMaterial](#) const *material, [FloatColor](#) const *color, [PrimitiveTopology](#) topology, int32_t instanceCount, uint32_t options, _Inout_ int32_t *drawCalls)
- AFTERWARP_API struct [SceneMeshes_t](#) * [SceneMeshesCreate](#) (struct [Device_t](#) *device)

Creates new instance of the container associated with a particular device.

- AFTERWARP_API void [SceneMeshesDestroy](#) (struct [SceneMeshes_t](#) *sceneMeshes)

Releases the container and its owned resources.

- AFTERWARP_API struct [SceneMesh_t](#) * [SceneMeshesAdd](#) (struct [SceneMeshes_t](#) *sceneMeshes, char const *name, [ObjectPayload](#) payload, [Vector](#) const *boundsMin, [Vector](#) const *boundsMax, float scale)

Creates a new mesh with the given parameters.

- AFTERWARP_API struct [SceneMesh_t](#) * [SceneMeshesAddFromBuffer](#) (struct [SceneMeshes_t](#) *sceneMeshes, char const *name, struct [MeshBuffer_t](#) *meshBuffer, [ObjectPayload](#) payload, [VertexElement](#) const vertexElements[], int32_t vertexElementCount, [_In_opt_ Vector](#) const *boundsMin, [_In_opt_ Vector](#) const *boundsMax, uint32_t channel, uint32_t semanticIndex, float scale)

Creates a new mesh, loading contents from a given mesh buffer.

- AFTERWARP_API struct [SceneMesh_t](#) * [SceneMeshesAddFromFile](#) (struct [SceneMeshes_t](#) *sceneMeshes, char const *name, char const *fileName, [ObjectPayload](#) payload, [VertexElement](#) const vertexElements[], int32_t vertexElementCount, uint32_t channel, uint32_t semanticIndex, [_Inout_ uint32_t](#) *options, float scale, [MeshLoadSaveFeedback](#) feedback, void *feedbackUser, char *debug, [_Inout_ int32_t](#) *debugLength)
- AFTERWARP_API [LibraryBool](#) [SceneMeshesSlice](#) (struct [SceneMeshes_t](#) *sceneMeshes, struct [SceneMesh_t](#) *parent, uint8_t type, char const *prefix)
- AFTERWARP_API struct [Device_t](#) * [SceneMeshesGetDevice](#) (struct [SceneMeshes_t](#) *sceneMeshes)

Returns device associated with the container.

- AFTERWARP_API int32_t [SceneMeshesGetCount](#) (struct [SceneMeshes_t](#) *sceneMeshes)

Returns number of existing meshes.

- AFTERWARP_API struct [SceneMesh_t](#) * [SceneMeshesGetMeshByIndex](#) (struct [SceneMeshes_t](#) *sceneMeshes, int32_t index)
- AFTERWARP_API struct [SceneMesh_t](#) * [SceneMeshesGetMeshByName](#) (struct [SceneMeshes_t](#) *sceneMeshes, char const *name)

Returns mesh with the given name (case-insensitive) or NULL if no mesh with such name exists.

- AFTERWARP_API struct [SceneMesh_t](#) * [SceneMeshesPayload](#) (struct [SceneMeshes_t](#) *sceneMeshes, [ObjectPayload](#) payload)

Returns a mesh that corresponds to the given payload or NULL if such doesn't exist.

- AFTERWARP_API void [SceneMeshesErase](#) (struct [SceneMeshes_t](#) *sceneMeshes, struct [SceneMesh_t](#) *sceneMesh)

Erases the given mesh.

- AFTERWARP_API void [SceneMeshesClear](#) (struct [SceneMeshes_t](#) *sceneMeshes)

Removes all existing meshes.

- AFTERWARP_API struct [Application_t](#) * [ApplicationCreate](#) ([_Inout_ ApplicationConfiguration](#) *configuration)

Creates a new application instance.

- AFTERWARP_API void [ApplicationDestroy](#) (struct [Application_t](#) *application)

Releases the given application instance.

- AFTERWARP_API void [ApplicationSetEvents](#) (struct [Application_t](#) *application, [ApplicationEvents](#) const *events, void *user)

- AFTERWARP_API [UntypedHandle](#) [ApplicationGetWindowHandle](#) (struct [Application_t](#) *application)

Retrieves the handle of main application window.

- AFTERWARP_API [LibraryBool](#) [ApplicationSetTitle](#) (struct [Application_t](#) *application, char const *title)

Sets new application title.

- AFTERWARP_API void [ApplicationGetTitle](#) (struct [Application_t](#) *application, char *title, [_Inout_ int32_t](#) *titleLength)

- AFTERWARP_API [LibraryBool](#) [ApplicationSetIconTitle](#) (struct [Application_t](#) *application, char const *iconTitle)

Sets new application's icon title.

- AFTERWARP_API void [ApplicationGetIconTitle](#) (struct [Application_t](#) *application, char *iconTitle, [_Inout_ int32_t](#) *iconTitleLength)

- AFTERWARP_API void [ApplicationGetExecutablePath](#) (struct [Application_t](#) *application, char *path, [_Inout_ int32_t](#) *pathLength)

- AFTERWARP_API void [ApplicationGetWindowRect](#) (struct [Application_t](#) *application, [Rect](#) *rect)
Retrieves current application's window rectangle.
- AFTERWARP_API void [ApplicationSetWindowRect](#) (struct [Application_t](#) *application, [Rect](#) const *rect)
Sets new application's window rectangle.
- AFTERWARP_API void [ApplicationGetClientRect](#) (struct [Application_t](#) *application, [Rect](#) *rect)
Retrieves current application's window client rectangle.
- AFTERWARP_API void [ApplicationSetClientSize](#) (struct [Application_t](#) *application, [Point](#) const *size)
Sets new application's window client size.
- AFTERWARP_API void [ApplicationGetMinimalSize](#) (struct [Application_t](#) *application, [Point](#) *size)
Returns current minimal size of the window. Values of zero means that there is no limit.
- AFTERWARP_API void [ApplicationSetMinimalSize](#) (struct [Application_t](#) *application, [Point](#) const *size)
Sets a minimal size of the window. Values of zero means that there is no limit.
- AFTERWARP_API double [ApplicationGetWindowScale](#) (struct [Application_t](#) *application)
Returns the scale of application window.
- AFTERWARP_API void [ApplicationInvalidate](#) (struct [Application_t](#) *application)
Requests the application window to be repainted.
- AFTERWARP_API [Key](#) [ApplicationTranslateVirtualKey](#) (struct [Application_t](#) *application, [int32_t](#) virtualKey)
Converts virtual key code to portable key code. Returns Key::Null if no equivalent is found.
- AFTERWARP_API [int32_t](#) [ApplicationConvertPortableKey](#) (struct [Application_t](#) *application, [Key](#) key)
Converts portable key code to virtual key code. Returns 0 if no equivalent is found.
- AFTERWARP_API [LibraryBool](#) [ApplicationExecute](#) (struct [Application_t](#) *application)
- AFTERWARP_API void [ApplicationTerminate](#) (struct [Application_t](#) *application)
Finishes execution of the application.
- AFTERWARP_API [ApplicationWindowState](#) [ApplicationGetWindowState](#) (struct [Application_t](#) *application)
Returns current application's window state.
- AFTERWARP_API [LibraryBool](#) [ApplicationSetWindowState](#) (struct [Application_t](#) *application, [ApplicationWindowState](#) windowState)
Changes application's window state.
- AFTERWARP_API [LibraryBool](#) [ApplicationReadTextFromClipboard](#) (struct [Application_t](#) *application, [char](#) *text, [_Inout_ int32_t](#) *textSize)
- AFTERWARP_API [LibraryBool](#) [ApplicationWriteTextToClipboard](#) (struct [Application_t](#) *application, [char](#) const *text)
Attempts to write text to clipboard.
- AFTERWARP_API [AppCursor](#) [ApplicationGetCursor](#) (struct [Application_t](#) *application)
Returns current application cursor.
- AFTERWARP_API [LibraryBool](#) [ApplicationSetCursor](#) (struct [Application_t](#) *application, [AppCursor](#) cursor)
Changes application cursor.
- AFTERWARP_API [LibraryBool](#) [ApplicationCaptureMouseInput](#) (struct [Application_t](#) *application)
Captures mouse input for the application window.
- AFTERWARP_API [LibraryBool](#) [ApplicationReleaseMouseInput](#) (struct [Application_t](#) *application)
Releases mouse input for the application window.
- AFTERWARP_API [LibraryBool](#) [ApplicationMouseInputCaptured](#) (struct [Application_t](#) *application)
Indicates whether the mouse input is currently captured.
- AFTERWARP_API [LibraryBool](#) [ApplicationSetIcons](#) (struct [Application_t](#) *application, struct [Surface_t](#) *const *surfaces, [int32_t](#) count)
- AFTERWARP_API [LibraryBool](#) [ApplicationSetIconsFromFiles](#) (struct [Application_t](#) *application, [char](#) const *const *fileNames, [int32_t](#) fileNameCount)
- AFTERWARP_API [LibraryBool](#) [ApplicationFileChooserDialog](#) (struct [Application_t](#) *application, [_Inout_ char](#) *filePath, [_Inout_ int32_t](#) *filePathLength, [FileChooserDialog](#) dialog, [char](#) const *filters, [char](#) const *caption, [char](#) const *accept, [char](#) const *reject)
- AFTERWARP_API struct [ActorCamera_t](#) * [ActorCameraCreate](#) ([Vector](#) const *position, [Vector](#) const *rotation, float distance)

- AFTERWARP_API void [ActorCameraDestroy](#) (struct [ActorCamera_t](#) *camera)
Releases given instance of camera module.
- AFTERWARP_API void [ActorCameraGetForward](#) (struct [ActorCamera_t](#) *camera, [Vector](#) *forward)
Returns camera's forward vector (looking direction).
- AFTERWARP_API void [ActorCameraGetRight](#) (struct [ActorCamera_t](#) *camera, [Vector](#) *right)
Returns camera's right vector (sideway direction).
- AFTERWARP_API void [ActorCameraGetCeiling](#) (struct [ActorCamera_t](#) *camera, [Vector](#) *ceiling)
Returns camera's ceiling vector (up direction).
- AFTERWARP_API void [ActorCameraGetPosition](#) (struct [ActorCamera_t](#) *camera, [Vector](#) *position)
Returns camera's position.
- AFTERWARP_API [LibraryBool](#) [ActorCameraSetPosition](#) (struct [ActorCamera_t](#) *camera, [Vector](#) const *position)
Updates camera's position. Returns true if position has actually changed.
- AFTERWARP_API void [ActorCameraGetRotation](#) (struct [ActorCamera_t](#) *camera, [Vector4](#) *rotation)
Returns current camera rotation. "w" component is a rotation for Lenticular printing.
- AFTERWARP_API [LibraryBool](#) [ActorCameraSetRotation](#) (struct [ActorCamera_t](#) *camera, [Vector4](#) const *rotation)
Updates camera's rotation. Returns true if rotation has actually changed.
- AFTERWARP_API float [ActorCameraGetDistance](#) (struct [ActorCamera_t](#) *camera)
Returns the distance between camera's position and the viewer (for third-person view).
- AFTERWARP_API void [ActorCameraSetDistance](#) (struct [ActorCamera_t](#) *camera, float distance)
Sets the distance between camera's position and the viewer (for third-person view).
- AFTERWARP_API void [ActorCameraGetQuaternion](#) (struct [ActorCamera_t](#) *camera, [Quaternion](#) *quaternion)
Returns current camera rotation as quaternion.
- AFTERWARP_API [LibraryBool](#) [ActorCameraSetQuaternion](#) (struct [ActorCamera_t](#) *camera, [Quaternion](#) const *quaternion)
Changes current camera's rotation as quaternion. Returns true if rotation has actually changed.
- AFTERWARP_API void [ActorCameraGetView](#) (struct [ActorCamera_t](#) *camera, [Matrix](#) *view)
Returns camera's resulting 3D view matrix.
- AFTERWARP_API void [ActorCameraZoom](#) (struct [ActorCamera_t](#) *camera, [PointF](#) const *position, [PointF](#) const *size, [Matrix](#) const *projection, float delta)
Changes camera's zoom by adjusting its distance and position around the specified point.
- AFTERWARP_API void [ActorCameraZoomOrtho](#) (struct [ActorCamera_t](#) *camera, [PointF](#) const *position, [PointF](#) const *size, float distance, float adjustedDistance, [Matrix](#) const *projection, [Matrix](#) const *adjustedProjection)
Adjusts camera position while performing orthographic zoom.
- AFTERWARP_API void [ActorCameraGetConstraints](#) (struct [ActorCamera_t](#) *camera, [CameraConstraints](#) *constraints)
Returns current 3D camera constraints.
- AFTERWARP_API void [ActorCameraSetConstraints](#) (struct [ActorCamera_t](#) *camera, [CameraConstraints](#) const *constraints)
Sets new 3D camera constraints.
- AFTERWARP_API [LibraryBool](#) [ActorCameraCommand](#) (struct [ActorCamera_t](#) *camera, [CameraCommand](#) command, [Vector](#) const *position, [PointF](#) const *size, [Vector](#) const *sense, [Matrix](#) const *projection)
- AFTERWARP_API struct [Timer_t](#) * [TimerCreate](#) (void)
- AFTERWARP_API void [TimerDestroy](#) (struct [Timer_t](#) *timer)
Releases given timer instance.
- AFTERWARP_API double [TimerGetSpeed](#) (struct [Timer_t](#) *timer)
Returns currently set multimedia timer speed (in terms of frames per second).
- AFTERWARP_API void [TimerSetSpeed](#) (struct [Timer_t](#) *timer, double speed)
- AFTERWARP_API [LibraryBool](#) [TimerNextSlice](#) (struct [Timer_t](#) *timer)
- AFTERWARP_API [LibraryBool](#) [TimerUpdateNextSlice](#) (struct [Timer_t](#) *timer)

- AFTERWARP_API void [TimerUpdate](#) (struct [Timer_t](#) *timer)
Updates latency, calculates average frame rate and updates fixed-rate token counter.
- AFTERWARP_API void [TimerReset](#) (struct [Timer_t](#) *timer)
- AFTERWARP_API float [TimerGetFrameRate](#) (struct [Timer_t](#) *timer)
- AFTERWARP_API uint64_t [TimerGetLatency](#) (struct [Timer_t](#) *timer)
- AFTERWARP_API int64_t [TimerGetTimeSlice](#) (struct [Timer_t](#) *timer)
Returns current time slice.
- AFTERWARP_API int64_t [TimerGetSkippedTimeSlices](#) (struct [Timer_t](#) *timer)
Returns total number of skipped time slices.
- AFTERWARP_API void [TimerTrimSkippedTimeSlices](#) (struct [Timer_t](#) *timer, int64_t slicesToTrim)
- AFTERWARP_API int32_t [TimerExtractTokens](#) (struct [Timer_t](#) *timer)
Extracts existing fixed-rate token counter from the timer.
- AFTERWARP_API struct [Widget_t](#) * [WidgetCreate](#) (struct [Widget_t](#) *manager, char const *className, char const *instanceName, [ObjectPayload](#) payload)
- AFTERWARP_API struct [Widget_t](#) * [WidgetManagerCreate](#) (struct [Application_t](#) *application, struct [TextRenderer_t](#) *textRenderer, char const *instanceName, [ObjectPayload](#) payload, [PixelFormat](#) format, int32_t multisamples, uint32_t attributes)
- AFTERWARP_API void [WidgetDestroy](#) (struct [Widget_t](#) *widget)
Releases an existing widget instance.
- AFTERWARP_API struct [Widget_t](#) * [WidgetGetManager](#) (struct [Widget_t](#) *widget)
Returns manager associated with the widget.
- AFTERWARP_API [ObjectPayload](#) [WidgetGetPayload](#) (struct [Widget_t](#) *widget)
Returns widget's payload.
- AFTERWARP_API struct [Widget_t](#) * [WidgetGetParent](#) (struct [Widget_t](#) *widget)
Returns widget's parent.
- AFTERWARP_API [LibraryBool](#) [WidgetSetParent](#) (struct [Widget_t](#) *widget, struct [Widget_t](#) *parent)
- AFTERWARP_API void [WidgetLocalToScreen](#) (struct [Widget_t](#) *widget, [PointF](#) *screenPos, [PointF](#) const *localPos)
Converts local widget position to screen coordinates.
- AFTERWARP_API void [WidgetScreenToLocal](#) (struct [Widget_t](#) *widget, [PointF](#) *localPos, [PointF](#) const *screenPos)
Converts screen coordinates to local widget position.
- AFTERWARP_API void [WidgetBringToFront](#) (struct [Widget_t](#) *widget)
Puts the widget on front of other widgets.
- AFTERWARP_API void [WidgetSendToBack](#) (struct [Widget_t](#) *widget)
Puts the widget behind all other widgets.
- AFTERWARP_API void [WidgetInvalidate](#) (struct [Widget_t](#) *widget)
Schedules widget repaint during next update cycle.
- AFTERWARP_API void [WidgetAccomodate](#) (struct [Widget_t](#) *widget)
Schedules widget repositioning during next update cycle.
- AFTERWARP_API [LibraryBool](#) [WidgetUpdate](#) (struct [Widget_t](#) *widget, double latency)
- AFTERWARP_API void [WidgetAcceptMouse](#) (struct [Widget_t](#) *widget, [MouseEvent](#) event, [MouseButton](#) button, [PointF](#) const *position)
Makes the widget receive and process mouse input.
- AFTERWARP_API void [WidgetAcceptKey](#) (struct [Widget_t](#) *widget, [KeyEvent](#) event, [Key](#) key, [UnicodeChar](#) charCode)
Makes the widget receive and process keyboard input.
- AFTERWARP_API int32_t [WidgetGetChildCount](#) (struct [Widget_t](#) *widget)
Returns number of existing widget's children.
- AFTERWARP_API struct [Widget_t](#) * [WidgetGetChildByIndex](#) (struct [Widget_t](#) *widget, int32_t index)
Returns an existing widget's child with the given index.
- AFTERWARP_API int32_t [WidgetGetChildIndex](#) (struct [Widget_t](#) *widget, struct [Widget_t](#) *child)

- AFTERWARP_API struct [Widget_t](#) * [WidgetFindAt](#) (struct [Widget_t](#) *widget, [PointF](#) const *position)
- AFTERWARP_API [LibraryBool](#) [WidgetInvokeEvent](#) (struct [Widget_t](#) *widget, char const *eventName)
Invokes a widget's event.
- AFTERWARP_API void [WidgetSetExternalEvent](#) (struct [Widget_t](#) *widget, [WidgetExternalEvent](#) event, void *user)
Specifies a new external event callback.
- AFTERWARP_API void [WidgetGetExternalEvent](#) (struct [Widget_t](#) *widget, [WidgetExternalEvent](#) *event, void **user)
Retrieves currently set external event callback.
- AFTERWARP_API [LibraryBool](#) [WidgetSetProperty](#) (struct [Widget_t](#) *widget, char const *propertyName, [WidgetPropertyType](#) valueType, void const *value, [LibraryBool](#) invokeChanged)
- AFTERWARP_API [LibraryBool](#) [WidgetGetProperty](#) (struct [Widget_t](#) *widget, char const *propertyName, [WidgetPropertyType](#) valueType, void *value)
- AFTERWARP_API [LibraryBool](#) [WidgetGetPropertyAsString](#) (struct [Widget_t](#) *widget, char const *property↵Name, char *value, [_Inout_int32_t](#) *valueLength)
- AFTERWARP_API [int32_t](#) [WidgetGetPropertyCount](#) (struct [Widget_t](#) *widget)
Returns number of existing properties.
- AFTERWARP_API [LibraryBool](#) [WidgetPropertyIdentify](#) (struct [Widget_t](#) *widget, [_Inout_ WidgetProperty](#) *property)
- AFTERWARP_API struct [Application_t](#) * [WidgetManagerGetApplication](#) (struct [Widget_t](#) *widget)
Returns application associated with the widget manager.
- AFTERWARP_API struct [TextRenderer_t](#) * [WidgetManagerGetTextRenderer](#) (struct [Widget_t](#) *widget)
Returns application associated with the widget manager.
- AFTERWARP_API struct [Widget_t](#) * [WidgetManagerGetWidgetByName](#) (struct [Widget_t](#) *widget, char const *name)
Returns an existing widget with the given name (case-insensitive).
- AFTERWARP_API struct [Widget_t](#) * [WidgetManagerGetWidgetByPayload](#) (struct [Widget_t](#) *widget, [ObjectPayload](#) payload)
Returns an existing widget with the given payload.
- AFTERWARP_API [LibraryBool](#) [WidgetManagerPresent](#) (struct [Widget_t](#) *widget)
Renders widget manager's area and all controls within.
- AFTERWARP_API struct [Texture_t](#) * [WidgetManagerGetTexture](#) (struct [Widget_t](#) *widget, [WidgetManagerTextureType](#) textureType)
Returns one of textures used for the composition.
- AFTERWARP_API [int32_t](#) [WidgetManagerBatchCount](#) (struct [Widget_t](#) *widget)
Returns total number of canvas batches during UI rendering.
- AFTERWARP_API [PixelFormat](#) [WidgetManagerGetFormat](#) (struct [Widget_t](#) *widget)
Returns pixel format of composition textures.
- AFTERWARP_API [int32_t](#) [WidgetManagerGetMultisamples](#) (struct [Widget_t](#) *widget)
Returns multisamples of rendering composition texture.
- AFTERWARP_API [uint32_t](#) [WidgetManagerGetAttributes](#) (struct [Widget_t](#) *widget)
Returns attributes used when the manager was created.
- AFTERWARP_API void [RandomSequenceInit](#) ([_Out_uint64_t](#) *state)
Initializes random number generator state using local time.
- AFTERWARP_API void [RandomSequenceInitBySeed](#) ([_Out_uint64_t](#) *state, [uint64_t](#) seed)
- AFTERWARP_API [uint32_t](#) [RandomSequenceGenerate](#) ([_Inout_uint64_t](#) *state)
Generates next random number in the sequence from the specified context.
- AFTERWARP_API [uint64_t](#) [RandomSequenceGenerate64](#) ([_Inout_uint64_t](#) *state)
Returns 64-bit random number by generating two 32-bit numbers.
- AFTERWARP_API float [RandomSequenceGenerateFloat](#) ([_Inout_uint64_t](#) *state)
Generates a random value in range [0, 1) with 23 bits of precision, using relatively linear distribution.
- AFTERWARP_API double [RandomSequenceGenerateDouble](#) ([_Inout_uint64_t](#) *state)

- Generates a random value in range [0, 1) with 53 bits of precision, using relatively linear distribution.*
- AFTERWARP_API int32_t [RandomSequenceGenerateRanged](#) (_Inout_ uint64_t *state, int32_t range)
- Generates a random value in range [0, Range) using relatively linear distribution.*
- AFTERWARP_API float [RandomSequenceGenerateGaussian](#) (_Inout_ uint64_t *state)
- Generates a random value using Gaussian distribution with mean 0 and standard deviation of 1.*
- AFTERWARP_API void [RayCreate](#) (Vector *origin, Vector *direction, PointF const *position, PointF const *surfaceSize, Matrix const *viewInverse, Matrix const *projection)
- AFTERWARP_API LibraryBool [RayIntersectTriangle](#) (Vector const *origin, Vector const *direction, Vector const *vertex1, Vector const *vertex2, Vector const *vertex3, LibraryBool backFacing, PointF *intersection, float *distance)
- AFTERWARP_API LibraryBool [RayIntersectCubeVolume](#) (Vector const *origin, Vector const *direction, Matrix const *world, float *distance)
- AFTERWARP_API LibraryBool [RayIntersectPlane](#) (Vector const *origin, Vector const *direction, Vector const *planePoint, Vector const *planeNormal, Vector *intersection, float *distance)
- AFTERWARP_API LibraryBool [PixelFormatValid](#) (PixelFormat format)
- Verifies that given pixel format is well-defined, known and valid.*
- AFTERWARP_API uint32_t [PixelFormatBits](#) (PixelFormat format)
- Returns number of bits that each pixel in current format occupies.*
- AFTERWARP_API void [PixelFormatConvert](#) (void *dest, void const *source, PixelFormat destFormat, PixelFormat sourceFormat)
- Converts a single pixel from one pixel format to another preserving as much information as possible.*
- AFTERWARP_API void [PixelFormatConvertArray](#) (void *dest, void const *source, PixelFormat destFormat, PixelFormat sourceFormat, uint32_t destPitch, uint32_t sourcePitch, int32_t width, int32_t height)
- Converts an array of pixels from one pixel format to another preserving as much information as possible.*
- AFTERWARP_API Color [MakeColor](#) (Color color, int32_t alpha)
- AFTERWARP_API Color [MakeColorF](#) (Color color, float alpha)
- AFTERWARP_API Color [MakeColorWithGray](#) (Color color, int32_t gray, int32_t alpha)
- AFTERWARP_API Color [MakeColorWithGrayF](#) (Color color, float gray, float alpha)
- AFTERWARP_API Color [MakeColorRGB](#) (int32_t red, int32_t green, int32_t blue, int32_t alpha)
- AFTERWARP_API Color [MakeColorRGBF](#) (float red, float green, float blue, float alpha)
- AFTERWARP_API Color [MakeColorGray](#) (int32_t gray, int32_t alpha)
- AFTERWARP_API Color [MakeColorGrayF](#) (float gray, float alpha)
- AFTERWARP_API Color [MakeColorAlpha](#) (int32_t alpha)
- AFTERWARP_API Color [MakeColorAlphaF](#) (float alpha)
- AFTERWARP_API int32_t [GetColorAlpha](#) (Color color)
- Returns alpha-channel in range [0, 255] for the given 32-bit RGBA color.*
- AFTERWARP_API float [GetColorAlphaF](#) (Color color)
- Returns alpha-channel in range [0, 1] for the given 32-bit RGBA color.*
- AFTERWARP_API Color [PremultiplyAlpha](#) (Color color)
- AFTERWARP_API Color [UnpremultiplyAlpha](#) (Color color)
- AFTERWARP_API Color [DisplaceRB](#) (Color color)
- Switches red and blue channels in 32-bit RGBA color value.*
- AFTERWARP_API Color [InvertColor](#) (Color color)
- Inverts each of the components in the color, including alpha-channel.*
- AFTERWARP_API Color [AddColors](#) (Color color1, Color color2)
- Adds two 32-bit RGBA color values together clamping the resulting values.*
- AFTERWARP_API Color [SubtractColors](#) (Color color1, Color color2)
- Subtracts two 32-bit RGBA color values clamping the resulting values.*
- AFTERWARP_API Color [MultiplyColors](#) (Color color1, Color color2)
- Multiplies two 32-bit RGBA color values together.*
- AFTERWARP_API Color [AverageColors](#) (Color color1, Color color2)
- Computes average of two given 32-bit RGBA color values.*
- AFTERWARP_API Color [AverageFourColors](#) (Color color1, Color color2, Color color3, Color color4)

Computes average of four given 32-bit RGBA color values.

- AFTERWARP_API [Color AverageSixColors](#) ([Color](#) color1, [Color](#) color2, [Color](#) color3, [Color](#) color4, [Color](#) color5, [Color](#) color6)

Computes average of six given 32-bit RGBA color values.

- AFTERWARP_API [Color BlendColors](#) ([Color](#) color1, [Color](#) color2, [int32_t](#) alpha)
- AFTERWARP_API [Color BlendFourColors](#) ([Color](#) topLeft, [Color](#) topRight, [Color](#) bottomRight, [Color](#) bottomLeft, [int32_t](#) alphaX, [int32_t](#) alphaY, [LibraryBool](#) horizontal)
- AFTERWARP_API [Color ComposeColors](#) ([Color](#) source, [Color](#) dest)
- AFTERWARP_API [int32_t ColorToGray](#) ([Color](#) color)
- AFTERWARP_API [int32_t ColorToGray16](#) ([Color](#) color)
- AFTERWARP_API [float ColorToGrayF](#) ([Color](#) color)
- AFTERWARP_API [float Lerp](#) ([float](#) value1, [float](#) value2, [float](#) theta)

Interpolates between two values linearly, where theta parameter is specified in [0, 1] range.

- AFTERWARP_API [float Cubic](#) ([float](#) predValue1, [float](#) value1, [float](#) value2, [float](#) succValue2, [float](#) theta)

Interpolates between four values using cubic spline, where theta parameter is specified in [0, 1] range.

- AFTERWARP_API [float CatmullRom](#) ([float](#) predValue1, [float](#) value1, [float](#) value2, [float](#) succValue2, [float](#) theta)
- AFTERWARP_API [float Hermite](#) ([float](#) predValue1, [float](#) value1, [float](#) value2, [float](#) succValue2, [float](#) theta, [float](#) tension, [float](#) bias)
- AFTERWARP_API [float SmoothStep](#) ([float](#) value1, [float](#) value2, [float](#) theta)
- AFTERWARP_API [float SmootherStep](#) ([float](#) value1, [float](#) value2, [float](#) theta)
- AFTERWARP_API [float SineTransform](#) ([float](#) value)

Transforms value in range of [0, 1] by using sine wave (kind of "accelerate then decelerate").

- AFTERWARP_API [float SineAccelerate](#) ([float](#) value)
- AFTERWARP_API [float SineDecelerate](#) ([float](#) value)
- AFTERWARP_API [float SineCycle](#) ([float](#) value)
- AFTERWARP_API [float SineTwoCycle](#) ([float](#) value)

Transforms value in range of [0, 1] to go full cycle through sine function (0 -> 1 -> 0 -> -1 -> 0).

- AFTERWARP_API [float BiasTransform](#) ([float](#) value, [float](#) bias)
- AFTERWARP_API [float GainTransform](#) ([float](#) value, [float](#) gain)
- AFTERWARP_API [void MeshBoundsToMatrixModel](#) ([Matrix](#) *model, [Vector](#) const *minBounds, [Vector](#) const *maxBounds, [MeshAligns](#) const *aligns, [float](#) scale)

Creates a model transformation matrix that centers the mesh and places it on top of zero plane.

- AFTERWARP_API [void MeshBoundsToMatrixVolume](#) ([Matrix](#) *volume, [Vector](#) const *minBounds, [Vector](#) const *maxBounds, [MeshAligns](#) const *aligns, [float](#) scale)

Creates a volume transformation matrix that centers the mesh and places it on top of zero plane.

- AFTERWARP_API [void MeshBoundsTagOffset](#) ([Vector](#) const *meshMinBounds, [Vector](#) const *meshMaxBounds, [Vector](#) const *tagMinBounds, [Vector](#) const *tagMaxBounds, [Vector](#) *offset)

Calculates an offset to displace from mesh to tag origin.

- AFTERWARP_API [void MeshBoundsToMatrixVolumeTag](#) ([Vector](#) const *meshMinBounds, [Vector](#) const *meshMaxBounds, [Vector](#) const *tagMinBounds, [Vector](#) const *tagMaxBounds, [Matrix](#) *volume, [float](#) size, [float](#) Bias)

Creates a volume transformation matrix that centers the mesh tag inside mesh volume.

- AFTERWARP_API [void VolumeCalculateNearFarPlanes](#) ([Matrix](#) const *worldView, [float](#) *nearPlane, [float](#) *farPlane)

Calculates near and far planes for a bounding box transformed by the given world/view matrix.

- AFTERWARP_API [void VolumeCalculateVisibleFrame](#) ([Matrix](#) const *worldViewProjection, [PointF](#) const *surfaceSize, [RectF](#) *visibleFrame)

Calculates visible surface frame for a bounding box transformed by the given world/view/projection matrix.

- AFTERWARP_API [uint32_t VertexElementsEstimatePitch](#) ([uint32_t](#) channel, [VertexElement](#) const vertexElements[], [int32_t](#) vertexElementCount)

Makes an estimation of pitch for the given channel based on existing element offsets.

- AFTERWARP_API [uint64_t GetSystemTicks](#) ([void](#))

Retrieves number of microseconds that passed since application startup.

7.1.1 Function Documentation

7.1.1.1 ActorCameraCommand()

```
AFTERWARP_API LibraryBool ActorCameraCommand (
    struct ActorCamera_t * camera,
    CameraCommand command,
    Vector const * position,
    PointF const * size,
    Vector const * sense,
    Matrix const * projection )
```

Issue a camera command for rotation/movement. Returns true if camera's position or rotation has changed. For movement, "sense" defines the normal of the movement plane and "position.z" its altitude. For dragging and rotation, "sense" defines sensitivity for X and Y axes respectively (Z is ignored). Projection matrix is only used for movement command.

7.1.1.2 ActorCameraCreate()

```
AFTERWARP_API struct ActorCamera_t * ActorCameraCreate (
    Vector const * position,
    Vector const * rotation,
    float distance )
```

Creates a new instance of a camera module that can represent both first-person and third-person views for 3D world exploration and object manipulation.

7.1.1.3 ActorCameraDestroy()

```
AFTERWARP_API void ActorCameraDestroy (
    struct ActorCamera_t * camera )
```

Releases given instance of camera module.

7.1.1.4 ActorCameraGetCeiling()

```
AFTERWARP_API void ActorCameraGetCeiling (
    struct ActorCamera_t * camera,
    Vector * ceiling )
```

Returns camera's ceiling vector (up direction).

7.1.1.5 ActorCameraGetConstraints()

```
AFTERWARP_API void ActorCameraGetConstraints (
    struct ActorCamera_t * camera,
    CameraConstraints * constraints )
```

Returns current 3D camera constraints.

7.1.1.6 ActorCameraGetDistance()

```
AFTERWARP_API float ActorCameraGetDistance (
    struct ActorCamera_t * camera )
```

Returns the distance between camera's position and the viewer (for third-person view).

7.1.1.7 ActorCameraGetForward()

```
AFTERWARP_API void ActorCameraGetForward (
    struct ActorCamera_t * camera,
    Vector * forward )
```

Returns camera's forward vector (looking direction).

7.1.1.8 ActorCameraGetPosition()

```
AFTERWARP_API void ActorCameraGetPosition (
    struct ActorCamera_t * camera,
    Vector * position )
```

Returns camera's position.

7.1.1.9 ActorCameraGetQuaternion()

```
AFTERWARP_API void ActorCameraGetQuaternion (
    struct ActorCamera_t * camera,
    Quaternion * quaternion )
```

Returns current camera rotation as quaternion.

7.1.1.10 ActorCameraGetRight()

```
AFTERWARP_API void ActorCameraGetRight (
    struct ActorCamera_t * camera,
    Vector * right )
```

Returns camera's right vector (sideway direction).

7.1.1.11 ActorCameraGetRotation()

```
AFTERWARP_API void ActorCameraGetRotation (
    struct ActorCamera_t * camera,
    Vector4 * rotation )
```

Returns current camera rotation. "w" component is a rotation for Lenticular printing.

7.1.1.12 ActorCameraGetView()

```
AFTERWARP_API void ActorCameraGetView (
    struct ActorCamera_t * camera,
    Matrix * view )
```

Returns camera's resulting 3D view matrix.

7.1.1.13 ActorCameraSetConstraints()

```
AFTERWARP_API void ActorCameraSetConstraints (
    struct ActorCamera_t * camera,
    CameraConstraints const * constraints )
```

Sets new 3D camera constraints.

7.1.1.14 ActorCameraSetDistance()

```
AFTERWARP_API void ActorCameraSetDistance (
    struct ActorCamera_t * camera,
    float distance )
```

Sets the distance between camera's position and the viewer (for third-person view).

7.1.1.15 ActorCameraSetPosition()

```
AFTERWARP_API LibraryBool ActorCameraSetPosition (
    struct ActorCamera_t * camera,
    Vector const * position )
```

Updates camera's position. Returns `true` if position has actually changed.

7.1.1.16 ActorCameraSetQuaternion()

```
AFTERWARP_API LibraryBool ActorCameraSetQuaternion (
    struct ActorCamera_t * camera,
    Quaternion const * quaternion )
```

Changes current camera's rotation as quaternion. Returns `true` if rotation has actually changed.

7.1.1.17 ActorCameraSetRotation()

```
AFTERWARP_API LibraryBool ActorCameraSetRotation (
    struct ActorCamera_t * camera,
    Vector4 const * rotation )
```

Sets new camera rotation. "w" component is a rotation for Lenticular printing. Returns `true` if rotation has actually changed.

7.1.1.18 ActorCameraZoom()

```
AFTERWARP_API void ActorCameraZoom (
    struct ActorCamera_t * camera,
    PointF const * position,
    PointF const * size,
    Matrix const * projection,
    float delta )
```

Changes camera's zoom by adjusting its distance and position around the specified point.

7.1.1.19 ActorCameraZoomOrtho()

```
AFTERWARP_API void ActorCameraZoomOrtho (
    struct ActorCamera_t * camera,
    PointF const * position,
    PointF const * size,
    float distance,
    float adjustedDistance,
    Matrix const * projection,
    Matrix const * adjustedProjection )
```

Adjusts camera position while performing orthographic zoom.

7.1.1.20 AddColors()

```
AFTERWARP_API Color AddColors (
    Color color1,
    Color color2 )
```

Adds two 32-bit RGBA color values together clamping the resulting values.

7.1.1.21 ApplicationCaptureMouseInput()

```
AFTERWARP_API LibraryBool ApplicationCaptureMouseInput (
    struct Application_t * application )
```

Captures mouse input for the application window.

7.1.1.22 ApplicationConvertPortableKey()

```
AFTERWARP_API int32_t ApplicationConvertPortableKey (
    struct Application_t * application,
    Key key )
```

Converts portable key code to virtual key code. Returns 0 if no equivalent is found.

7.1.1.23 ApplicationCreate()

```
AFTERWARP_API struct Application_t * ApplicationCreate (
    _Inout_ ApplicationConfiguration * configuration )
```

Creates a new application instance.

7.1.1.24 ApplicationDestroy()

```
AFTERWARP_API void ApplicationDestroy (
    struct Application_t * application )
```

Releases the given application instance.

7.1.1.25 ApplicationExecute()

```
AFTERWARP_API LibraryBool ApplicationExecute (
    struct Application_t * application )
```

Executes application's main loop (or returns "false" when there were errors during application create event).

7.1.1.26 ApplicationFileChooserDialog()

```
AFTERWARP_API LibraryBool ApplicationFileChooserDialog (
    struct Application_t * application,
    _Inout_ char * filePath,
    _Inout_ int32_t * filePathLength,
    FileChooserDialog dialog,
    char const * filters,
    char const * caption,
    char const * accept,
    char const * reject )
```

Shows a file chooser dialog to pick a file for loading or saving. Starting file path, as well as suggested file name are taken from "filePath". "filters" is a list of one or more filter, specified by its name and extension wildcard, separated by semicolon. Multiple filters are separated by '|'. An example filter would be: Wavefront OBJ files (*.obj);*.obj|All files (*.*)*.*

7.1.1.27 ApplicationGetClientRect()

```
AFTERWARP_API void ApplicationGetClientRect (
    struct Application_t * application,
    Rect * rect )
```

Retrieves current application's window client rectangle.

7.1.1.28 ApplicationGetCursor()

```
AFTERWARP_API AppCursor ApplicationGetCursor (
    struct Application_t * application )
```

Returns current application cursor.

7.1.1.29 ApplicationGetExecutablePath()

```
AFTERWARP_API void ApplicationGetExecutablePath (
    struct Application_t * application,
    char * path,
    _Inout_ int32_t * pathLength )
```

Retrieves current application's execution path. The pointed value of `pathLength` defines the maximum number of characters, including null-terminating character, that can be copied. The actual string length is stored in `pathLength`.

7.1.1.30 ApplicationGetIconTitle()

```
AFTERWARP_API void ApplicationGetIconTitle (
    struct Application_t * application,
    char * iconTitle,
    _Inout_ int32_t * iconTitleLength )
```

Retrieves current application icon title. The pointed value of `iconTitleLength` defines the maximum number of characters, including null-terminating character, that can be copied. The actual string length is stored in `iconTitleLength`.

7.1.1.31 ApplicationGetMinimalSize()

```
AFTERWARP_API void ApplicationGetMinimalSize (
    struct Application_t * application,
    Point * size )
```

Returns current minimal size of the window. Values of zero means that there is no limit.

7.1.1.32 ApplicationGetTitle()

```
AFTERWARP_API void ApplicationGetTitle (
    struct Application_t * application,
    char * title,
    _Inout_ int32_t * titleLength )
```

Retrieves current application title. The pointed value of `titleLength` defines the maximum number of characters, including null-terminating character, that can be copied. The actual string length is stored in `titleLength`.

7.1.1.33 ApplicationGetWindowHandle()

```
AFTERWARP_API UntypedHandle ApplicationGetWindowHandle (
    struct Application_t * application )
```

Retrieves the handle of main application window.

7.1.1.34 ApplicationGetWindowRect()

```
AFTERWARP_API void ApplicationGetWindowRect (
    struct Application_t * application,
    Rect * rect )
```

Retrieves current application's window rectangle.

7.1.1.35 ApplicationGetWindowScale()

```
AFTERWARP_API double ApplicationGetWindowScale (
    struct Application_t * application )
```

Returns the scale of application window.

7.1.1.36 ApplicationGetWindowState()

```
AFTERWARP_API ApplicationWindowState ApplicationGetWindowState (
    struct Application_t * application )
```

Returns current application's window state.

7.1.1.37 ApplicationInvalidate()

```
AFTERWARP_API void ApplicationInvalidate (
    struct Application_t * application )
```

Requests the application window to be repainted.

7.1.1.38 ApplicationMouseInputCaptured()

```
AFTERWARP_API LibraryBool ApplicationMouseInputCaptured (
    struct Application_t * application )
```

Indicates whether the mouse input is currently captured.

7.1.1.39 ApplicationReadTextFromClipboard()

```
AFTERWARP_API LibraryBool ApplicationReadTextFromClipboard (
    struct Application_t * application,
    char * text,
    _Inout_ int32_t * textSize )
```

Attempts to read text from clipboard. Writes up to `textSize` number of characters, including null-terminating character to the destination string. Updates `textSize` to reflect the actual number of characters written.

7.1.1.40 ApplicationReleaseMouseInput()

```
AFTERWARP_API LibraryBool ApplicationReleaseMouseInput (
    struct Application_t * application )
```

Releases mouse input for the application window.

7.1.1.41 ApplicationSetClientSize()

```
AFTERWARP_API void ApplicationSetClientSize (
    struct Application_t * application,
    Point const * size )
```

Sets new application's window client size.

7.1.1.42 ApplicationSetCursor()

```
AFTERWARP_API LibraryBool ApplicationSetCursor (
    struct Application_t * application,
    AppCursor cursor )
```

Changes application cursor.

7.1.1.43 ApplicationSetEvents()

```
AFTERWARP_API void ApplicationSetEvents (
    struct Application_t * application,
    ApplicationEvents const * events,
    void * user )
```

Sets events that will be invoked by the application. If all events are set to NULL, or `events` is set to NULL, then the appropriate events will use default handlers.

7.1.1.44 ApplicationSetIcons()

```
AFTERWARP_API LibraryBool ApplicationSetIcons (
    struct Application_t * application,
    struct Surface_t *const * surfaces,
    int32_t count )
```

Specifies one or more application icons. Note: the provided array including each of the surfaces should be released by the caller.

7.1.1.45 ApplicationSetIconsFromFiles()

```
AFTERWARP_API LibraryBool ApplicationSetIconsFromFiles (
    struct Application_t * application,
    char const *const * fileNames,
    int32_t fileNameCount )
```

Changes application icons by loading them from external files. Note: this method will only work while at least one device instance currently exists.

7.1.1.46 ApplicationSetIconTitle()

```
AFTERWARP_API LibraryBool ApplicationSetIconTitle (
    struct Application_t * application,
    char const * iconTitle )
```

Sets new application's icon title.

7.1.1.47 ApplicationSetMinimalSize()

```
AFTERWARP_API void ApplicationSetMinimalSize (
    struct Application_t * application,
    Point const * size )
```

Sets a minimal size of the window. Values of zero means that there is no limit.

7.1.1.48 ApplicationSetTitle()

```
AFTERWARP_API LibraryBool ApplicationSetTitle (
    struct Application_t * application,
    char const * title )
```

Sets new application title.

7.1.1.49 ApplicationSetWindowRect()

```
AFTERWARP_API void ApplicationSetWindowRect (
    struct Application_t * application,
    Rect const * rect )
```

Sets new application's window rectangle.

7.1.1.50 ApplicationSetWindowState()

```
AFTERWARP_API LibraryBool ApplicationSetWindowState (
    struct Application_t * application,
    ApplicationWindowState windowState )
```

Changes application's window state.

7.1.1.51 ApplicationTerminate()

```
AFTERWARP_API void ApplicationTerminate (
    struct Application_t * application )
```

Finishes execution of the application.

7.1.1.52 ApplicationTranslateVirtualKey()

```
AFTERWARP_API Key ApplicationTranslateVirtualKey (
    struct Application_t * application,
    int32_t virtualKey )
```

Converts virtual key code to portable key code. Returns Key::Null if no equivalent is found.

7.1.1.53 ApplicationWriteTextToClipboard()

```
AFTERWARP_API LibraryBool ApplicationWriteTextToClipboard (
    struct Application_t * application,
    char const * text )
```

Attempts to write text to clipboard.

7.1.1.54 AverageColors()

```
AFTERWARP_API Color AverageColors (
    Color color1,
    Color color2 )
```

Computes average of two given 32-bit RGBA color values.

7.1.1.55 AverageFourColors()

```
AFTERWARP_API Color AverageFourColors (
    Color color1,
    Color color2,
    Color color3,
    Color color4 )
```

Computes average of four given 32-bit RGBA color values.

7.1.1.56 AverageSixColors()

```
AFTERWARP_API Color AverageSixColors (
    Color color1,
    Color color2,
    Color color3,
    Color color4,
    Color color5,
    Color color6 )
```

Computes average of six given 32-bit RGBA color values.

7.1.1.57 BiasTransform()

```
AFTERWARP_API float BiasTransform (
    float value,
    float bias )
```

Transforms value in range of [0, 1] with a power function, adding bias to either start of the curve or its end. Bias value of 0.25 would be similar to [SineAccelerate\(\)](#), whereas value of 0.75 would be similar to that of [SineDecelerate\(\)](#).

7.1.1.58 BlendColors()

```
AFTERWARP_API Color BlendColors (
    Color color1,
    Color color2,
    int32_t alpha )
```

Computes alpha-blending for a pair of 32-bit RGBA colors values. `alpha` must be in [0..255] range.

7.1.1.59 BlendFourColors()

```
AFTERWARP_API Color BlendFourColors (
    Color topLeft,
    Color topRight,
    Color bottomRight,
    Color bottomLeft,
    int32_t alphaX,
    int32_t alphaY,
    LibraryBool horizontal )
```

Computes resulting alpha-blended value between four 32-bit RGBA colors using linear interpolation. `alphaX` and `alphaY` must be in [0..255] range. `horizontal` determines first axis to interpolate.

7.1.1.60 BufferCopy()

```
AFTERWARP_API LibraryBool BufferCopy (
    struct Buffer_t * buffer,
    struct Buffer_t * bufferSource,
    uint32_t offsetDest,
    uint32_t offsetSource,
    uint32_t size )
```

Updates the contents of the buffer with data copied from another buffer. If `size` is left at zero, the remaining portion of buffer starting at given byte offset will be copied.

7.1.1.61 BufferCreate()

```
AFTERWARP_API struct Buffer_t * BufferCreate (
    struct Device_t * device,
    BufferDataType dataType,
    BufferAccessType accessType,
    uint32_t size,
    uint32_t pitch,
    PixelFormat format,
    void const * initialData )
```

Creates a new instance of hardware buffer associated with a particular device with the given parameters and (optional) initial data.

7.1.1.62 BufferDestroy()

```
AFTERWARP_API void BufferDestroy (
    struct Buffer_t * buffer )
```

Releases given instance of hardware buffer.

7.1.1.63 BufferGetAccessType()

```
AFTERWARP_API BufferAccessType BufferGetAccessType (
    struct Buffer_t * buffer )
```

Returns the access type of for buffer data.

7.1.1.64 BufferGetDataType()

```
AFTERWARP_API BufferDataType BufferGetDataType (
    struct Buffer_t * buffer )
```

Returns the type of data that buffer contains.

7.1.1.65 BufferGetDevice()

```
AFTERWARP_API struct Device_t * BufferGetDevice (
    struct Buffer_t * buffer )
```

Returns device associated with the given buffer.

7.1.1.66 BufferGetFormat()

```
AFTERWARP_API PixelFormat BufferGetFormat (
    struct Buffer_t * buffer )
```

Returns pixel format that represents the buffer contents.

7.1.1.67 BufferGetPitch()

```
AFTERWARP_API uint32_t BufferGetPitch (
    struct Buffer_t * buffer )
```

Returns the number of bytes for offset to move for advancing the pointer from one element to another.

7.1.1.68 BufferGetPlatformHandle()

```
AFTERWARP_API UntypedHandle BufferGetPlatformHandle (
    struct Buffer_t * buffer )
```

Returns platform-specific buffer handle.

7.1.1.69 BufferGetSize()

```
AFTERWARP_API uint32_t BufferGetSize (
    struct Buffer_t * buffer )
```

Returns buffer size in bytes.

7.1.1.70 BufferRetrieve()

```
AFTERWARP_API LibraryBool BufferRetrieve (
    struct Buffer_t * buffer,
    void * data,
    uint32_t offset,
    uint32_t size )
```

Retrieves the contents of the buffer. If `size` is left at zero, the remaining portion of buffer starting at given byte offset will be retrieved.

7.1.1.71 BufferUpdate()

```
AFTERWARP_API LibraryBool BufferUpdate (
    struct Buffer_t * buffer,
    void const * data,
    uint32_t offset,
    uint32_t size )
```

Updates the contents of the buffer with a new data. If `size` is left at zero, the remaining portion of buffer starting at given byte offset will be updated. Note that if buffer has been created with an access type that is non-default, then typically a whole buffer should be updated, attempting to update a portion of it may be implementation-dependent.

7.1.1.72 CanvasArc()

```
AFTERWARP_API void CanvasArc (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * radius,
    float initAngle,
    float endAngle,
    int32_t steps,
    Color colorInside,
    Color colorOutside,
    BlendingEffect effect )
```

Draws a filled arc with the given origin, radiuses, angles, steps and color gradient going away from the center of the arc.

7.1.1.73 CanvasArcGrad()

```
AFTERWARP_API void CanvasArcGrad (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * radius,
    float initAngle,
    float endAngle,
    int32_t steps,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws a filled arc with the given origin, radiuses, angles, steps and colors.

7.1.1.74 CanvasBegin()

```
AFTERWARP_API LibraryBool CanvasBegin (
    struct Canvas_t * canvas )
```

Starts rendering with the canvas.

7.1.1.75 CanvasBufferClear()

```
AFTERWARP_API void CanvasBufferClear (
    struct CanvasBuffer_t * buffer )
```

Clears all buffers.

7.1.1.76 CanvasBufferClearAndShrink()

```
AFTERWARP_API void CanvasBufferClearAndShrink (
    struct CanvasBuffer_t * buffer )
```

Clears all buffers and releases their memory.

7.1.1.77 CanvasBufferCreate()

```
AFTERWARP_API struct CanvasBuffer_t * CanvasBufferCreate (
    void )
```

Creates a new container for vertices and indices defining one or more triangular meshes that can be suitable for rendering with a 2D canvas.

7.1.1.78 CanvasBufferDestroy()

```
AFTERWARP_API void CanvasBufferDestroy (
    struct CanvasBuffer_t * buffer )
```

Releases an existing instance of canvas buffer container.

7.1.1.79 CanvasBufferGetColors()

```
AFTERWARP_API Color * CanvasBufferGetColors (
    struct CanvasBuffer_t * buffer )
```

Returns pointer to an existing array of colors.

7.1.1.80 CanvasBufferGetExtents()

```
AFTERWARP_API void CanvasBufferGetExtents (
    struct CanvasBuffer_t * buffer,
    RectF * extents )
```

Calculates boundaries for the existing vertices.

7.1.1.81 CanvasBufferGetIndexCount()

```
AFTERWARP_API int32_t CanvasBufferGetIndexCount (
    struct CanvasBuffer_t * buffer )
```

Returns existing number of indices.

7.1.1.82 CanvasBufferGetIndices()

```
AFTERWARP_API int32_t * CanvasBufferGetIndices (
    struct CanvasBuffer_t * buffer )
```

Returns pointer to an existing array of vertex indices.

7.1.1.83 CanvasBufferGetVertexCount()

```
AFTERWARP_API int32_t CanvasBufferGetVertexCount (
    struct CanvasBuffer_t * buffer )
```

Returns existing number of vertices.

7.1.1.84 CanvasBufferGetVertices()

```
AFTERWARP_API PointF * CanvasBufferGetVertices (
    struct CanvasBuffer_t * buffer )
```

Returns pointer to an existing array of vertices.

7.1.1.85 CanvasBufferSetIndexCount()

```
AFTERWARP_API LibraryBool CanvasBufferSetIndexCount (
    struct CanvasBuffer_t * buffer,
    int32_t indexCount )
```

Adjusts the existing number of indices.

7.1.1.86 CanvasBufferSetVertexCount()

```
AFTERWARP_API LibraryBool CanvasBufferSetVertexCount (
    struct CanvasBuffer_t * buffer,
    int32_t vertexCount )
```

Adjusts the existing number of vertices.

7.1.1.87 CanvasCreate()

```
AFTERWARP_API struct Canvas_t * CanvasCreate (
    struct Device_t * device )
```

Creates new 2D rendering canvas.

7.1.1.88 CanvasDestroy()

```
AFTERWARP_API void CanvasDestroy (
    struct Canvas_t * canvas )
```

Releases given instance of 2D rendering canvas.

7.1.1.89 CanvasDrawBuffer()

```
AFTERWARP_API void CanvasDrawBuffer (
    struct Canvas_t * canvas,
    struct CanvasBuffer_t * buffer,
    BlendingEffect effect )
```

Draws triangles from an existing buffer.

7.1.1.90 CanvasEllipse()

```
AFTERWARP_API void CanvasEllipse (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * radius,
    int32_t steps,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws a filled ellipse with the given origin, radiuses, steps and colors.

7.1.1.91 CanvasEnd()

```
AFTERWARP_API void CanvasEnd (
    struct Canvas_t * canvas )
```

Finishes rendering with the canvas.

7.1.1.92 CanvasFillRect()

```
AFTERWARP_API void CanvasFillRect (
    struct Canvas_t * canvas,
    RectF const * rect,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws a filled rectangle with the given vertices and colors.

7.1.1.93 CanvasFillRoundRect()

```
AFTERWARP_API void CanvasFillRoundRect (
    struct Canvas_t * canvas,
    RectF const * rect,
    ColorRect const * colors,
    float radius,
    int32_t steps,
    BlendingEffect effect )
```

Draws a filled rectangle with the given gradient and round corners.

7.1.1.94 CanvasFillRoundRectBottom()

```
AFTERWARP_API void CanvasFillRoundRectBottom (
    struct Canvas_t * canvas,
    RectF const * rect,
    ColorRect const * colors,
    float radius,
    int32_t steps,
    BlendingEffect effect )
```

Draws a filled rectangle with the given gradient and round corners on bottom.

7.1.1.95 CanvasFillRoundRectTop()

```
AFTERWARP_API void CanvasFillRoundRectTop (
    struct Canvas_t * canvas,
    RectF const * rect,
    ColorRect const * colors,
    float radius,
    int32_t steps,
    BlendingEffect effect )
```

Draws a filled rectangle with the given gradient and round corners on top.

7.1.1.96 CanvasFillRoundRectTopInverse()

```
AFTERWARP_API void CanvasFillRoundRectTopInverse (
    struct Canvas_t * canvas,
    RectF const * rect,
    ColorRect const * colors,
    float radius,
    int32_t steps,
    BlendingEffect effect )
```

Draws a filled rectangle with the given gradient and round corners, but top is reverted. This can be used to render rounded window background behind caption.

7.1.1.97 CanvasFlush()

```
AFTERWARP_API void CanvasFlush (
    struct Canvas_t * canvas )
```

Flushes the canvas by rendering primitives that are still in batch buffers.

7.1.1.98 CanvasFrameRect()

```
AFTERWARP_API void CanvasFrameRect (
    struct Canvas_t * canvas,
    RectF const * rect,
    ColorRect const * colors,
    float thickness,
    BlendingEffect effect )
```

Draws a rectangle of varying thickness with the given vertices and colors.

7.1.1.99 CanvasFrameRoundRect()

```
AFTERWARP_API void CanvasFrameRoundRect (
    struct Canvas_t * canvas,
    RectF const * rect,
    ColorRect const * colors,
    float radius,
    float thickness,
    int32_t steps,
    BlendingEffect effect )
```

Draws a round frame for the given rectangle area with varying thickness and gradient.

7.1.1.100 CanvasGetAttributes()

```
AFTERWARP_API uint32_t CanvasGetAttributes (
    struct Canvas_t * canvas )
```

Retrieves current canvas rendering attributes.

7.1.1.101 CanvasGetBatchCount()

```
AFTERWARP_API int32_t CanvasGetBatchCount (
    struct Canvas_t * canvas )
```

Returns number of batches that were rendered between current set of begin/end block. Drawing each individual batch involves some overhead, so when this parameter happens to be considerably high at some point, the rendering code should be revised for better grouping of images, shapes and blending types.

7.1.1.102 CanvasGetClipRect()

```
AFTERWARP_API void CanvasGetClipRect (
    struct Canvas_t * canvas,
    Rect * clipRect )
```

Returns currently set clipping rectangle.

7.1.1.103 CanvasGetContextState()

```
AFTERWARP_API CanvasContextState CanvasGetContextState (
    struct Canvas_t * canvas )
```

Returns currently configured canvas context state.

7.1.1.104 CanvasGetDevice()

```
AFTERWARP_API struct Device_t * CanvasGetDevice (
    struct Canvas_t * canvas )
```

Returns device associated with the given canvas.

7.1.1.105 CanvasGetSamplerState()

```
AFTERWARP_API LibraryBool CanvasGetSamplerState (
    struct Canvas_t * canvas,
    CanvasSamplerState * samplerState )
```

Retrieves current canvas sampler state (if such exists).

7.1.1.106 CanvasGetSignedDistanceField()

```
AFTERWARP_API void CanvasGetSignedDistanceField (
    struct Canvas_t * canvas,
    SignedDistanceField * signedDistanceField )
```

Retrieves current 3D transformation (world/view/projection) matrix.

7.1.1.107 CanvasGetTransform()

```
AFTERWARP_API void CanvasGetTransform (
    struct Canvas_t * canvas,
    Matrix3x2 * transform )
```

Returns currently set 2D transformation matrix.

7.1.1.108 CanvasGetViewProjection()

```
AFTERWARP_API void CanvasGetViewProjection (
    struct Canvas_t * canvas,
    Matrix * viewProjection )
```

Retrieves current 3D view/projection transformation matrix.

7.1.1.109 CanvasGetWorld()

```
AFTERWARP_API void CanvasGetWorld (
    struct Canvas_t * canvas,
    Matrix * world )
```

Retrieves current 3D world transformation matrix.

7.1.1.110 CanvasHexagon()

```
AFTERWARP_API void CanvasHexagon (
    struct Canvas_t * canvas,
    Color color1,
    Color color2,
    Color color3,
    Color color4,
    Color color5,
    Color color6,
    BlendingEffect effect )
```

Draws hexagon of unity size centered along axis origin and filled with gradient of six colors at the corresponding vertices. The bounding width of hexagon is exactly one, which simplifies placement of multiple hexagons. The final size, position and rotation of hexagon can be given using one or a combination of several 3x2 transformation matrices multiplied together.

7.1.1.111 CanvasHexagonGrad()

```
AFTERWARP_API void CanvasHexagonGrad (
    struct Canvas_t * canvas,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws hexagon of unity size centered along axis origin and filled with four-color gradient interpolated to the corresponding vertices. The bounding width of hexagon is exactly one, which simplifies placement of multiple hexagons. The final size, position and rotation of hexagon can be given using one or a combination of several 3x2 transformation matrices multiplied together.

7.1.1.112 CanvasHighlight()

```
AFTERWARP_API void CanvasHighlight (
    struct Canvas_t * canvas,
    RectF const * rect,
    float cornerRadius,
    float luma,
    float distance,
    int32_t steps )
```

Draws either a highlight or a shadow around the specified rectangle with the given parameters, mimicking effect of gaussian highlight.

7.1.1.113 CanvasLine()

```
AFTERWARP_API void CanvasLine (
    struct Canvas_t * canvas,
    PointF const * srcPos,
    PointF const * destPos,
    ColorPair const * colors,
    BlendingEffect effect )
```

Draws a line with the given positions and colors.

7.1.1.114 CanvasLineCircle()

```
AFTERWARP_API void CanvasLineCircle (
    struct Canvas_t * canvas,
    PointF const * origin,
    float radius,
    int32_t steps,
    Color color,
    BlendingEffect effect )
```

Draws circle with the given origin, radius, color and steps using "line" primitive.

7.1.1.115 CanvasLineEllipse()

```
AFTERWARP_API void CanvasLineEllipse (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * radius,
    int32_t steps,
    Color color,
    BlendingEffect effect )
```

Draws ellipse with the given origin, radiuses, color and steps using "line" primitive.

7.1.1.116 CanvasLineHexagon()

```
AFTERWARP_API void CanvasLineHexagon (
    struct Canvas_t * canvas,
    Color color1,
    Color color2,
    Color color3,
    Color color4,
    Color color5,
    Color color6,
    BlendingEffect effect )
```

Draws lines between each vertex in a hexagon of unity size centered along axis origin and drawn with gradient of six colors at the corresponding vertices. The bounding width of hexagon is exactly one, which simplifies placement of multiple hexagons. The final size, position and rotation of hexagon can be given using one or a combination of several 3x2 transformation matrices multiplied together. This method uses `line` primitive.

7.1.1.117 CanvasLineHexagonGrad()

```
AFTERWARP_API void CanvasLineHexagonGrad (
    struct Canvas_t * canvas,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws lines between each vertex in a hexagon of unity size centered along axis origin and filled with four-color gradient interpolated to the corresponding vertices. The bounding width of hexagon is exactly one, which simplifies placement of multiple hexagons. The final size, position and rotation of hexagon can be given using one or a combination of several 3x2 transformation matrices multiplied together. This method uses `line` primitive.

7.1.1.118 CanvasLineQuad()

```
AFTERWARP_API void CanvasLineQuad (
    struct Canvas_t * canvas,
    Quad const * vertices,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws a wireframe quadrilateral with the given vertices and colors using "line" primitive.

7.1.1.119 CanvasLines()

```
AFTERWARP_API void CanvasLines (
    struct Canvas_t * canvas,
    PointF const vertices[],
    Color const colors[],
    int32_t const indices[],
    int32_t vertexCount,
    int32_t primitiveCount,
    BlendingEffect effect )
```

Draws lines with the given vertices, colors and indices.

7.1.1.120 CanvasLineTriangle()

```
AFTERWARP_API void CanvasLineTriangle (
    struct Canvas_t * canvas,
    PointF const * vertex1,
    PointF const * vertex2,
    PointF const * vertex3,
    Color color1,
    Color color2,
    Color color3,
    BlendingEffect effect )
```

Draws lines between specified triangle vertices and colors.

7.1.1.121 CanvasPixel()

```
AFTERWARP_API void CanvasPixel (
    struct Canvas_t * canvas,
    PointF const * position,
    Color color,
    BlendingEffect effect )
```

Draws a pixel with the given position and color.

7.1.1.122 CanvasPixels()

```
AFTERWARP_API void CanvasPixels (
    struct Canvas_t * canvas,
    PointF const positions[],
    Color const colors[],
    int32_t elementCount,
    BlendingEffect effect )
```

Draws pixels with the given positions and colors.

7.1.1.123 CanvasQuad()

```
AFTERWARP_API void CanvasQuad (
    struct Canvas_t * canvas,
    Quad const * vertices,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws a filled quadrilateral with the given vertices and colors.

7.1.1.124 CanvasQuadImage()

```
AFTERWARP_API void CanvasQuadImage (
    struct Canvas_t * canvas,
    struct ImageAtlas_t * imageAtlas,
    Quad const * vertices,
    int32_t regionIndex,
    ColorRect const * colors,
    RectF const * sourceRect,
    uint32_t modifiers,
    BlendingEffect effect )
```

Draws a textured quadrilateral from source portion of image atlas, multiplied by a color gradient.

7.1.1.125 CanvasRectWithHole()

```
AFTERWARP_API void CanvasRectWithHole (
    struct Canvas_t * canvas,
    RectF const * rect,
    PointF const * holeOrigin,
    PointF const * holeRadius,
    ColorPair const * colors,
    int32_t steps,
    BlendingEffect effect )
```

Draws a filled rectangle with a hole inside of it, filled with the given colors.

7.1.1.126 CanvasReset()

```
AFTERWARP_API void CanvasReset (
    struct Canvas_t * canvas )
```

Clears canvas internal buffers and cached resources.

7.1.1.127 CanvasResetSamplerState()

```
AFTERWARP_API void CanvasResetSamplerState (
    struct Canvas_t * canvas )
```

Resets any canvas sampler state that is currently set.

7.1.1.128 CanvasRibbon()

```
AFTERWARP_API void CanvasRibbon (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * insideRadius,
    PointF const * outsideRadius,
    int32_t steps,
    Color color,
    BlendingEffect effect )
```

Draws a filled ribbon between inner and outer radiuses filled with a single color.

7.1.1.129 CanvasRibbonGrad()

```
AFTERWARP_API void CanvasRibbonGrad (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * insideRadius,
    PointF const * outsideRadius,
    int32_t steps,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws a filled ribbon between inner and outer radiuses filled with four-color gradient.

7.1.1.130 CanvasRibbonTri()

```
AFTERWARP_API void CanvasRibbonTri (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * insideRadius,
    PointF const * outsideRadius,
    int32_t steps,
    ColorPair const * color1,
    ColorPair const * color2,
    ColorPair const * color3,
    BlendingEffect effect )
```

Draws a filled ribbon between inner and outer radiuses filled with continuous three-pair gradient.

7.1.1.131 CanvasSetAttributes()

```
AFTERWARP_API void CanvasSetAttributes (
    struct Canvas_t * canvas,
    uint32_t attributes )
```

Sets new canvas rendering attributes.

7.1.1.132 CanvasSetClipRect()

```
AFTERWARP_API LibraryBool CanvasSetClipRect (
    struct Canvas_t * canvas,
    Rect const * rect )
```

Sets new clipping rectangle.

7.1.1.133 CanvasSetContextState()

```
AFTERWARP_API LibraryBool CanvasSetContextState (
    struct Canvas_t * canvas,
    CanvasContextState contextState )
```

Configures device rendering state to one of pre-defined canvas states.

7.1.1.134 CanvasSetSamplerState()

```
AFTERWARP_API LibraryBool CanvasSetSamplerState (
    struct Canvas_t * canvas,
    CanvasSamplerState const * samplerState )
```

Sets new canvas sampler state.

7.1.1.135 CanvasSetSignedDistanceField()

```
AFTERWARP_API void CanvasSetSignedDistanceField (
    struct Canvas_t * canvas,
    SignedDistanceField const * signedDistanceField )
```

Sets new Signed Distance Field (SDF) parameters. The actual SDF rendering needs to be enabled by specifying one of the appropriate canvas attributes.

7.1.1.136 CanvasSetTransform()

```
AFTERWARP_API void CanvasSetTransform (
    struct Canvas_t * canvas,
    Matrix3x2 const * transform )
```

Changes 2D transformation matrix.

7.1.1.137 CanvasSetViewProjection()

```
AFTERWARP_API void CanvasSetViewProjection (
    struct Canvas_t * canvas,
    Matrix const * viewProjection )
```

Sets new 3D view/projection transformation matrix.

7.1.1.138 CanvasSetWorld()

```
AFTERWARP_API void CanvasSetWorld (
    struct Canvas_t * canvas,
    Matrix const * world )
```

Sets new 3D world transformation matrix.

7.1.1.139 CanvasTape()

```
AFTERWARP_API void CanvasTape (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * insideRadius,
    PointF const * outsideRadius,
    float initAngle,
    float endAngle,
    int32_t steps,
    Color color,
    BlendingEffect effect )
```

Draws a filled tape between the given radiuses and angles, filled with a single color.

7.1.1.140 CanvasTapeGrad()

```
AFTERWARP_API void CanvasTapeGrad (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * insideRadius,
    PointF const * outsideRadius,
    float initAngle,
    float endAngle,
    int32_t steps,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws a filled tape between the given radiuses and angles, filled with four-color gradient.

7.1.1.141 CanvasTapeTri()

```
AFTERWARP_API void CanvasTapeTri (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * insideRadius,
    PointF const * outsideRadius,
    float initAngle,
    float endAngle,
    int32_t steps,
    ColorPair const * color1,
    ColorPair const * color2,
    ColorPair const * color3,
    BlendingEffect effect )
```

Draws a filled tape between the given radiuses and angles, filled with continuous three-pair gradient.

7.1.1.142 CanvasTexturedQuad()

```
AFTERWARP_API void CanvasTexturedQuad (
    struct Canvas_t * canvas,
    struct Texture_t * texture,
    Quad const * vertices,
    Quad const * texCoords,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws a textured quadrilateral multiplied with a color gradient specified by the given vertices, colors and texture coordinates (in range from 0 to 1).

7.1.1.143 CanvasTexturedQuadRegion()

```
AFTERWARP_API void CanvasTexturedQuadRegion (
    struct Canvas_t * canvas,
    struct Texture_t * texture,
    Quad const * vertices,
    Quad const * texCoords,
    ColorRect const * colors,
    BlendingEffect effect )
```

Draws a textured quadrilateral multiplied with a color gradient specified by the given vertices, colors and texture coordinates (in pixels).

7.1.1.144 CanvasTexturedRoundRect()

```
AFTERWARP_API void CanvasTexturedRoundRect (
    struct Canvas_t * canvas,
    struct Texture_t * texture,
    RectF const * rect,
    Quad const * texCoords,
    ColorRect const * colors,
    float radius,
    int32_t steps,
    BlendingEffect effect )
```

Draws a textured rectangle with round corners and the given gradient and texture coordinates.

7.1.1.145 CanvasTexturedRoundRectRegion()

```
AFTERWARP_API void CanvasTexturedRoundRectRegion (
    struct Canvas_t * canvas,
    struct Texture_t * texture,
    RectF const * rect,
    Quad const * texCoords,
    ColorRect const * colors,
    float radius,
    int32_t steps,
    BlendingEffect effect )
```

Draws a textured rectangle with round corners and the given gradient and texture coordinates (in pixels).

7.1.1.146 CanvasTexturedTriangle()

```
AFTERWARP_API void CanvasTexturedTriangle (
    struct Canvas_t * canvas,
    struct Texture_t * texture,
    PointF const * vertex1,
    PointF const * vertex2,
    PointF const * vertex3,
    PointF const * texCoord1,
    PointF const * texCoord2,
    PointF const * texCoord3,
    Color color1,
    Color color2,
    Color color3,
    BlendingEffect effect )
```

Draws a textured triangle multiplied with a color gradient specified by given positions, colors and texture coordinates (in range from 0 to 1).

7.1.1.147 CanvasTexturedTriangleRegion()

```
AFTERWARP_API void CanvasTexturedTriangleRegion (
    struct Canvas_t * canvas,
    struct Texture_t * texture,
    PointF const * vertex1,
    PointF const * vertex2,
    PointF const * vertex3,
    PointF const * texCoord1,
    PointF const * texCoord2,
    PointF const * texCoord3,
    Color color1,
    Color color2,
    Color color3,
    BlendingEffect effect )
```

Draws a textured triangle multiplied with a color gradient specified by given positions, colors and texture coordinates (in pixels).

7.1.1.148 CanvasTexturedTriangles()

```
AFTERWARP_API void CanvasTexturedTriangles (
    struct Canvas_t * canvas,
    struct Texture_t * texture,
    PointF const vertices[],
    PointF const texCoords[],
    Color const colors[],
    int32_t const indices[],
    int32_t vertexCount,
    int32_t primitiveCount,
    BlendingEffect effect )
```

Draws triangles with the given vertices, texture coordinates, colors and indices.

7.1.1.149 CanvasThickLine()

```
AFTERWARP_API void CanvasThickLine (
    struct Canvas_t * canvas,
    PointF const * srcPos,
    PointF const * destPos,
    ColorPair const * colors,
    float thickness,
    BlendingEffect effect )
```

Draws a line of varying thickness with the given positions and colors.

7.1.1.150 CanvasThickLineCircle()

```
AFTERWARP_API void CanvasThickLineCircle (
    struct Canvas_t * canvas,
    PointF const * origin,
    float radius,
    int32_t steps,
    Color color,
    float thickness,
    BlendingEffect effect )
```

Draws circle of varying thickness with the given origin, radius, color and steps.

7.1.1.151 CanvasThickLineEllipse()

```
AFTERWARP_API void CanvasThickLineEllipse (
    struct Canvas_t * canvas,
    PointF const * origin,
    PointF const * radius,
    int32_t steps,
    Color color,
    float thickness,
    BlendingEffect effect )
```

Draws ellipse of varying thickness with the given origin, radiuses, color and steps.

7.1.1.152 CanvasThickLineHexagon()

```
AFTERWARP_API void CanvasThickLineHexagon (
    struct Canvas_t * canvas,
    Color color1,
    Color color2,
    Color color3,
    Color color4,
    Color color5,
    Color color6,
    float thickness,
    BlendingEffect effect )
```

Draws lines of varying thickness between each vertex in a hexagon of unity size centered along axis origin and drawn with gradient of six colors at the corresponding corners. The final size, position and rotation of hexagon can be given using one or a combination of several 3x2 transformation matrices multiplied together.

7.1.1.153 CanvasThickLineHexagonGrad()

```
AFTERWARP_API void CanvasThickLineHexagonGrad (
    struct Canvas_t * canvas,
    ColorRect const * colors,
    float thickness,
    BlendingEffect effect )
```

Draws lines of varying thickness between each vertex in a hexagon of unity size centered along axis origin and filled with four-color gradient interpolated to the corresponding vertices. The final size, position and rotation of hexagon can be given using one or a combination of several 3x2 transformation matrices multiplied together.

7.1.1.154 CanvasThickLineQuad()

```
AFTERWARP_API void CanvasThickLineQuad (
    struct Canvas_t * canvas,
    Quad const * vertices,
    ColorRect const * colors,
    float thickness,
    BlendingEffect effect )
```

Draws a wireframe quadrilateral of varying thickness with the given vertices and colors.

7.1.1.155 CanvasThickLineTriangle()

```
AFTERWARP_API void CanvasThickLineTriangle (
    struct Canvas_t * canvas,
    PointF const * vertex1,
    PointF const * vertex2,
    PointF const * vertex3,
    Color color1,
    Color color2,
    Color color3,
    float thickness,
    BlendingEffect effect )
```

Draws line path of varying thickness between specified triangle vertices and colors.

7.1.1.156 CanvasTriangle()

```
AFTERWARP_API void CanvasTriangle (
    struct Canvas_t * canvas,
    PointF const * vertex1,
    PointF const * vertex2,
    PointF const * vertex3,
    Color color1,
    Color color2,
    Color color3,
    BlendingEffect effect )
```

Draws triangle filled with color gradient specified by given positions and colors.

7.1.1.157 CanvasTriangles()

```
AFTERWARP_API void CanvasTriangles (
    struct Canvas_t * canvas,
    PointF const vertices[],
    Color const colors[],
    int32_t const indices[],
    int32_t vertexCount,
    int32_t primitiveCount,
    BlendingEffect effect )
```

Draws triangles with the given vertices, colors and indices.

7.1.1.158 CatmullRom()

```
AFTERWARP_API float CatmullRom (
    float predValue1,
    float value1,
    float value2,
    float succValue2,
    float theta )
```

Interpolates between four values using Catmull-Rom spline, where theta parameter is specified in [0, 1] range.

7.1.1.159 ColorDitheringCreate()

```
AFTERWARP_API struct ColorDithering_t * ColorDitheringCreate (
    struct Device_t * device )
```

Creates a new instance of color dithering module.

7.1.1.160 ColorDitheringDestroy()

```
AFTERWARP_API void ColorDitheringDestroy (
    struct ColorDithering_t * colorDithering )
```

Releases given instance of color dithering module.

7.1.1.161 ColorDitheringExecute()

```
AFTERWARP_API LibraryBool ColorDitheringExecute (
    struct ColorDithering_t * colorDithering,
    struct Texture_t * source )
```

Performs color dithering of the source texture rendering a full-screen quad.

7.1.1.162 ColorDitheringExecuteTo()

```
AFTERWARP_API LibraryBool ColorDitheringExecuteTo (
    struct ColorDithering_t * colorDithering,
    struct Texture_t * destination,
    struct Texture_t * source )
```

Performs color dithering of the source texture rendering to destination texture.

7.1.1.163 ColorDitheringGetDevice()

```
AFTERWARP_API struct Device_t * ColorDitheringGetDevice (
    struct ColorDithering_t * colorDithering )
```

Returns device associated with the given module.

7.1.1.164 ColorDitheringGetFormat()

```
AFTERWARP_API ColorDitheringFormat ColorDitheringGetFormat (
    struct ColorDithering_t * colorDithering )
```

Returns target color quantization format for dithering.

7.1.1.165 ColorDitheringSetFormat()

```
AFTERWARP_API void ColorDitheringSetFormat (
    struct ColorDithering_t * colorDithering,
    ColorDitheringFormat format )
```

Updates target color quantization format for dithering.

7.1.1.166 ColorToGray()

```
AFTERWARP_API int32_t ColorToGray (
    Color color )
```

Returns grayscale value in range of [0..255] from the given 32-bit RGBA color value. The resulting value can be considered the color's Luma. The alpha-channel is ignored.

7.1.1.167 ColorToGray16()

```
AFTERWARP_API int32_t ColorToGray16 (
    Color color )
```

Returns grayscale value in range of [0..65535] from the given 32-bit RGBA color value. The resulting value can be considered the color's Luma. The alpha-channel is ignored.

7.1.1.168 ColorToGrayF()

```
AFTERWARP_API float ColorToGrayF (
    Color color )
```

Returns grayscale value in range of [0, 1] from the given 32-bit RGBA color value. The resulting value can be considered the color's Luma. The alpha-channel is ignored.

7.1.1.169 ComposeColors()

```
AFTERWARP_API Color ComposeColors (
    Color source,
    Color dest )
```

Composes source color onto destination color using source and destination color alphas. This essentially means alpha-blending source color onto destination using source's alpha, but with appropriate alpha composition.

7.1.1.170 ComputeProgramBegin()

```
AFTERWARP_API LibraryBool ComputeProgramBegin (
    struct ComputeProgram_t * program )
```

Activates the compute shader program.

7.1.1.171 ComputeProgramBindBuffer()

```
AFTERWARP_API LibraryBool ComputeProgramBindBuffer (
    struct ComputeProgram_t * program,
    struct Buffer_t * buffer,
    uint32_t channel,
    uint32_t offset )
```

Assigns a generic buffer (or a portion of starting from the given offset in bytes) to a particular channel. The current bindings can be considered valid until `ComputeProgramEnd()` or `ComputeProgramResetBindings()` is called.

7.1.1.172 ComputeProgramBindTexture()

```
AFTERWARP_API LibraryBool ComputeProgramBindTexture (
    struct ComputeProgram_t * program,
    struct Texture_t * texture,
    ComputeBindTextureFormat const * bindFormat )
```

Assigns texture to a particular channel.

7.1.1.173 ComputeProgramCommit()

```
AFTERWARP_API LibraryBool ComputeProgramCommit (
    struct ComputeProgram_t * program )
```

Applies any binding changes to constant buffers, which usually occur when a dispatch call is issued, to occur immediately.

7.1.1.174 ComputeProgramCreate()

```
AFTERWARP_API struct ComputeProgram_t * ComputeProgramCreate (
    struct Device_t * device,
    _In_opt_ ProgramElement const * programElements[],
    int32_t programElementCount,
    _In_opt_ char const * shader,
    int32_t shaderLength )
```

Creates a new instance of compute shader program.

7.1.1.175 ComputeProgramDestroy()

```
AFTERWARP_API void ComputeProgramDestroy (
    struct ComputeProgram_t * program )
```

Releases given instance of compute shader program.

7.1.1.176 ComputeProgramDispatch()

```
AFTERWARP_API LibraryBool ComputeProgramDispatch (
    struct ComputeProgram_t * program,
    int32_t groupsX,
    int32_t groupsY,
    int32_t groupsZ )
```

Launches one or more compute shader work groups.

7.1.1.177 ComputeProgramEnd()

```
AFTERWARP_API void ComputeProgramEnd (
    struct ComputeProgram_t * program )
```

Deactivates the compute shader program.

7.1.1.178 ComputeProgramGetDevice()

```
AFTERWARP_API struct Device_t * ComputeProgramGetDevice (
    struct ComputeProgram_t * program )
```

Returns device associated with the given compute program.

7.1.1.179 ComputeProgramResetBindings()

```
AFTERWARP_API void ComputeProgramResetBindings (
    struct ComputeProgram_t * program )
```

Resets any bindings that were previously made as if `ComputeProgramUnbindXXX()` would be called for each of them.

7.1.1.180 ComputeProgramUnbindBuffer()

```
AFTERWARP_API void ComputeProgramUnbindBuffer (
    struct ComputeProgram_t * program,
    struct Buffer_t * buffer,
    uint32_t channel )
```

Removes association that was previously made with `ComputeProgramBindBuffer()` between the buffer and the given channel.

7.1.1.181 ComputeProgramUnbindTexture()

```
AFTERWARP_API void ComputeProgramUnbindTexture (
    struct ComputeProgram_t * program,
    struct Texture_t * texture,
    ComputeBindTextureFormat const * bindFormat )
```

Removes existing texture association with the particular channel.

7.1.1.182 Cubic()

```
AFTERWARP_API float Cubic (
    float predValue1,
    float value1,
    float value2,
    float succValue2,
    float theta )
```

Interpolates between four values using cubic spline, where theta parameter is specified in [0, 1] range.

7.1.1.183 DeviceClear()

```
AFTERWARP_API LibraryBool DeviceClear (
    struct Device_t * device,
    uint8_t clearLayers,
    FloatColor const * clearColor,
    float clearDepth,
    uint32_t clearStencil )
```

Clears rendering surface that is currently active.

7.1.1.184 DeviceCreate()

```
AFTERWARP_API struct Device_t * DeviceCreate (
    struct Application_t * application,
    DeviceTechnology technology,
    DeviceAttributes attributes )
```

Creates a new graphics device of the given technology and attributes.

7.1.1.185 DeviceCreateWrapped()

```
AFTERWARP_API struct Device_t * DeviceCreateWrapped (
    struct Application_t * application,
    DeviceTechnology technology,
    DeviceAttributes attributes,
    UntypedHandle platformDevice )
```

Creates a graphics device wrapped around an already initialized graphics context of the given technology and attributes.

7.1.1.186 DeviceDestroy()

```
AFTERWARP_API void DeviceDestroy (
    struct Device_t * device )
```

Releases given instance of device.

7.1.1.187 DeviceGetAttributes()

```
AFTERWARP_API DeviceAttributes DeviceGetAttributes (
    struct Device_t * device )
```

Returns attributes with which the device has been created.

7.1.1.188 DeviceGetBehavior()

```
AFTERWARP_API DeviceBehavior DeviceGetBehavior (
    struct Device_t * device )
```

Returns device behavior attributes.

7.1.1.189 DeviceGetLegacyBits()

```
AFTERWARP_API DeviceLegacyBits DeviceGetLegacyBits (
    struct Device_t * device )
```

Returns legacy feature bits that define older hardware features.

7.1.1.190 DeviceGetPlatform()

```
AFTERWARP_API DevicePlatform DeviceGetPlatform (
    struct Device_t * device )
```

Returns OS platform the device application is currently running on.

7.1.1.191 DeviceGetPlatformDevice()

```
AFTERWARP_API UntypedHandle DeviceGetPlatformDevice (
    struct Device_t * device )
```

Returns platform-specific device context.

7.1.1.192 DeviceGetRenderingState()

```
AFTERWARP_API void DeviceGetRenderingState (
    struct Device_t * device,
    RenderingState * state )
```

Returns current rendering states.

7.1.1.193 DeviceGetScissor()

```
AFTERWARP_API void DeviceGetScissor (
    struct Device_t * device,
    Rect * scissor )
```

Returns current scissor parameters.

7.1.1.194 DeviceGetTechFeatureVersion()

```
AFTERWARP_API uint32_t DeviceGetTechFeatureVersion (
    struct Device_t * device )
```

Returns the feature level version of technology that is currently being used. The difference between this and technology version is that the last one indicates type of technology being used (for example, DirectX 3D), while this one indicates the level of features available (for example, DirectX 9.0c). The values here are specified in hexadecimal format. That is, a value of 0x213 would indicate version 2.1.3.

7.1.1.195 DeviceGetTechnology()

```
AFTERWARP_API DeviceTechnology DeviceGetTechnology (
    struct Device_t * device )
```

Returns graphics technology type that is currently being used.

7.1.1.196 DeviceGetTechVersion()

```
AFTERWARP_API uint32_t DeviceGetTechVersion (
    struct Device_t * device )
```

Returns the version of technology that is currently being used. The values are specified in hexadecimal format. That is, a value of 0x100 indicates version 1.0, while a value of 0x247 would indicate version 2.4.7. The value is used in combination with device technology, so if technology is DirectX 3D, and this function returns 0xA10, it means that DirectX 10.1 is being used.

7.1.1.197 DeviceGetViewport()

```
AFTERWARP_API void DeviceGetViewport (
    struct Device_t * device,
    Rect * viewport )
```

Returns current viewport parameters.

7.1.1.198 DeviceMemoryBarrier()

```
AFTERWARP_API void DeviceMemoryBarrier (
    struct Device_t * device,
    uint32_t barrierBits )
```

Issues a memory barrier according to the specified bit flags.

7.1.1.199 DeviceResetCache()

```
AFTERWARP_API void DeviceResetCache (
    struct Device_t * device )
```

Releases any cached device resources.

7.1.1.200 DeviceSetRenderingState()

```
AFTERWARP_API LibraryBool DeviceSetRenderingState (
    struct Device_t * device,
    RenderingState const * state )
```

Sets new rendering parameters and operation characteristics.

7.1.1.201 DeviceSetScissor()

```
AFTERWARP_API LibraryBool DeviceSetScissor (
    struct Device_t * device,
    Rect const * scissor )
```

Specifies new scissor parameters.

7.1.1.202 DeviceSetViewport()

```
AFTERWARP_API LibraryBool DeviceSetViewport (
    struct Device_t * device,
    Rect const * viewport )
```

Specifies new viewport parameters.

7.1.1.203 DisplaceRB()

```
AFTERWARP_API Color DisplaceRB (
    Color color )
```

Switches red and blue channels in 32-bit RGBA color value.

7.1.1.204 GainTransform()

```
AFTERWARP_API float GainTransform (
    float value,
    float gain )
```

Transforms value in range of [0, 1] with a power function, adding gain at start and end of the curve, similar to [SineTransform\(\)](#). Values of "gain" closer to zero produce stronger distortions, whereas values close to zero produce almost linear curves.

7.1.1.205 GaussianBlurCreate()

```
AFTERWARP_API struct GaussianBlur_t * GaussianBlurCreate (
    struct Device_t * device,
    float sigma,
    int32_t samples,
    LibraryBool hardwareFiltering )
```

Creates a new instance of gaussian blur module.

7.1.1.206 GaussianBlurDestroy()

```
AFTERWARP_API void GaussianBlurDestroy (
    struct GaussianBlur_t * gaussianBlur )
```

Releases given instance of gaussian blur module.

7.1.1.207 GaussianBlurGetChroma()

```
AFTERWARP_API float GaussianBlurGetChroma (
    struct GaussianBlur_t * gaussianBlur )
```

Returns chroma offset for color adjustment.

7.1.1.208 GaussianBlurGetDevice()

```
AFTERWARP_API struct Device_t * GaussianBlurGetDevice (
    struct GaussianBlur_t * gaussianBlur )
```

Returns device associated with the given gaussian blur.

7.1.1.209 GaussianBlurGetFixedSamples()

```
AFTERWARP_API int32_t GaussianBlurGetFixedSamples (
    struct GaussianBlur_t * gaussianBlur )
```

Returns maximum number of samples supported by a fixed-sample blur, which enables higher performance.

7.1.1.210 GaussianBlurGetHardwareFiltering()

```
AFTERWARP_API LibraryBool GaussianBlurGetHardwareFiltering (
    struct GaussianBlur_t * gaussianBlur )
```

Indicates whether hardware filtering is used to accelerate the blur.

7.1.1.211 GaussianBlurGetSamples()

```
AFTERWARP_API int32_t GaussianBlurGetSamples (
    struct GaussianBlur_t * gaussianBlur )
```

Returns current number of samples in the kernel.

7.1.1.212 GaussianBlurGetSigma()

```
AFTERWARP_API float GaussianBlurGetSigma (
    struct GaussianBlur_t * gaussianBlur )
```

Returns current value of sigma.

7.1.1.213 GaussianBlurSetChroma()

```
AFTERWARP_API void GaussianBlurSetChroma (
    struct GaussianBlur_t * gaussianBlur,
    float chroma )
```

Changes chroma offset in range of [0, inf) for color adjustment (1 means no adjustment is needed).

7.1.1.214 GaussianBlurSetHardwareFiltering()

```
AFTERWARP_API void GaussianBlurSetHardwareFiltering (
    struct GaussianBlur_t * gaussianBlur,
    LibraryBool hardwareFiltering )
```

Adjusts whether hardware filtering should be used to accelerate the blur.

7.1.1.215 GaussianBlurSetSamples()

```
AFTERWARP_API void GaussianBlurSetSamples (
    struct GaussianBlur_t * gaussianBlur,
    int32_t samples )
```

Sets new number of samples in the kernel. Fixed-size sampling kernel requires this number to be odd, between 5 and 17 (manual filtering), or 19 (hardware filtering). For variable-size kernel, any number of samples higher than the maximum number of fixed samples would work; higher values would involve additional performance cost.

7.1.1.216 GaussianBlurSetSigma()

```
AFTERWARP_API void GaussianBlurSetSigma (
    struct GaussianBlur_t * gaussianBlur,
    float sigma )
```

Sets new value of sigma. The value must be equal or higher than zero.

7.1.1.217 GaussianBlurUpdate()

```
AFTERWARP_API LibraryBool GaussianBlurUpdate (
    struct GaussianBlur_t * gaussianBlur,
    struct Texture_t * destination,
    struct Texture_t * intermediary,
    struct Texture_t * source )
```

Applies blur through use of an intermediate texture. Both destination and intermediary textures must have drawable attribute. If source texture has drawable attribute, then it can be passed as destination texture as well.

7.1.1.218 GaussianBlurUpdateAt()

```
AFTERWARP_API LibraryBool GaussianBlurUpdateAt (
    struct GaussianBlur_t * gaussianBlur,
    struct Texture_t * destination,
    struct Texture_t * intermediary,
    struct Texture_t * source,
    Point const * destPos,
    Rect const * intermRect )
```

Applies a horizontal blur from source to intermediary texture, then a vertical blur from intermediary to destination texture. Both destination and intermediary textures must have drawable attribute. If source texture has drawable attribute, then it can be passed as destination texture as well. If destination position and/or intermediate rectangle are specified, then only a portion of intermediary texture will be copied to destination at the given offset.

7.1.1.219 GetColorAlpha()

```
AFTERWARP_API int32_t GetColorAlpha (
    Color color )
```

Returns alpha-channel in range [0, 255] for the given 32-bit RGBA color.

7.1.1.220 GetColorAlphaF()

```
AFTERWARP_API float GetColorAlphaF (
    Color color )
```

Returns alpha-channel in range [0, 1] for the given 32-bit RGBA color.

7.1.1.221 GetSystemTicks()

```
AFTERWARP_API uint64_t GetSystemTicks (
    void )
```

Retrieves number of microseconds that passed since application startup.

7.1.1.222 GrapherArrow()

```
AFTERWARP_API void GrapherArrow (
    struct Grapher_t * grapher,
    PointF const * targetSize,
    Vector const * origin,
    Vector const * destination,
    Color color,
    float thickness,
    float size,
    LineCaps caps )
```

Draws an arrow from origin to destination and the given parameters. `targetSize` is the rendering surface size (required for proper arrow angle calculation).

7.1.1.223 GrapherBegin()

```
AFTERWARP_API LibraryBool GrapherBegin (
    struct Grapher_t * grapher )
```

Prepares grapher for rendering.

7.1.1.224 GrapherBoundingBox()

```
AFTERWARP_API void GrapherBoundingBox (
    struct Grapher_t * grapher,
    Matrix const * volume,
    Color color,
    float thickness,
    float length,
    LineCaps caps )
```

Draws lines around a bounding box for the given volume matrix.

7.1.1.225 GrapherCreate()

```
AFTERWARP_API struct Grapher_t * GrapherCreate (
    struct Device_t * device )
```

Creates new 3D plotting grapher.

7.1.1.226 GrapherDestroy()

```
AFTERWARP_API void GrapherDestroy (
    struct Grapher_t * grapher )
```

Releases given instance of 3D plotting grapher.

7.1.1.227 GrapherDottedLine()

```
AFTERWARP_API void GrapherDottedLine (
    struct Grapher_t * grapher,
    Vector const * position1,
    Vector const * position2,
    Color color1,
    Color color2,
    float thickness1,
    float thickness2,
    float sparsity,
    PointShape shape,
    float angle )
```

Draws a line of dots with the given parameters.

7.1.1.228 GrapherEnd()

```
AFTERWARP_API void GrapherEnd (
    struct Grapher_t * grapher )
```

Finishes grapher rendering.

7.1.1.229 GrapherFlush()

```
AFTERWARP_API void GrapherFlush (
    struct Grapher_t * grapher )
```

Flushes grapher cache and draws any pending primitives on the rendering surface. This can be useful to make sure that nothing remains in graph cache before changing any device states.

7.1.1.230 GrapherGetBatchCount()

```
AFTERWARP_API int32_t GrapherGetBatchCount (
    struct Grapher_t * grapher )
```

Returns number of batches that were sent to rendering pipeline. Drawing each individual batch involves some overhead, which is implementation-specific. If this parameter happens to be considerably high at some point, the rendering code should be revised for better grouping of plotting primitives.

7.1.1.231 GrapherGetDevice()

```
AFTERWARP_API struct Device_t * GrapherGetDevice (
    struct Grapher_t * grapher )
```

Returns device associated with the given grapher module.

7.1.1.232 GrapherGetTransform()

```
AFTERWARP_API void GrapherGetTransform (
    struct Grapher_t * grapher,
    Matrix * transform )
```

Returns currently set 3D transformation (world/view/projection) matrix.

7.1.1.233 GrapherLine()

```
AFTERWARP_API void GrapherLine (
    struct Grapher_t * grapher,
    Vector const * position1,
    Vector const * position2,
    Color color1,
    Color color2,
    float thickness1,
    float thickness2,
    LineCaps caps )
```

Draws a single line with the given parameters.

7.1.1.234 GrapherLines()

```
AFTERWARP_API void GrapherLines (
    struct Grapher_t * grapher,
    Vector4 const vertices[],
    Color const colors[],
    int32_t const indices[],
    int32_t vertexCount,
    int32_t indexCount,
    LineCaps caps )
```

Draws lines at their given vertices, thicknesses ("w" component of each vertex), colors and indices.

7.1.1.235 GrapherPoint()

```
AFTERWARP_API void GrapherPoint (
    struct Grapher_t * grapher,
    Vector const * position,
    Color color,
    float size,
    PointShape shape,
    float angle )
```

Draws a single point with the given parameters.

7.1.1.236 GrapherPoints()

```
AFTERWARP_API void GrapherPoints (
    struct Grapher_t * grapher,
    Vector4 const vertices[],
    Color const colors[],
    float const angles[],
    int32_t elementCount,
    PointShape shape )
```

Draws points at their given vertices with their respective sizes ("w" component of each vertex), colors, angles and the appropriate shape in 3D space.

7.1.1.237 GrapherReset()

```
AFTERWARP_API void GrapherReset (
    struct Grapher_t * grapher )
```

Resets grapher internal resources and releases any non-critical memory buffers. This can reduce overall memory consumption, when switching from one complex visual scene to another, to allow graph to "adapt" to a new scene from a fresh start.

7.1.1.238 GrapherSetTransform()

```
AFTERWARP_API void GrapherSetTransform (
    struct Grapher_t * grapher,
    Matrix const * transform )
```

Changes 3D transformation (world/view/projection) matrix.

7.1.1.239 Hermite()

```
AFTERWARP_API float Hermite (
    float predValue1,
    float value1,
    float value2,
    float succValue2,
    float theta,
    float tension,
    float bias )
```

Interpolates between four values using Hermite spline, where theta parameter is specified in [0, 1] range, tension in [-1, 1] range (from low to high) and bias in (-inf, +inf) range: negative towards the second element, positive towards the first one.

7.1.1.240 ImageAtlasClearRegions()

```
AFTERWARP_API void ImageAtlasClearRegions (
    struct ImageAtlas_t * atlas )
```

Removes all regions.

7.1.1.241 ImageAtlasClearTextures()

```
AFTERWARP_API void ImageAtlasClearTextures (
    struct ImageAtlas_t * atlas )
```

Removes all existing textures.

7.1.1.242 ImageAtlasCreate()

```
AFTERWARP_API struct ImageAtlas_t * ImageAtlasCreate (
    struct Device_t * device )
```

Creates new instance of image atlas associated with a particular device.

7.1.1.243 ImageAtlasCreateRegion()

```
AFTERWARP_API int32_t ImageAtlasCreateRegion (
    struct ImageAtlas_t * atlas,
    Rect const * rect,
    int32_t textureIndex )
```

Creates and adds a new region to the list.

7.1.1.244 ImageAtlasCreateTexture()

```
AFTERWARP_API int32_t ImageAtlasCreateTexture (
    struct ImageAtlas_t * atlas,
    Point const * size,
    PixelFormat format,
    TextureAttributes attributes )
```

Attempts to create and initialize a texture with the given parameters, which is then added to the list. In case of failure, -1 is returned.

7.1.1.245 ImageAtlasDestroy()

```
AFTERWARP_API void ImageAtlasDestroy (
    struct ImageAtlas_t * atlas )
```

Releases given instance of image atlas.

7.1.1.246 ImageAtlasGetDevice()

```
AFTERWARP_API struct Device_t * ImageAtlasGetDevice (
    struct ImageAtlas_t * atlas )
```

Returns device associated with the given image atlas.

7.1.1.247 ImageAtlasMakeRegions()

```
AFTERWARP_API void ImageAtlasMakeRegions (
    struct ImageAtlas_t * atlas,
    Point const * patternSize,
    Point const * visibleSize,
    int32_t patternCount )
```

Creates list of regions based on repeated rectangular pattern dimensions.

7.1.1.248 ImageAtlasPackRegion()

```
AFTERWARP_API int32_t ImageAtlasPackRegion (
    struct ImageAtlas_t * atlas,
    Point const * size,
    int32_t padding )
```

Finds a suitable location to accommodate the given rectangle in existing available space, adds the appropriate region to the list and returns its index. If no space is available, -1 is returned.

7.1.1.249 ImageAtlasPackSurface()

```
AFTERWARP_API int32_t ImageAtlasPackSurface (
    struct ImageAtlas_t * atlas,
    struct Surface_t * surface,
    Rect const * sourceRect,
    int32_t padding )
```

Finds a suitable location to accomodate the given surface in existing available space, adds the appropriate region to the list and copies the contents of surface to the appropriate texture, returning the final region index. If no space is available or copying fails, -1 is returned.

7.1.1.250 ImageAtlasRegion()

```
AFTERWARP_API LibraryBool ImageAtlasRegion (
    struct ImageAtlas_t * atlas,
    int32_t regionIndex,
    ImageRegion * region )
```

Returns region information for the given index.

7.1.1.251 ImageAtlasRegionCount()

```
AFTERWARP_API int32_t ImageAtlasRegionCount (
    struct ImageAtlas_t * atlas )
```

Returns the number of regions contained within the atlas.

7.1.1.252 ImageAtlasRemoveRegion()

```
AFTERWARP_API void ImageAtlasRemoveRegion (
    struct ImageAtlas_t * atlas,
    int32_t regionIndex )
```

Removes region with given index from the list.

7.1.1.253 ImageAtlasRemoveTexture()

```
AFTERWARP_API void ImageAtlasRemoveTexture (
    struct ImageAtlas_t * atlas,
    int32_t textureIndex )
```

Removes texture with the given index.

7.1.1.254 ImageAtlasTexture()

```
AFTERWARP_API struct Texture_t * ImageAtlasTexture (
    struct ImageAtlas_t * atlas,
    int32_t textureIndex )
```

Returns texture associated with a particular index or NULL if such is not available.

7.1.1.255 ImageAtlasTextureCount()

```
AFTERWARP_API int32_t ImageAtlasTextureCount (
    struct ImageAtlas_t * atlas )
```

Returns the number of textures contained within the atlas.

7.1.1.256 InvertColor()

```
AFTERWARP_API Color InvertColor (
    Color color )
```

Inverts each of the components in the color, including alpha-channel.

7.1.1.257 KawaseBlurCreate()

```
AFTERWARP_API struct KawaseBlur_t * KawaseBlurCreate (
    struct Device_t * device )
```

Creates a new instance of kawase blur module.

7.1.1.258 KawaseBlurDestroy()

```
AFTERWARP_API void KawaseBlurDestroy (
    struct KawaseBlur_t * kawaseBlur )
```

Releases given instance of kawase blur module.

7.1.1.259 KawaseBlurGetDevice()

```
AFTERWARP_API struct Device_t * KawaseBlurGetDevice (
    struct KawaseBlur_t * kawaseBlur )
```

Returns device associated with the given kawase blur.

7.1.1.260 KawaseBlurGetOffset()

```
AFTERWARP_API void KawaseBlurGetOffset (
    struct KawaseBlur_t * kawaseBlur,
    PointF * offset )
```

Returns pixel offset for each individual step.

7.1.1.261 KawaseBlurGetPasses()

```
AFTERWARP_API int32_t KawaseBlurGetPasses (
    struct KawaseBlur_t * kawaseBlur )
```

Returns number of blur passes (each pass includes two processing steps).

7.1.1.262 KawaseBlurSetOffset()

```
AFTERWARP_API void KawaseBlurSetOffset (
    struct KawaseBlur_t * kawaseBlur,
    PointF const * offset )
```

Changes pixel offset for each individual step.

7.1.1.263 KawaseBlurSetPasses()

```
AFTERWARP_API void KawaseBlurSetPasses (
    struct KawaseBlur_t * kawaseBlur,
    int32_t passes )
```

Updates number of blur passes (each pass includes two processing steps).

7.1.1.264 KawaseBlurSinglePass()

```
AFTERWARP_API LibraryBool KawaseBlurSinglePass (
    struct KawaseBlur_t * kawaseBlur,
    struct Texture_t * destination,
    struct Texture_t * source,
    PointF const * offset )
```

Applies blur in a single pass with the given custom offset. This can be used simultaneously when performing downsampling.

7.1.1.265 KawaseBlurUpdate()

```
AFTERWARP_API LibraryBool KawaseBlurUpdate (
    struct KawaseBlur_t * kawaseBlur,
    struct Texture_t * destination,
    struct Texture_t * intermediary,
    struct Texture_t * source )
```

Applies an advancing blur kernel from source to intermediary texture, then from intermediary to destination texture. Both destination and intermediary textures must have drawable attribute. If source texture has drawable attribute, then it can be passed as destination texture as well.

7.1.1.266 Lerp()

```
AFTERWARP_API float Lerp (
    float value1,
    float value2,
    float theta )
```

Interpolates between two values linearly, where theta parameter is specified in [0, 1] range.

7.1.1.267 LibraryGetVersion()

```
AFTERWARP_API LibraryBool LibraryGetVersion (
    int32_t * major,
    int32_t * minor,
    uint32_t * build )
```

Returns current version of the shared library.

7.1.1.268 LibrarySerialCode()

```
AFTERWARP_API void LibrarySerialCode (
    char const * serial,
    int32_t length )
```

Specifies a business serial key for the shared library.

7.1.1.269 MakeColor()

```
AFTERWARP_API Color MakeColor (
    Color color,
    int32_t alpha )
```

Creates 32-bit RGBA color with the specified color value, having its alpha-channel multiplied by the specified coefficient and divided by 255. *alpha* must be within [0, 255] range.

7.1.1.270 MakeColorAlpha()

```
AFTERWARP_API Color MakeColorAlpha (
    int32_t alpha )
```

Creates 32-bit RGBA color with the specified alpha-channel and each of red, green and blue components set to 255. *alpha* must be within [0, 255] range.

7.1.1.271 MakeColorAlphaF()

```
AFTERWARP_API Color MakeColorAlphaF (
    float alpha )
```

Creates 32-bit RGBA color with alpha-channel specified by the given coefficient (multiplied by 255) and the rest of components set to 255. *alpha* must be within [0, 1] range.

7.1.1.272 MakeColorF()

```
AFTERWARP_API Color MakeColorF (
    Color color,
    float alpha )
```

Creates 32-bit RGBA color where the specified color value has its alpha-channel multiplied by the given coefficient. *alpha* must be within [0, 1] range.

7.1.1.273 MakeColorGray()

```
AFTERWARP_API Color MakeColorGray (
    int32_t gray,
    int32_t alpha )
```

Creates 32-bit RGBA color using specified grayscale and alpha values. `gray` and `alpha` must be within [0, 255] range.

7.1.1.274 MakeColorGrayF()

```
AFTERWARP_API Color MakeColorGrayF (
    float gray,
    float alpha )
```

Creates 32-bit RGBA color using specified grayscale and alpha-channel values (both multiplied by 255). `gray` and `alpha` must be within [0, 1] range.

7.1.1.275 MakeColorRGB()

```
AFTERWARP_API Color MakeColorRGB (
    int32_t red,
    int32_t green,
    int32_t blue,
    int32_t alpha )
```

Creates 32-bit RGBA color using specified individual integer components for red, green, blue and alpha-channel. All parameters must be within [0, 255] range.

7.1.1.276 MakeColorRGBF()

```
AFTERWARP_API Color MakeColorRGBF (
    float red,
    float green,
    float blue,
    float alpha )
```

Creates 32-bit RGBA color using specified individual floating-point components for red, green, blue and alpha-channel. All parameters must be within [0, 1] range.

7.1.1.277 MakeColorWithGray()

```
AFTERWARP_API Color MakeColorWithGray (
    Color color,
    int32_t gray,
    int32_t alpha )
```

Creates 32-bit RGBA color where the original color value has its components multiplied by the given grayscale value and alpha-channel multiplied by the specified coefficient, and all components divided by 255. `gray` and `alpha` must be in range of [0, 255].

7.1.1.278 MakeColorWithGrayF()

```
AFTERWARP_API Color MakeColorWithGrayF (
    Color color,
    float gray,
    float alpha )
```

Creates 32-bit RGBA color where the original color value has its components multiplied by the given grayscale value and alpha-channel multiplied by the specified coefficient. `gray` and `alpha` must be within [0, 1] range.

7.1.1.279 MeshBoundsTagOffset()

```
AFTERWARP_API void MeshBoundsTagOffset (
    Vector const * meshMinBounds,
    Vector const * meshMaxBounds,
    Vector const * tagMinBounds,
    Vector const * tagMaxBounds,
    Vector * offset )
```

Calculates an offset to displace from mesh to tag origin.

7.1.1.280 MeshBoundsToMatrixModel()

```
AFTERWARP_API void MeshBoundsToMatrixModel (
    Matrix * model,
    Vector const * minBounds,
    Vector const * maxBounds,
    MeshAligns const * aligns,
    float scale )
```

Creates a model transformation matrix that centers the mesh and places it on top of zero plane.

7.1.1.281 MeshBoundsToMatrixVolume()

```
AFTERWARP_API void MeshBoundsToMatrixVolume (
    Matrix * volume,
    Vector const * minBounds,
    Vector const * maxBounds,
    MeshAligns const * aligns,
    float scale )
```

Creates a volume transformation matrix that centers the mesh and places it on top of zero plane.

7.1.1.282 MeshBoundsToMatrixVolumeTag()

```
AFTERWARP_API void MeshBoundsToMatrixVolumeTag (
    Vector const * meshMinBounds,
    Vector const * meshMaxBounds,
    Vector const * tagMinBounds,
    Vector const * tagMaxBounds,
    Matrix * volume,
    float sizeBias )
```

Creates a volume transformation matrix that centers the mesh tag inside mesh volume.

7.1.1.283 MeshBufferCalculateBounds()

```
AFTERWARP_API void MeshBufferCalculateBounds (
    struct MeshBuffer_t * meshBuffer,
    Vector * vertexMin,
    Vector * vertexMax,
    int32_t firstVertex,
    int32_t vertexCount )
```

Calculates minimum and maximum vertex coordinate boundaries in the given range of mesh.

7.1.1.284 MeshBufferCalculateFlatNormals()

```
AFTERWARP_API LibraryBool MeshBufferCalculateFlatNormals (
    struct MeshBuffer_t * meshBuffer,
    float epsilon )
```

Recalculates mesh normals for the given range of indices by using normals of triangles. This rebuilds vertex and index lists entirely.

7.1.1.285 MeshBufferCalculateNormals()

```
AFTERWARP_API void MeshBufferCalculateNormals (
    struct MeshBuffer_t * meshBuffer,
    int32_t firstVertex,
    int32_t vertexCount,
    int32_t firstIndex,
    int32_t indexCount )
```

Recalculates mesh normals for the given range of indices by calculating face normals first and then averaging the resulting values to face vertices.

7.1.1.286 MeshBufferCalculateNormalsWeld()

```
AFTERWARP_API void MeshBufferCalculateNormalsWeld (
    struct MeshBuffer_t * meshBuffer,
    int32_t firstVertex,
    int32_t vertexCount,
    int32_t firstIndex,
    int32_t indexCount,
    float weldEpsilon )
```

Recalculates mesh normals for the given range of indices. Vertices that have their positions almost the same (depends on "weldEpsilon") will be considered duplicated and thus will receive weighted normals from neighbor faces as if it would be the same shared vertex.

7.1.1.287 MeshBufferCentralize()

```
AFTERWARP_API void MeshBufferCentralize (
    struct MeshBuffer_t * meshBuffer,
    int32_t firstVertex,
    int32_t vertexCount )
```

Calculates coordinate boundaries and centralizes the vertices in the given range of mesh.

7.1.1.288 MeshBufferClear()

```
AFTERWARP_API void MeshBufferClear (
    struct MeshBuffer_t * meshBuffer )
```

Removes all vertices and indices in the mesh buffer.

7.1.1.289 MeshBufferCombine()

```
AFTERWARP_API LibraryBool MeshBufferCombine (
    struct MeshBuffer_t * meshBuffer,
    struct MeshBuffer_t * meshBufferSource,
    int32_t firstVertex,
    int32_t vertexCount,
    int32_t firstIndex,
    int32_t indexCount )
```

Includes vertices and indices from source mesh buffer into the current one.

7.1.1.290 MeshBufferConeSimple()

```
AFTERWARP_API LibraryBool MeshBufferConeSimple (
    struct MeshBuffer_t * meshBuffer,
    int32_t sections,
    Vector const * origin,
    float radius,
    float height,
    FloatColor const * color,
    LibraryBool indicesClockwise )
```

Creates a simplistic cone with the given parameters in XZ plane with its base at origin.

7.1.1.291 MeshBufferCreate()

```
AFTERWARP_API struct MeshBuffer_t * MeshBufferCreate (
    void )
```

Creates a new instance of 3D mesh buffer.

7.1.1.292 MeshBufferCreateModel()

```
AFTERWARP_API struct MeshModel_t * MeshBufferCreateModel (
    struct MeshBuffer_t * meshBuffer,
    struct Device_t * device,
    VertexElement const vertexElements[],
    int32_t vertexElementCount,
    int32_t firstVertex,
    int32_t vertexCount,
    int32_t firstIndex,
    int32_t indexCount,
    uint32_t channel )
```

Creates new instance of 3D mesh model with the given vertex elements and actual data from this buffer.

7.1.1.293 MeshBufferCube()

```
AFTERWARP_API LibraryBool MeshBufferCube (
    struct MeshBuffer_t * meshBuffer,
    int32_t horizSections,
    int32_t vertSections,
    int32_t depthSections,
    Vector const * origin,
    Vector const * horizAxis,
    Vector const * vertAxis,
    Vector const * depthAxis,
    PointF const * initTexCoord,
    PointF const * endTexCoord,
    FloatColor const * color,
    LibraryBool indicesClockwise )
```

Creates a 3D cube with the given parameters. Axis vectors define both orientation and size.

7.1.1.294 MeshBufferCubeMinimal()

```
AFTERWARP_API LibraryBool MeshBufferCubeMinimal (
    struct MeshBuffer_t * meshBuffer,
    Vector const * origin,
    Vector const * size,
    FloatColor const * color,
    LibraryBool indicesClockwise )
```

Creates a 3D cube with minimal number of vertices and indices. Texture coordinates, normals and tangents are not generated.

7.1.1.295 MeshBufferCubeRound()

```
AFTERWARP_API LibraryBool MeshBufferCubeRound (
    struct MeshBuffer_t * meshBuffer,
    Vector const * origin,
    Vector const * size,
    float roundness,
    FloatColor const * color,
    LibraryBool indicesClockwise )
```

Creates a 3D cube with round corners.

7.1.1.296 MeshBufferCylinder()

```
AFTERWARP_API LibraryBool MeshBufferCylinder (
    struct MeshBuffer_t * meshBuffer,
    int32_t radialSections,
    int32_t heightSections,
    Vector const * origin,
    Vector const * horizAxis,
    Vector const * vertAxis,
    Vector const * heightAxis,
    float initAngle,
```

```

float endAngle,
float shape,
PointF const * initTexCoord,
PointF const * endTexCoord,
float bendAngle,
float lateralAngle,
FloatColor const * color,
LibraryBool indicesClockwise )

```

Creates a cylinder with the given parameters. Axis vectors define both orientation and size. Initial and ending angles are specified in $[0, 2 * \text{PI}]$ range.

7.1.1.297 MeshBufferDestroy()

```

AFTERWARP_API void MeshBufferDestroy (
    struct MeshBuffer_t * meshBuffer )

```

Releases given instance of 3D mesh buffer.

7.1.1.298 MeshBufferDisc()

```

AFTERWARP_API LibraryBool MeshBufferDisc (
    struct MeshBuffer_t * meshBuffer,
    int32_t radialSections,
    int32_t innerSections,
    Vector const * origin,
    Vector const * horizAxis,
    Vector const * vertAxis,
    Vector const * normal,
    float initAngle,
    float endAngle,
    float shape,
    float initRadius,
    PointF const * initTexCoord,
    PointF const * endTexCoord,
    float lateralAngle,
    FloatColor const * color,
    LibraryBool indicesClockwise )

```

Creates a disc with the given parameters. Axis vectors define both orientation and size. Initial and ending angles are specified in $[0, 2 * \text{PI}]$ range. `initRadius` defines initial radius in proportion to overall disc size.

7.1.1.299 MeshBufferEliminateUnusedVertices()

```

AFTERWARP_API void MeshBufferEliminateUnusedVertices (
    struct MeshBuffer_t * meshBuffer,
    int32_t firstVertex,
    int32_t vertexCount )

```

Processes given range of indices, marking vertices that are used and then eliminating vertices that were left unused, updating the entire array of indices.

7.1.1.300 MeshBufferExtrusion()

```
AFTERWARP_API LibraryBool MeshBufferExtrusion (
    struct MeshBuffer_t * meshBuffer,
    PathElement const path[],
    int32_t pathLength,
    float depth,
    PointF const * texCoords,
    FloatColor const colors[],
    LibraryBool quality )
```

Creates an extruded mesh around the given path. The mesh is colored with a gradient of four colors. That is, "colors" parameter is an array of four colors each corresponding to top/left, top/right, bottom/right and bottom/left corners.

7.1.1.301 MeshBufferFrustumVolume()

```
AFTERWARP_API LibraryBool MeshBufferFrustumVolume (
    struct MeshBuffer_t * meshBuffer,
    Matrix const * viewProjectionInverse,
    LibraryBool depthClipNegative )
```

Creates a 3D volume of a view frustum from the given view/projection matrix.

7.1.1.302 MeshBufferGeosphere()

```
AFTERWARP_API LibraryBool MeshBufferGeosphere (
    struct MeshBuffer_t * meshBuffer,
    int32_t subdivisions,
    FloatColor const * color,
    LibraryBool flatNormals,
    LibraryBool indicesClockwise )
```

Creates a 3D geosphere with the given parameters.

7.1.1.303 MeshBufferGetIndexCount()

```
AFTERWARP_API int32_t MeshBufferGetIndexCount (
    struct MeshBuffer_t * meshBuffer )
```

Returns total number of indices in 3D mesh buffer.

7.1.1.304 MeshBufferGetIndices()

```
AFTERWARP_API int32_t * MeshBufferGetIndices (
    struct MeshBuffer_t * meshBuffer )
```

Returns pointer to indices stored in the 3D mesh buffer.

7.1.1.305 MeshBufferGetTransform()

```
AFTERWARP_API void MeshBufferGetTransform (
    struct MeshBuffer_t * meshBuffer,
    Matrix * transform )
```

Returns currently set transformation matrix.

7.1.1.306 MeshBufferGetVertexCount()

```
AFTERWARP_API int32_t MeshBufferGetVertexCount (
    struct MeshBuffer_t * meshBuffer )
```

Returns total number of vertices in 3D mesh buffer.

7.1.1.307 MeshBufferGetVertices()

```
AFTERWARP_API MeshBufferEntry * MeshBufferGetVertices (
    struct MeshBuffer_t * meshBuffer )
```

Returns pointer to vertices stored in 3D mesh buffer.

7.1.1.308 MeshBufferInvertIndexOrder()

```
AFTERWARP_API void MeshBufferInvertIndexOrder (
    struct MeshBuffer_t * meshBuffer,
    int32_t firstIndex,
    int32_t indexCount )
```

Inverts order of indices (e.g. from clockwise to counter-clockwise and vice-versa).

7.1.1.309 MeshBufferInvertNormals()

```
AFTERWARP_API void MeshBufferInvertNormals (
    struct MeshBuffer_t * meshBuffer,
    int32_t firstVertex,
    int32_t vertexCount )
```

Invert the normals for vertices in the given range. If vertex count is not specified, then remaining vertices starting at the given vertex and up to the end of the list are processed.

7.1.1.310 MeshBufferJoinDuplicateVertices()

```
AFTERWARP_API LibraryBool MeshBufferJoinDuplicateVertices (
    struct MeshBuffer_t * meshBuffer,
    int32_t firstVertex,
    int32_t vertexCount,
    float threshold )
```

Joins vertices that have duplicate positions, averaging their normals, tangents and colors. This has complexity of $O(n^2)$, where n is number of vertices to process.

7.1.1.311 MeshBufferLoadFromFile()

```
AFTERWARP_API LibraryBool MeshBufferLoadFromFile (
    struct MeshBuffer_t * meshBuffer,
    char const * fileName,
    struct SceneMeshMaterials_t * materials,
    struct MeshMetaTags_t * tags,
    _Inout_ uint32_t * options,
    MeshLoadSaveFeedback feedback,
    void * feedbackUser,
    char * debug,
    _Inout_ int32_t * debugLength )
```

Loads natively Wavefront OBJ mesh file format (or a proprietary mesh binary format) from disk to the mesh buffer. Optionally loads materials included with the mesh file and/or meta tags (important boundaries of sub-meshes). In loading fails, optionally provides debug information.

7.1.1.312 MeshBufferLoadFromFileEx()

```
AFTERWARP_API LibraryBool MeshBufferLoadFromFileEx (
    struct MeshBuffer_t * meshBuffer,
    struct MeshMetaTags_t * tags,
    char const * fileName,
    Vector * minBounds,
    Vector * maxBounds,
    LibraryBool normals,
    char * debug,
    _Inout_ int32_t * debugLength )
```

Attempts to load the mesh file from disk using Assimp DLL to mesh buffer and calculates its boundaries. Optionally loads mesh meta tags (important boundaries of sub-meshes) and/or provides debug information. If `normals` parameter is `true`, then normals will be optionally generated when such are not present.

7.1.1.313 MeshBufferPlane()

```
AFTERWARP_API LibraryBool MeshBufferPlane (
    struct MeshBuffer_t * meshBuffer,
    int32_t horizSections,
    int32_t vertSections,
    Vector const * origin,
    Vector const * horizAxis,
    Vector const * vertAxis,
    Vector const * normal,
    PointF const * initTexCoord,
    PointF const * endTexCoord,
    FloatColor const * color,
    LibraryBool indicesClockwise )
```

Creates a 3D plane with the given parameters. Axis vectors define both orientation and size.

7.1.1.314 MeshBufferSaveToFile()

```
AFTERWARP_API LibraryBool MeshBufferSaveToFile (
    struct MeshBuffer_t * meshBuffer,
    char const * fileName,
    struct SceneMeshMaterials_t * materials,
    struct MeshMetaTags_t * tags,
    uint32_t options,
    MeshLoadSaveFeedback feedback,
    void * feedbackUser,
    char * debug,
    _Inout_ int32_t * debugLength )
```

Saves mesh buffer contents to a Wavefront OBJ mesh format natively to file on disk. Materials and tags are optional and will also be saved, if provided. If saving fails, optionally provides debug information.

7.1.1.315 MeshBufferSaveToFileEx()

```
AFTERWARP_API LibraryBool MeshBufferSaveToFileEx (
    struct MeshBuffer_t * meshBuffer,
    char const * fileName,
    uint32_t const * options,
    char * debug,
    _Inout_ int32_t * debugLength )
```

Attempts to save the mesh to file on disk using Assimp DLL from an existing mesh buffer. Optionally provides debug information.

7.1.1.316 MeshBufferSetIndexCount()

```
AFTERWARP_API LibraryBool MeshBufferSetIndexCount (
    struct MeshBuffer_t * meshBuffer,
    int32_t indexCount )
```

Resizes 3D mesh buffer to accomodate new number of indices.

7.1.1.317 MeshBufferSetTransform()

```
AFTERWARP_API void MeshBufferSetTransform (
    struct MeshBuffer_t * meshBuffer,
    Matrix const * transform )
```

Sets new transformation matrix that is applied to generated vertices during construction.

7.1.1.318 MeshBufferSetVertexCount()

```
AFTERWARP_API LibraryBool MeshBufferSetVertexCount (
    struct MeshBuffer_t * meshBuffer,
    int32_t vertexCount )
```

Resizes 3D mesh buffer to accomodate new number of vertices.

7.1.1.319 MeshBufferSuperEllipse()

```
AFTERWARP_API LibraryBool MeshBufferSuperEllipse (
    struct MeshBuffer_t * meshBuffer,
    int32_t longitudeSections,
    int32_t latitudeSections,
    Vector const * origin,
    Vector const * radius,
    float initLongitudeAngle,
    float endLongitudeAngle,
    float longitudeShape,
    float initLatitudeAngle,
    float endLatitudeAngle,
    float latitudeShape,
    PointF const * initTexCoord,
    PointF const * endTexCoord,
    FloatColor const * color,
    LibraryBool indicesClockwise )
```

Creates a 3D superellipse with the given parameters. Longitude's initial and ending angles are specified in $[0, 2 * \text{PI}]$ ranges, whereas latitude's initial and ending angles are specified in a different range of $[-\text{PI} / 2, \text{PI} / 2]$.

7.1.1.320 MeshBufferSupertoroid()

```
AFTERWARP_API LibraryBool MeshBufferSupertoroid (
    struct MeshBuffer_t * meshBuffer,
    int32_t outerSections,
    int32_t innerSections,
    Vector const * origin,
    float initOuterAngle,
    float endOuterAngle,
    float outerRadius,
    float outerShape,
    float initInnerAngle,
    float endInnerAngle,
    float innerRadius,
    float innerShape,
    PointF const * initTexCoord,
    PointF const * endTexCoord,
    FloatColor const * color,
    LibraryBool indicesClockwise )
```

Creates a 3D supertoroid located in XY plane with the given parameters. Initial and ending angles for both inner and outer rings are specified in $[0, 2 * \text{PI}]$ range.

7.1.1.321 MeshBufferTorus()

```
AFTERWARP_API LibraryBool MeshBufferTorus (
    struct MeshBuffer_t * meshBuffer,
    int32_t outerSections,
    int32_t innerSections,
    Vector const * origin,
    Vector const * horizAxis,
    Vector const * vertAxis,
```

```

float initOuterAngle,
float endOuterAngle,
float outerShape,
float initInnerAngle,
float endInnerAngle,
float innerShape,
PointF const * innerRadius,
PointF const * initTexCoord,
PointF const * endTexCoord,
float lateralAngle,
FloatColor const * color,
LibraryBool indicesClockwise )

```

Creates a 3D torus with the given parameters. Axis vectors define both orientation and size. Initial and ending angles for both inner and outer rings are specified in $[0, 2 * \text{PI}]$ range.

7.1.1.322 MeshBufferTorusKnot()

```

AFTERWARP_API LibraryBool MeshBufferTorusKnot (
    struct MeshBuffer_t * meshBuffer,
    int32_t outerSections,
    int32_t innerSections,
    Vector const * origin,
    int32_t p,
    int32_t q,
    float initOuterAngle,
    float endOuterAngle,
    float outerRadius,
    float initInnerAngle,
    float endInnerAngle,
    float innerShape,
    PointF const * innerRadius,
    PointF const * initTexCoord,
    PointF const * endTexCoord,
    float lateralAngle,
    FloatColor const * color,
    LibraryBool indicesClockwise )

```

Creates a 3D p-q torus knot with the given parameters. Initial and ending angles for both inner and outer rings are specified in $[0, 2 * \text{PI}]$ range.

7.1.1.323 MeshBufferTransferIndexElements()

```

AFTERWARP_API uint32_t MeshBufferTransferIndexElements (
    struct MeshBuffer_t * meshBuffer,
    void * buffer,
    uint32_t pitch,
    int32_t firstVertex,
    int32_t firstIndex,
    int32_t indexCount )

```

Transfers mesh indices to the specified index buffer. The buffer data type is determined by its pitch. Returns number of bytes that the resulting data occupies. If `buffer` is NULL, this function only calculates how many bytes the data will occupy.

7.1.1.324 MeshBufferTransferIndices()

```
AFTERWARP_API LibraryBool MeshBufferTransferIndices (
    struct MeshBuffer_t * meshBuffer,
    struct Buffer_t * buffer,
    int32_t firstVertex,
    int32_t firstIndex,
    int32_t indexCount,
    uint32_t offset )
```

Transfers mesh index elements to an index buffer object with the given format at the specified offset.

7.1.1.325 MeshBufferTransferVertexElements()

```
AFTERWARP_API uint32_t MeshBufferTransferVertexElements (
    struct MeshBuffer_t * meshBuffer,
    void * buffer,
    VertexElement const vertexElements[],
    int32_t vertexElementCount,
    int32_t firstVertex,
    int32_t vertexCount,
    uint32_t channel,
    uint32_t semanticIndex )
```

Transfers mesh vertex elements to an interleaved system buffer with the given format at the specified channel. Returns number of bytes that the resulting data occupies. If `buffer` is `NULL`, this function only calculates how many bytes the data will occupy.

7.1.1.326 MeshBufferTransferVertices()

```
AFTERWARP_API LibraryBool MeshBufferTransferVertices (
    struct MeshBuffer_t * meshBuffer,
    struct Buffer_t * buffer,
    VertexElement const vertexElements[],
    int32_t vertexElementCount,
    int32_t firstVertex,
    int32_t vertexCount,
    uint32_t channel,
    uint32_t offset,
    uint32_t semanticIndex )
```

Transfers mesh vertex elements to an interleaved vertex buffer object with the given format at the specified channel and offset.

7.1.1.327 MeshBufferTransformVertices()

```
AFTERWARP_API void MeshBufferTransformVertices (
    struct MeshBuffer_t * meshBuffer,
    int32_t firstVertex,
    int32_t vertexCount )
```

Applies current transformation to the vertices.

7.1.1.328 MeshBufferVoxelize()

```
AFTERWARP_API struct MeshVoxel_t * MeshBufferVoxelize (
    struct MeshBuffer_t * meshBuffer,
    uint8_t levels,
    LibraryBool colors )
```

Creates and builds a 3D voxel representation for geometry in the mesh buffer. A special care must be taken to define levels, as using high values may exhaust available RAM resources; typical values should be between 3 and 4.

7.1.1.329 MeshMetaTagGetBounds()

```
AFTERWARP_API void MeshMetaTagGetBounds (
    struct MeshMetaTag_t * tag,
    Vector * boundsMin,
    Vector * boundsMax )
```

Calculates and returns boundaries for all existing tag's portions, if such exist.

7.1.1.330 MeshMetaTagGetName()

```
AFTERWARP_API void MeshMetaTagGetName (
    struct MeshMetaTag_t * tag,
    char * name,
    _Inout_ int32_t * nameLength )
```

Returns name of the tag. The pointed value of `nameLength` defines the maximum number of characters, including null-terminating character, that can be copied. The actual string length is stored in `nameLength`.

7.1.1.331 MeshMetaTagGetParent()

```
AFTERWARP_API struct MeshMetaTags_t * MeshMetaTagGetParent (
    struct MeshMetaTag_t * tag )
```

Returns mesh meta tag's parent container.

7.1.1.332 MeshMetaTagGetPortionCount()

```
AFTERWARP_API int32_t MeshMetaTagGetPortionCount (
    struct MeshMetaTag_t * tag )
```

Returns number of existing portions associated with the tag.

7.1.1.333 MeshMetaTagGetType()

```
AFTERWARP_API uint8_t MeshMetaTagGetType (
    struct MeshMetaTag_t * tag )
```

Returns type of the tag.

7.1.1.334 MeshMetaTagPortionAdd()

```
AFTERWARP_API LibraryBool MeshMetaTagPortionAdd (
    struct MeshMetaTag_t * tag,
    MeshMetaTagPortion const * portion )
```

Adds a new portion to the tag.

7.1.1.335 MeshMetaTagPortionErase()

```
AFTERWARP_API void MeshMetaTagPortionErase (
    struct MeshMetaTag_t * tag,
    int32_t index )
```

Erases a portion with the given index.

7.1.1.336 MeshMetaTagPortionGet()

```
AFTERWARP_API LibraryBool MeshMetaTagPortionGet (
    struct MeshMetaTag_t * tag,
    MeshMetaTagPortion * portion,
    int32_t index )
```

Returns a portion that corresponds to the tag with a given index.

7.1.1.337 MeshMetaTagPortionsClear()

```
AFTERWARP_API void MeshMetaTagPortionsClear (
    struct MeshMetaTag_t * tag )
```

Removes all existing portions.

7.1.1.338 MeshMetaTagPortionsCopy()

```
AFTERWARP_API LibraryBool MeshMetaTagPortionsCopy (
    struct MeshMetaTag_t * tag,
    struct MeshMetaTag_t * source )
```

Removes any existing portions and then copies them from an existing tag.

7.1.1.339 MeshMetaTagsAdd()

```
AFTERWARP_API struct MeshMetaTag_t * MeshMetaTagsAdd (
    struct MeshMetaTags_t * tags,
    char const * name,
    uint8_t type )
```

Creates a new tag with the given name (case-sensitive) and type.

7.1.1.340 MeshMetaTagsClear()

```
AFTERWARP_API void MeshMetaTagsClear (
    struct MeshMetaTags_t * tags )
```

Removes all existing tags from container list.

7.1.1.341 MeshMetaTagsCopy()

```
AFTERWARP_API LibraryBool MeshMetaTagsCopy (
    struct MeshMetaTags_t * tags,
    struct MeshMetaTags_t * tagsAnother )
```

Copies tags from another container.

7.1.1.342 MeshMetaTagsCount()

```
AFTERWARP_API int32_t MeshMetaTagsCount (
    struct MeshMetaTags_t * tags )
```

Returns number of existing tags.

7.1.1.343 MeshMetaTagsCreate()

```
AFTERWARP_API struct MeshMetaTags_t * MeshMetaTagsCreate (
    void )
```

Creates a new instance of 3D mesh tag container.

7.1.1.344 MeshMetaTagsDestroy()

```
AFTERWARP_API void MeshMetaTagsDestroy (
    struct MeshMetaTags_t * tags )
```

Releases the given 3D mesh tag container instance.

7.1.1.345 MeshMetaTagsErase()

```
AFTERWARP_API void MeshMetaTagsErase (
    struct MeshMetaTags_t * tags,
    int32_t index )
```

Removes an existing tag from container list.

7.1.1.346 MeshMetaTagsGetByIndex()

```
AFTERWARP_API struct MeshMetaTag_t * MeshMetaTagsGetByIndex (
    struct MeshMetaTags_t * tags,
    int32_t index )
```

Returns a particular tag with the given index.

7.1.1.347 MeshMetaTagsGetByName()

```
AFTERWARP_API struct MeshMetaTag_t * MeshMetaTagsGetByName (
    struct MeshMetaTags_t * tags,
    char const * name,
    uint8_t type )
```

Returns a particular tag with the given name (case-sensitive) and type.

7.1.1.348 MeshMetaTagsTakeAway()

```
AFTERWARP_API void MeshMetaTagsTakeAway (
    struct MeshMetaTags_t * tags,
    struct MeshMetaTags_t * tagsAnother )
```

Takes away the internal contents from another tags container without doing any copying.

7.1.1.349 MeshModelCreate()

```
AFTERWARP_API struct MeshModel_t * MeshModelCreate (
    struct Device_t * device,
    int32_t vertexCount,
    uint32_t vertexElementPitch,
    int32_t indexCount,
    uint32_t channel,
    void const * initialVertexData,
    void const * initialIndexData )
```

Creates a new instance of 3D mesh model.

7.1.1.350 MeshModelDestroy()

```
AFTERWARP_API void MeshModelDestroy (
    struct MeshModel_t * meshModel )
```

Releases given instance of 3D mesh model.

7.1.1.351 MeshModelDismantle()

```
AFTERWARP_API void MeshModelDismantle (
    struct MeshModel_t * meshModel )
```

Releases vertex and/or index buffers, setting number of vertices and indices to zero. This basically turns an existing model into a "dummy" or a placeholder.

7.1.1.352 MeshModelDraw()

```
AFTERWARP_API LibraryBool MeshModelDraw (
    struct MeshModel_t * meshModel,
    struct Program_t * program,
    PrimitiveTopology topology,
    int32_t elementCount,
    int32_t firstIndex,
    int32_t baseVertex,
    LibraryBool postUnbind )
```

Renders the mesh model with the given program.

7.1.1.353 MeshModelDrawInstances()

```
AFTERWARP_API LibraryBool MeshModelDrawInstances (
    struct MeshModel_t * meshModel,
    struct Program_t * program,
    int32_t instanceCount,
    PrimitiveTopology topology,
    int32_t elementCount,
    int32_t firstIndex,
    int32_t baseVertex,
    LibraryBool postUnbind )
```

Renders multiple instances of the mesh model with a given program.

7.1.1.354 MeshModelGetChannel()

```
AFTERWARP_API uint32_t MeshModelGetChannel (
    struct MeshModel_t * meshModel )
```

Returns channel associated with vertex buffer format.

7.1.1.355 MeshModelGetIndexBuffer()

```
AFTERWARP_API struct Buffer_t * MeshModelGetIndexBuffer (
    struct MeshModel_t * meshModel )
```

Returns pointer to integrated index buffer.

7.1.1.356 MeshModelGetIndexCount()

```
AFTERWARP_API int32_t MeshModelGetIndexCount (
    struct MeshModel_t * meshModel )
```

Returns total number of indices stored in the model.

7.1.1.357 MeshModelGetVertexBuffer()

```
AFTERWARP_API struct Buffer_t * MeshModelGetVertexBuffer (
    struct MeshModel_t * meshModel )
```

Returns pointer to integrated vertex buffer.

7.1.1.358 MeshModelGetVertexCount()

```
AFTERWARP_API int32_t MeshModelGetVertexCount (
    struct MeshModel_t * meshModel )
```

Returns total number of vertices stored in the model.

7.1.1.359 MeshModelRenderable()

```
AFTERWARP_API LibraryBool MeshModelRenderable (
    struct MeshModel_t * meshModel )
```

Tests whether the model can actually be rendered. For that, it must contain at least a non-empty vertex buffer. Non-renderable mesh is basically a "dummy" that serves as a placeholder.

7.1.1.360 MeshModelTakeAway()

```
AFTERWARP_API void MeshModelTakeAway (
    struct MeshModel_t * meshModel,
    struct MeshModel_t * meshModelAnother )
```

Takes away the internal contents from another 3D mesh model container without doing any copying.

7.1.1.361 MeshVoxelComputeParameters()

```
AFTERWARP_API void MeshVoxelComputeParameters (
    struct MeshVoxel_t * meshVoxel,
    int32_t dimensions[],
    uint8_t * levels,
    LibraryBool * colors )
```

Calculates existing voxel dimensions, levels and whether colors are present.

7.1.1.362 MeshVoxelCreate()

```
AFTERWARP_API struct MeshVoxel_t * MeshVoxelCreate (
    void )
```

Creates an empty 3D mesh voxel representation.

7.1.1.363 MeshVoxelDestroy()

```
AFTERWARP_API void MeshVoxelDestroy (
    struct MeshVoxel_t * meshVoxel )
```

Releases the given 3D mesh voxel representation.

7.1.1.364 MeshVoxelExtents()

```
AFTERWARP_API void MeshVoxelExtents (
    struct MeshVoxel_t * meshVoxel,
    Vector * position,
    Vector * size )
```

Returns position and size of 3D mesh voxel representation.

7.1.1.365 MeshVoxelIntersect()

```
AFTERWARP_API LibraryBool MeshVoxelIntersect (
    struct MeshVoxel_t * meshVoxel,
    Vector const * origin,
    Vector const * direction,
    Matrix const * world,
    Matrix const * view,
    float * distance,
    int32_t * testsCount )
```

Performs intersection between voxel representation and ray.

7.1.1.366 MeshVoxelLoadFromFile()

```
AFTERWARP_API LibraryBool MeshVoxelLoadFromFile (
    struct MeshVoxel_t * meshVoxel,
    char const * fileName,
    int32_t dimensions[],
    uint8_t * levels,
    LibraryBool * colors )
```

Loads an existing 3D mesh voxel representation from a file on disk.

7.1.1.367 MeshVoxelLoadFromFileInMemory()

```
AFTERWARP_API LibraryBool MeshVoxelLoadFromFileInMemory (
    struct MeshVoxel_t * meshVoxel,
    void * buffer,
    uint32_t bufferSize,
    int32_t dimensions[],
    uint8_t * levels,
    LibraryBool * colors )
```

Loads an existing 3D mesh voxel representation from a file in memory.

7.1.1.368 MeshVoxelSaveToFile()

```
AFTERWARP_API LibraryBool MeshVoxelSaveToFile (
    struct MeshVoxel_t * meshVoxel,
    char const * fileName,
    int32_t const dimensions[],
    uint8_t levels,
    LibraryBool colors )
```

Saves 3D mesh voxel representation to a file on disk.

7.1.1.369 MeshVoxelTakeAway()

```
AFTERWARP_API void MeshVoxelTakeAway (
    struct MeshVoxel_t * meshVoxel,
    struct MeshVoxel_t * meshVoxelAnother )
```

Takes away the internal contents from another 3D mesh voxel representation replacing any existing contents, without doing any copying.

7.1.1.370 MeshVoxelVisualize()

```
AFTERWARP_API LibraryBool MeshVoxelVisualize (
    struct MeshVoxel_t * meshVoxel,
    uint8_t levelMax,
    MeshVoxelVisualizeFunc visualizeFunc,
    void * user )
```

Visualizes 3D mesh voxel representation by invoking callback function for each cube.

7.1.1.371 MultiplyColors()

```
AFTERWARP_API Color MultiplyColors (
    Color color1,
    Color color2 )
```

Multiplies two 32-bit RGBA color values together.

7.1.1.372 ObjectMaterialsAdd()

```
AFTERWARP_API ObjectMaterial * ObjectMaterialsAdd (
    struct ObjectMaterials_t * objectMaterials )
```

Adds a new material with default parameters.

7.1.1.373 ObjectMaterialsClear()

```
AFTERWARP_API void ObjectMaterialsClear (
    struct ObjectMaterials_t * objectMaterials )
```

Removes all existing materials.

7.1.1.374 ObjectMaterialsCreate()

```
AFTERWARP_API struct ObjectMaterials_t * ObjectMaterialsCreate (
    void )
```

Creates a new instance of 3D object materials container.

7.1.1.375 ObjectMaterialsDestroy()

```
AFTERWARP_API void ObjectMaterialsDestroy (
    struct ObjectMaterials_t * objectMaterials )
```

Releases given instance of 3D object materials container.

7.1.1.376 ObjectMaterialsErase()

```
AFTERWARP_API LibraryBool ObjectMaterialsErase (
    struct ObjectMaterials_t * objectMaterials,
    int32_t index )
```

Erases material with the given index.

7.1.1.377 ObjectMaterialsGetCount()

```
AFTERWARP_API int32_t ObjectMaterialsGetCount (
    struct ObjectMaterials_t * objectMaterials )
```

Returns total number of materials.

7.1.1.378 ObjectMaterialsGetElement()

```
AFTERWARP_API ObjectMaterial * ObjectMaterialsGetElement (
    struct ObjectMaterials_t * objectMaterials,
    int32_t index )
```

Returns pointer to an existing 3D object material parameters.

7.1.1.379 ObjectModelConnectLatches()

```
AFTERWARP_API void ObjectModelConnectLatches (
    struct ObjectModel_t * objectModel,
    int32_t latchParent,
    int32_t latchLocal )
```

Connects a local bone joint with the given index to a corresponding joint of the parent.

7.1.1.380 ObjectModelConnectLatchesByName()

```
AFTERWARP_API void ObjectModelConnectLatchesByName (
    struct ObjectModel_t * objectModel,
    _In_opt_ char const * latchParentName,
    _In_opt_ char const * latchLocalName )
```

Connects a local bone joint with the given name (case-insensitive) to a corresponding joint of the parent.

7.1.1.381 ObjectModelGetAABB()

```
AFTERWARP_API void ObjectModelGetAABB (
    struct ObjectModel_t * objectModel,
    Vector * boxMin,
    Vector * boxMax )
```

Updates and returns axis-aligned bounding box of the object.

7.1.1.382 ObjectModelGetAlignments()

```
AFTERWARP_API void ObjectModelGetAlignments (
    struct ObjectModel_t * objectModel,
    MeshAligns * aligns )
```

Returns mesh alignments.

7.1.1.383 ObjectModelGetAttributes()

```
AFTERWARP_API uint32_t ObjectModelGetAttributes (
    struct ObjectModel_t * objectModel )
```

Returns current object model's attributes.

7.1.1.384 ObjectModelGetChild()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelGetChild (
    struct ObjectModel_t * objectModel,
    int32_t index )
```

Returns pointer to a particular object model's child.

7.1.1.385 ObjectModelGetChildCount()

```
AFTERWARP_API int32_t ObjectModelGetChildCount (
    struct ObjectModel_t * objectModel )
```

Returns number of object model's children.

7.1.1.386 ObjectModelGetColor()

```
AFTERWARP_API void ObjectModelGetColor (
    struct ObjectModel_t * objectModel,
    FloatColor * color )
```

Returns object's color.

7.1.1.387 ObjectModelGetConnectedLatches()

```
AFTERWARP_API void ObjectModelGetConnectedLatches (
    struct ObjectModel_t * objectModel,
    int32_t * latchParent,
    int32_t * latchLocal )
```

Returns indexes of currently connected bone joints.

7.1.1.388 ObjectModelGetDepthBias()

```
AFTERWARP_API float ObjectModelGetDepthBias (
    struct ObjectModel_t * objectModel )
```

Returns current depth bias, which is used during object selection.

7.1.1.389 ObjectModelGetDescription()

```
AFTERWARP_API void ObjectModelGetDescription (
    struct ObjectModel_t * objectModel,
    char * description,
    _Inout_ int32_t * descriptionLength )
```

Retrieves object's description text.

7.1.1.390 ObjectModelGetHighlight()

```
AFTERWARP_API void ObjectModelGetHighlight (
    struct ObjectModel_t * objectModel,
    FloatColor * highlight )
```

Returns object's highlight color.

7.1.1.391 ObjectModelGetID()

```
AFTERWARP_API ObjectModelID ObjectModelGetID (
    struct ObjectModel_t * objectModel )
```

Returns object's unique identification number.

7.1.1.392 ObjectModelGetLatchTransform()

```
AFTERWARP_API void ObjectModelGetLatchTransform (
    struct ObjectModel_t * objectModel,
    Matrix * transform,
    int32_t latchIndex,
    LibraryBool local )
```

Retrieves either local or global transformation matrix for a given latch index.

7.1.1.393 ObjectModelGetLatchTransformByName()

```
AFTERWARP_API void ObjectModelGetLatchTransformByName (
    struct ObjectModel_t * objectModel,
    Matrix * transform,
    char const * name,
    LibraryBool local )
```

Retrieves either local or global transformation matrix for a latch with the given name.

7.1.1.394 ObjectModelGetLatchWaypointCouple()

```
AFTERWARP_API void ObjectModelGetLatchWaypointCouple (
    struct ObjectModel_t * objectModel,
    int32_t group,
    float distance,
    Vector * position,
    Quaternion * orientation )
```

Calculates interpolated position and orientation based on the given distance for the given latch group.

7.1.1.395 ObjectModelGetLatchWaypointCoupleMatrix()

```
AFTERWARP_API void ObjectModelGetLatchWaypointCoupleMatrix (
    struct ObjectModel_t * objectModel,
    int32_t group,
    float distance,
    Matrix * transform )
```

Calculates transformation matrix with interpolated position and orientation based on the given distance for the given latch group.

7.1.1.396 ObjectModelGetLayers()

```
AFTERWARP_API uint64_t ObjectModelGetLayers (
    struct ObjectModel_t * objectModel )
```

Returns layers on which the object is visible.

7.1.1.397 ObjectModelGetMaterial()

```
AFTERWARP_API int32_t ObjectModelGetMaterial (
    struct ObjectModel_t * objectModel )
```

Returns object's material.

7.1.1.398 ObjectModelGetMesh()

```
AFTERWARP_API struct SceneMesh_t * ObjectModelGetMesh (
    struct ObjectModel_t * objectModel )
```

Returns object's renderable mesh.

7.1.1.399 ObjectModelGetMeshName()

```
AFTERWARP_API void ObjectModelGetMeshName (
    struct ObjectModel_t * objectModel,
    char * meshName,
    _Inout_ int32_t * meshNameLength )
```

Returns name of the currently associated mesh (or NULL, if no mesh is associated).

7.1.1.400 ObjectModelGetName()

```
AFTERWARP_API void ObjectModelGetName (
    struct ObjectModel_t * objectModel,
    char * modelName,
    _Inout_ int32_t * modelNameLength )
```

Retrieves object's name.

7.1.1.401 ObjectModelGetNext()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelGetNext (
    struct ObjectModel_t * objectModel )
```

Retrieves next object from the same collection to which the given object belongs. If the given object is last object in the collection, NULL will be returned.

7.1.1.402 ObjectModelGetOrderIndex()

```
AFTERWARP_API int32_t ObjectModelGetOrderIndex (
    struct ObjectModel_t * objectModel )
```

Returns current priority order index.

7.1.1.403 ObjectModelGetOwner()

```
AFTERWARP_API struct ObjectModels_t * ObjectModelGetOwner (
    struct ObjectModel_t * objectModel )
```

Returns a container that owns the object.

7.1.1.404 ObjectModelGetParent()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelGetParent (
    struct ObjectModel_t * objectModel )
```

Returns object model's parent.

7.1.1.405 ObjectModelGetPayload()

```
AFTERWARP_API ObjectPayload ObjectModelGetPayload (
    struct ObjectModel_t * objectModel )
```

Returns object's payload.

7.1.1.406 ObjectModelGetPosition()

```
AFTERWARP_API void ObjectModelGetPosition (
    struct ObjectModel_t * objectModel,
    Vector * position )
```

Extracts and returns current object position.

7.1.1.407 ObjectModelGetSize()

```
AFTERWARP_API void ObjectModelGetSize (
    struct ObjectModel_t * objectModel,
    Vector * size )
```

Returns the object's size.

7.1.1.408 ObjectModelGetTransform()

```
AFTERWARP_API void ObjectModelGetTransform (
    struct ObjectModel_t * objectModel,
    ModelTransform transform,
    Matrix * matrix )
```

Retrieves one of current object model's transforms.

7.1.1.409 ObjectModelGetVoxel()

```
AFTERWARP_API struct MeshVoxel_t * ObjectModelGetVoxel (
    struct ObjectModel_t * objectModel )
```

Returns object model's voxel representation.

7.1.1.410 ObjectModelGetWaypointDistance()

```
AFTERWARP_API float ObjectModelGetWaypointDistance (
    struct ObjectModel_t * objectModel,
    int32_t group )
```

Returns calculated waypoint traveling distance for the given latch group.

7.1.1.411 ObjectModelInvalidate()

```
AFTERWARP_API void ObjectModelInvalidate (
    struct ObjectModel_t * objectModel )
```

Marks local transform, volume and position as dirty, including children.

7.1.1.412 ObjectModelsAdd()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelsAdd (
    struct ObjectModels_t * objectModels,
    _In_opt_ char const * name,
    ObjectPayload payload,
    uint32_t attributes )
```

Creates a new object with the given name and/or payload association.

7.1.1.413 ObjectModelsAddWithID()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelsAddWithID (
    struct ObjectModels_t * objectModels,
    ObjectModelID id,
    _In_opt_ char const * name,
    ObjectPayload payload,
    uint32_t attributes )
```

Creates a new object with a given id, name and/or payload association. If the given ID already exists, or an "unknown ID" is given, a new unique one will be assigned to the object. Unique IDs are generated incrementally with roll-over when highest possible ID is reached, so eventually lower order IDs might be reused; the function ensures, however, that a given ID is unique in sense that it is not assigned to another other object.

7.1.1.414 ObjectModelsClear()

```
AFTERWARP_API void ObjectModelsClear (
    struct ObjectModels_t * objectModels )
```

Removes all objects.

7.1.1.415 ObjectModelsClearViews()

```
AFTERWARP_API void ObjectModelsClearViews (
    struct ObjectModels_t * objectModels )
```

Removes all existing views from the collection.

7.1.1.416 ObjectModelsCreate()

```
AFTERWARP_API struct ObjectModels_t * ObjectModelsCreate (
    struct SceneMeshes_t * sceneMeshes )
```

Creates a new instance of the container associated with a particular list of meshes.

7.1.1.417 ObjectModelsCreateView()

```
AFTERWARP_API struct ObjectModelView_t * ObjectModelsCreateView (
    struct ObjectModels_t * objectModels )
```

Creates a new view associated with this collection.

7.1.1.418 ObjectModelsDestroy()

```
AFTERWARP_API void ObjectModelsDestroy (
    struct ObjectModels_t * objectModels )
```

Releases current instance of the container.

7.1.1.419 ObjectModelsErase()

```
AFTERWARP_API void ObjectModelsErase (
    struct ObjectModels_t * objectModels,
    struct ObjectModel_t * objectModel )
```

Removes the given object.

7.1.1.420 ObjectModelsEraseView()

```
AFTERWARP_API void ObjectModelsEraseView (
    struct ObjectModels_t * objectModels,
    struct ObjectModelView_t * objectModelView )
```

Erases an existing view from the collection.

7.1.1.421 ObjectModelSetAlignments()

```
AFTERWARP_API void ObjectModelSetAlignments (
    struct ObjectModel_t * objectModel,
    MeshAligns const * aligns )
```

Changes mesh alignments.

7.1.1.422 ObjectModelSetAttributes()

```
AFTERWARP_API void ObjectModelSetAttributes (
    struct ObjectModel_t * objectModel,
    uint32_t attributes )
```

Updates current object's attributes.

7.1.1.423 ObjectModelSetColor()

```
AFTERWARP_API void ObjectModelSetColor (
    struct ObjectModel_t * objectModel,
    FloatColor const * color )
```

Updates object's color.

7.1.1.424 ObjectModelSetDepthBias()

```
AFTERWARP_API void ObjectModelSetDepthBias (
    struct ObjectModel_t * objectModel,
    float depthBias )
```

Updates depth bias, which is used during object selection.

7.1.1.425 ObjectModelSetDescription()

```
AFTERWARP_API LibraryBool ObjectModelSetDescription (
    struct ObjectModel_t * objectModel,
    char const * description )
```

Changes object's description text.

7.1.1.426 ObjectModelSetHighlight()

```
AFTERWARP_API void ObjectModelSetHighlight (
    struct ObjectModel_t * objectModel,
    FloatColor const * highlight )
```

Updates object's highlight color.

7.1.1.427 ObjectModelSetLayers()

```
AFTERWARP_API void ObjectModelSetLayers (
    struct ObjectModel_t * objectModel,
    uint64_t layers )
```

Specifies layers on which the object is visible. Note: If no layer bits are set, then object would appear visible on all layers.

7.1.1.428 ObjectModelSetMaterial()

```
AFTERWARP_API void ObjectModelSetMaterial (
    struct ObjectModel_t * objectModel,
    int32_t material )
```

Updates object's material.

7.1.1.429 ObjectModelSetMesh()

```
AFTERWARP_API LibraryBool ObjectModelSetMesh (
    struct ObjectModel_t * objectModel,
    struct SceneMesh_t * sceneMesh )
```

Changes object's renderable mesh.

7.1.1.430 ObjectModelSetMeshName()

```
AFTERWARP_API LibraryBool ObjectModelSetMeshName (
    struct ObjectModel_t * objectModel,
    char const * meshName )
```

Changes object's renderable mesh to one with the given name.

7.1.1.431 ObjectModelSetName()

```
AFTERWARP_API LibraryBool ObjectModelSetName (
    struct ObjectModel_t * objectModel,
    char const * name )
```

Changes object's name.

7.1.1.432 ObjectModelSetOrderIndex()

```
AFTERWARP_API void ObjectModelSetOrderIndex (
    struct ObjectModel_t * objectModel,
    int32_t orderIndex )
```

Updates object's priority order index.

7.1.1.433 ObjectModelSetParent()

```
AFTERWARP_API LibraryBool ObjectModelSetParent (
    struct ObjectModel_t * objectModel,
    struct ObjectModel_t * parent )
```

Changes object model's parent.

7.1.1.434 ObjectModelSetSize()

```
AFTERWARP_API void ObjectModelSetSize (
    struct ObjectModel_t * objectModel,
    Vector const * size )
```

Changes the object's size.

7.1.1.435 ObjectModelSetTransform()

```
AFTERWARP_API LibraryBool ObjectModelSetTransform (
    struct ObjectModel_t * objectModel,
    ModelTransform transform,
    Matrix const * matrix )
```

Updates one of current object model's transforms.

7.1.1.436 ObjectModelSetVoxel()

```
AFTERWARP_API void ObjectModelSetVoxel (
    struct ObjectModel_t * objectModel,
    struct MeshVoxel_t * meshVoxel )
```

Updates object model's voxel representation.

7.1.1.437 ObjectModelsGetFirst()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelsGetFirst (
    struct ObjectModels_t * objectModels )
```

Returns a first object from the collection of existing objects.

7.1.1.438 ObjectModelsGetMeshes()

```
AFTERWARP_API struct SceneMeshes_t * ObjectModelsGetMeshes (
    struct ObjectModels_t * objectModels )
```

Returns the associated mesh container.

7.1.1.439 ObjectModelsGetObjectByID()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelsGetObjectByID (
    struct ObjectModels_t * objectModels,
    ObjectModelID id )
```

Returns an object with the given ID or NULL if such doesn't exist.

7.1.1.440 ObjectModelsGetObjectByName()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelsGetObjectByName (
    struct ObjectModels_t * objectModels,
    char const * name )
```

Returns an object with the given name (case-insensitive) or NULL if such doesn't exist.

7.1.1.441 ObjectModelsGetObjectCount()

```
AFTERWARP_API int32_t ObjectModelsGetObjectCount (
    struct ObjectModels_t * objectModels )
```

Returns number of existing objects.

7.1.1.442 ObjectModelsPayload()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelsPayload (
    struct ObjectModels_t * objectModels,
    ObjectPayload payload )
```

Returns object that corresponds to the given payload in 3D object model container.

7.1.1.443 ObjectModelViewAutoDraw()

```
AFTERWARP_API LibraryBool ObjectModelViewAutoDraw (
    struct ObjectModelView_t * objectModelView,
    struct Scene_t * scene,
    struct ObjectMaterials_t * materials,
    uint32_t options,
    _Inout_ int32_t * drawCalls )
```

Renders objects that are visible in the view according to the provided options. If *scene* is NULL, this does not perform instancing or issue actual draw calls, instead performing a simple non-instancing run and counting any tentative draw calls that would be issued; this can be used to determine if any objects would be rendered at all or not to decide whether to perform a certain rendering pass.

7.1.1.444 ObjectModelViewGetIntersectedObjects()

```
AFTERWARP_API int32_t ObjectModelViewGetIntersectedObjects (
    struct ObjectModelView_t * objectModelView )
```

Returns number of objects that passed intersection during last selection attempt.

7.1.1.445 ObjectModelViewGetIntersectedRays()

```
AFTERWARP_API int32_t ObjectModelViewGetIntersectedRays (
    struct ObjectModelView_t * objectModelView )
```

Returns number of ray vs AABB intersections performed last time selection was performed.

7.1.1.446 ObjectModelViewGetLayers()

```
AFTERWARP_API uint64_t ObjectModelViewGetLayers (
    struct ObjectModelView_t * objectModelView )
```

Returns layers that determine object visibility in the view.

7.1.1.447 ObjectModelViewGetObject()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelViewGetObject (
    struct ObjectModelView_t * objectModelView,
    int32_t index )
```

Returns an object with the given index from the view container.

7.1.1.448 ObjectModelViewGetObjectCount()

```
AFTERWARP_API int32_t ObjectModelViewGetObjectCount (
    struct ObjectModelView_t * objectModelView )
```

Returns number of existing objects that are visible in the view.

7.1.1.449 ObjectModelViewGetObjectsNotCulled()

```
AFTERWARP_API int32_t ObjectModelViewGetObjectsNotCulled (
    struct ObjectModelView_t * objectModelView )
```

Returns number of objects that were not performed cull test, either because they were determined to be fully visible, or because they were marked as hierarchyless.

7.1.1.450 ObjectModelViewGetOwner()

```
AFTERWARP_API struct ObjectModels_t * ObjectModelViewGetOwner (
    struct ObjectModelView_t * objectModelView )
```

Returns parent of the view container.

7.1.1.451 ObjectModelViewGetProjection()

```
AFTERWARP_API void ObjectModelViewGetProjection (
    struct ObjectModelView_t * objectModelView,
    Matrix * projection,
    LibraryBool * depthClipNegative )
```

Returns "Projection" transformation matrix of the view container.

7.1.1.452 ObjectModelViewGetView()

```
AFTERWARP_API void ObjectModelViewGetView (
    struct ObjectModelView_t * objectModelView,
    Matrix * view )
```

Returns "View" transformation matrix of the container.

7.1.1.453 ObjectModelViewGetViewProjection()

```
AFTERWARP_API void ObjectModelViewGetViewProjection (
    struct ObjectModelView_t * objectModelView,
    Matrix * viewProjection )
```

Returns combined "View/Projection" matrix of the container.

7.1.1.454 ObjectModelViewInvalidate()

```
AFTERWARP_API void ObjectModelViewInvalidate (
    struct ObjectModelView_t * objectModelView )
```

Notifies the container that it needs to be repainted.

7.1.1.455 ObjectModelViewSelect()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelViewSelect (
    struct ObjectModelView_t * objectModelView,
    Vector const * origin,
    Vector const * direction,
    float * distance )
```

Performs object selection either through object's voxel hierarchy if such is present and if not, through ray-volume intersection. All selectable and visible objects are tested and the object with closest intersection is finally chosen.

7.1.1.456 ObjectModelViewSelectAny()

```
AFTERWARP_API struct ObjectModel_t * ObjectModelViewSelectAny (
    struct ObjectModelView_t * objectModelView,
    Vector const * origin,
    Vector const * direction,
    float * distance )
```

Performs object selection either through object's voxel hierarchy if such is present and if not, through ray-volume intersection. All visible objects are tested and the object with closest intersection is finally chosen.

7.1.1.457 ObjectModelViewSetLayers()

```
AFTERWARP_API void ObjectModelViewSetLayers (
    struct ObjectModelView_t * objectModelView,
    uint64_t layers )
```

Changes layers that determine object visibility in the view.

7.1.1.458 ObjectModelViewSetProjection()

```
AFTERWARP_API void ObjectModelViewSetProjection (
    struct ObjectModelView_t * objectModelView,
    Matrix const * projection,
    LibraryBool depthClipNegative )
```

Changes "Projection" transformation matrix of the view container.

7.1.1.459 ObjectModelViewSetView()

```
AFTERWARP_API void ObjectModelViewSetView (
    struct ObjectModelView_t * objectModelView,
    Matrix const * view )
```

Changes "View" transformation matrix of the container.

7.1.1.460 ObjectModelViewSort()

```
AFTERWARP_API void ObjectModelViewSort (
    struct ObjectModelView_t * objectModelView,
    ObjectModelViewCompare compare )
```

Sorts the list of objects using a predefined comparison function.

7.1.1.461 ObjectModelViewSortWith()

```
AFTERWARP_API void ObjectModelViewSortWith (
    struct ObjectModelView_t * objectModelView,
    ObjectModelCompareFunc compareFunc,
    void * user )
```

Sorts the list of objects using a custom comparison function.

7.1.1.462 ObjectModelViewUpdate()

```
AFTERWARP_API LibraryBool ObjectModelViewUpdate (
    struct ObjectModelView_t * objectModelView )
```

Creates a list of objects for selection and rendering from the view container. Returns `true` if there are any objects visible in the view.

7.1.1.463 ObjectModelViewUpdateNeeded()

```
AFTERWARP_API void ObjectModelViewUpdateNeeded (
    struct ObjectModelView\_t * objectModelView )
```

Notifies the container that all visible objects must be recalculated.

7.1.1.464 OceanSimulationCreate()

```
AFTERWARP_API struct OceanSimulation\_t * OceanSimulationCreate (
    struct Device\_t * device )
```

Creates a new instance of module for rendering semi-infinite 3D ocean wave fields.

7.1.1.465 OceanSimulationDestroy()

```
AFTERWARP_API void OceanSimulationDestroy (
    struct OceanSimulation\_t * oceanSimulation )
```

Releases an existing ocean rendering module.

7.1.1.466 OceanSimulationGetAttributes()

```
AFTERWARP_API SceneAttributes OceanSimulationGetAttributes (
    struct OceanSimulation\_t * oceanSimulation )
```

Returns currently set rendering attributes.

7.1.1.467 OceanSimulationGetDevice()

```
AFTERWARP_API struct Device\_t * OceanSimulationGetDevice (
    struct OceanSimulation\_t * oceanSimulation )
```

Returns device associated with the given module.

7.1.1.468 OceanSimulationGetMaterial()

```
AFTERWARP_API void OceanSimulationGetMaterial (
    struct OceanSimulation\_t * oceanSimulation,
    OceanMaterial * material )
```

Returns ocean's material.

7.1.1.469 OceanSimulationGetPlane()

```
AFTERWARP_API void OceanSimulationGetPlane (
    struct OceanSimulation\_t * oceanSimulation,
    Vector4 * plane )
```

Returns grid's plane equation.

7.1.1.470 OceanSimulationGetProjection()

```
AFTERWARP_API void OceanSimulationGetProjection (
    struct OceanSimulation_t * oceanSimulation,
    Matrix * projection )
```

Returns projection transformation matrix.

7.1.1.471 OceanSimulationGetSamplerShadow()

```
AFTERWARP_API struct Sampler_t * OceanSimulationGetSamplerShadow (
    struct OceanSimulation_t * oceanSimulation )
```

Returns sampler for reading from shadow map.

7.1.1.472 OceanSimulationGetScale()

```
AFTERWARP_API void OceanSimulationGetScale (
    struct OceanSimulation_t * oceanSimulation,
    Vector * scale )
```

Returns scale of the displacement map.

7.1.1.473 OceanSimulationGetSections()

```
AFTERWARP_API void OceanSimulationGetSections (
    struct OceanSimulation_t * oceanSimulation,
    Point * sections )
```

Returns number of sub-divisions in screen-space.

7.1.1.474 OceanSimulationGetView()

```
AFTERWARP_API void OceanSimulationGetView (
    struct OceanSimulation_t * oceanSimulation,
    Matrix * view )
```

Returns view transformation matrix.

7.1.1.475 OceanSimulationGetViewDistance()

```
AFTERWARP_API float OceanSimulationGetViewDistance (
    struct OceanSimulation_t * oceanSimulation )
```

Returns maximum height-map viewing distance.

7.1.1.476 OceanSimulationGetWavesParameters()

```
AFTERWARP_API void OceanSimulationGetWavesParameters (
    struct OceanSimulation_t * oceanSimulation,
    OceanWavesParameters * parameters )
```

Returns ocean waves simulation parameters.

7.1.1.477 OceanSimulationGetWavesTexture()

```
AFTERWARP_API struct Texture_t * OceanSimulationGetWavesTexture (
    struct OceanSimulation_t * oceanSimulation,
    int32_t index )
```

Returns an existing ocean waves simulation texture. This would be mostly useful for debug purposes.

7.1.1.478 OceanSimulationRender()

```
AFTERWARP_API LibraryBool OceanSimulationRender (
    struct OceanSimulation_t * oceanSimulation,
    struct Texture_t * linearDepths,
    struct SceneLights_t * sceneLights,
    struct ShadowCastingAtlas_t * atlas )
```

Renders the simulation to an existing active rendering surface.

7.1.1.479 OceanSimulationSetAttributes()

```
AFTERWARP_API void OceanSimulationSetAttributes (
    struct OceanSimulation_t * oceanSimulation,
    SceneAttributes attributes )
```

Updates the rendering attributes. Note that only SCENE_ATTRIBUTE_SHADOWS_CUBIC is currently recognized by the module.

7.1.1.480 OceanSimulationSetMaterial()

```
AFTERWARP_API LibraryBool OceanSimulationSetMaterial (
    struct OceanSimulation_t * oceanSimulation,
    OceanMaterial const * material )
```

Updates ocean's material.

7.1.1.481 OceanSimulationSetPlane()

```
AFTERWARP_API LibraryBool OceanSimulationSetPlane (
    struct OceanSimulation_t * oceanSimulation,
    Vector4 const * plane )
```

Changes grid's plane equation.

7.1.1.482 OceanSimulationSetProjection()

```
AFTERWARP_API LibraryBool OceanSimulationSetProjection (
    struct OceanSimulation_t * oceanSimulation,
    Matrix const * projection )
```

Changes projection transformation matrix.

7.1.1.483 OceanSimulationSetScale()

```
AFTERWARP_API LibraryBool OceanSimulationSetScale (
    struct OceanSimulation_t * oceanSimulation,
    Vector const * scale )
```

Changes scale of the displacement map.

7.1.1.484 OceanSimulationSetSections()

```
AFTERWARP_API LibraryBool OceanSimulationSetSections (
    struct OceanSimulation_t * oceanSimulation,
    Point const * sections )
```

Changes number of sub-divisions in screen-space.

7.1.1.485 OceanSimulationSetView()

```
AFTERWARP_API LibraryBool OceanSimulationSetView (
    struct OceanSimulation_t * oceanSimulation,
    Matrix const * view )
```

Changes view transformation matrix.

7.1.1.486 OceanSimulationSetViewDistance()

```
AFTERWARP_API LibraryBool OceanSimulationSetViewDistance (
    struct OceanSimulation_t * oceanSimulation,
    float viewDistance )
```

Changes maximum height-map viewing distance.

7.1.1.487 OceanSimulationSetWavesParameters()

```
AFTERWARP_API LibraryBool OceanSimulationSetWavesParameters (
    struct OceanSimulation_t * oceanSimulation,
    OceanWavesParameters const * parameters )
```

Updates ocean waves simulation parameters.

7.1.1.488 OceanSimulationUpdate()

```
AFTERWARP_API LibraryBool OceanSimulationUpdate (
    struct OceanSimulation_t * oceanSimulation,
    double latency )
```

Executes the simulation of ocean waves and updates internal textures.

7.1.1.489 PathBrokerCreate()

```
AFTERWARP_API struct PathBroker_t * PathBrokerCreate (
    void )
```

Creates a new module for creating paths.

7.1.1.490 PathBrokerDestroy()

```
AFTERWARP_API void PathBrokerDestroy (
    struct PathBroker_t * broker )
```

Releases an existing instance of path creation module.

7.1.1.491 PathBrokerFill()

```
AFTERWARP_API LibraryBool PathBrokerFill (
    struct PathBroker_t * broker,
    struct CanvasBuffer_t * buffer,
    PathElement const path[],
    int32_t pathLength,
    ColorRect const * colors,
    LibraryBool quality )
```

Pre-renders a fill for the given path. `quality` parameter determines how the resulting contours are triangulated.

7.1.1.492 PathBrokerReset()

```
AFTERWARP_API void PathBrokerReset (
    struct PathBroker_t * broker )
```

Resets internal cache and releases any allocated memory.

7.1.1.493 PathBrokerStroke()

```
AFTERWARP_API LibraryBool PathBrokerStroke (
    struct PathBroker_t * broker,
    struct CanvasBuffer_t * buffer,
    PathElement const path[],
    int32_t pathLength,
    float thickness,
    Color color,
    PathJoint joints,
    LineCaps caps,
    float miterLimit,
    float smoothStep )
```

Pre-renders a stroke for the given path.

7.1.1.494 PixelFormatBits()

```
AFTERWARP_API uint32_t PixelFormatBits (
    PixelFormat format )
```

Returns number of bits that each pixel in current format occupies.

7.1.1.495 PixelFormatConvert()

```
AFTERWARP_API void PixelFormatConvert (
    void * dest,
    void const * source,
    PixelFormat destFormat,
    PixelFormat sourceFormat )
```

Converts a single pixel from one pixel format to another preserving as much information as possible.

7.1.1.496 PixelFormatConvertArray()

```
AFTERWARP_API void PixelFormatConvertArray (
    void * dest,
    void const * source,
    PixelFormat destFormat,
    PixelFormat sourceFormat,
    uint32_t destPitch,
    uint32_t sourcePitch,
    int32_t width,
    int32_t height )
```

Converts an array of pixels from one pixel format to another preserving as much information as possible.

7.1.1.497 PixelFormatValid()

```
AFTERWARP_API LibraryBool PixelFormatValid (
    PixelFormat format )
```

Verifies that given pixel format is well-defined, known and valid.

7.1.1.498 PremultiplyAlpha()

```
AFTERWARP_API Color PremultiplyAlpha (
    Color color )
```

Takes 32-bit RGBA color with unpremultiplied alpha and multiplies each of red, green, and blue components by its alpha-channel, resulting in premultiplied alpha color.

7.1.1.499 ProgramBegin()

```
AFTERWARP_API LibraryBool ProgramBegin (
    struct Program_t * program )
```

Activates the shader program and prepares for rendering.

7.1.1.500 ProgramBind()

```
AFTERWARP_API LibraryBool ProgramBind (
    struct Program_t * program,
    struct Buffer_t * buffer,
    uint32_t channel,
    uint32_t offset )
```

Assigns a generic buffer (or a portion of starting from the given offset in bytes) to a particular channel. Vertex and index buffers are associated with input stream channels, whereas constant buffers are associated with shader program channels - each of these being unrelated and dependent on shader vertex and program element declarations. The current bindings can be considered valid until ProgramEnd, ProgramResetBindings or ProgramResetCache is called.

7.1.1.501 ProgramCommit()

```
AFTERWARP_API LibraryBool ProgramCommit (
    struct Program_t * program )
```

Applies any binding changes to constant and/or vertex buffers, which usually occur when a draw call is issued, to occur immediately.

7.1.1.502 ProgramCreate()

```
AFTERWARP_API struct Program_t * ProgramCreate (
    struct Device_t * device,
    _In_opt_ VertexElement const vertexElements[],
    int32_t vertexElementCount,
    _In_opt_ ProgramElement const programElements[],
    int32_t programElementCount,
    _In_opt_ ProgramVariable const programVariables[],
    int32_t programVariableCount,
    _In_opt_ char const * shaderVertex,
    int32_t shaderVertexLength,
    _In_opt_ char const * shaderHull,
    int32_t shaderHullLength,
    _In_opt_ char const * shaderDomain,
    int32_t shaderDomainLength,
    _In_opt_ char const * shaderGeometry,
    int32_t shaderGeometryLength,
    _In_opt_ char const * shaderPixel,
    int32_t shaderPixelLength )
```

Creates a new instance of shader program.

7.1.1.503 ProgramDestroy()

```
AFTERWARP_API void ProgramDestroy (
    struct Program_t * program )
```

Releases given instance of shader program.

7.1.1.504 ProgramDraw()

```
AFTERWARP_API LibraryBool ProgramDraw (
    struct Program_t * program,
    PrimitiveTopology topology,
    int32_t vertexCount,
    int32_t baseVertex )
```

Renders primitives with the given number of vertices.

7.1.1.505 ProgramDrawIndexed()

```
AFTERWARP_API LibraryBool ProgramDrawIndexed (
    struct Program_t * program,
    PrimitiveTopology topology,
    int32_t indexCount,
    int32_t firstIndex,
    int32_t baseVertex )
```

Renders indexed primitives with the given number of indices.

7.1.1.506 ProgramDrawInstances()

```
AFTERWARP_API LibraryBool ProgramDrawInstances (
    struct Program_t * program,
    PrimitiveTopology topology,
    int32_t vertexCount,
    int32_t instanceCount,
    int32_t baseVertex )
```

Renders multiple number of instances of the given primitives.

7.1.1.507 ProgramDrawInstancesIndexed()

```
AFTERWARP_API LibraryBool ProgramDrawInstancesIndexed (
    struct Program_t * program,
    PrimitiveTopology topology,
    int32_t indexCount,
    int32_t instanceCount,
    int32_t firstIndex,
    int32_t baseVertex )
```

Renders multiple number of instances of the given indexed primitives.

7.1.1.508 ProgramEnd()

```
AFTERWARP_API void ProgramEnd (
    struct Program_t * program )
```

Deactivates the shader program, resets previously active input streams and bindings.

7.1.1.509 ProgramGetDevice()

```
AFTERWARP_API struct Device_t * ProgramGetDevice (
    struct Program_t * program )
```

Returns device associated with the given program.

7.1.1.510 ProgramResetBindings()

```
AFTERWARP_API void ProgramResetBindings (
    struct Program_t * program )
```

Resets any bindings that were previously made as if ProgramUnbind would be called for each of them.

7.1.1.511 ProgramResetCache()

```
AFTERWARP_API void ProgramResetCache (
    struct Program_t * program )
```

Resets any cache associated with vertex buffers. In case of OpenGL, this resets internal Vertex Array Object (VAO) cache.

7.1.1.512 ProgramSetPatchVertices()

```
AFTERWARP_API LibraryBool ProgramSetPatchVertices (
    struct Program_t * program,
    int32_t patchVertices )
```

Specifies number of vertices in each patch. This only applies when rendering patch topology.

7.1.1.513 ProgramUnbind()

```
AFTERWARP_API void ProgramUnbind (
    struct Program_t * program,
    struct Buffer_t * buffer,
    uint32_t channel )
```

Removes association that was previously made with ProgramBind between the buffer and the given channel.

7.1.1.514 ProgramUpdateByIndex()

```
AFTERWARP_API LibraryBool ProgramUpdateByIndex (
    struct Program_t * program,
    int32_t variableIndex,
    void const * variableData,
    uint32_t size )
```

Updates a portion of a specific variable that corresponds to the given index. The provided size indicates how many bytes should be updated. If the provided size is left at zero, the entire contents of given variable will be updated. This function should only be used inside ProgramBegin and ProgramEnd call block.

7.1.1.515 ProgramUpdateByName()

```
AFTERWARP_API LibraryBool ProgramUpdateByName (
    struct Program_t * program,
    char const * variableName,
    void const * variableData,
    uint32_t size )
```

Updates a portion of a specific variable identified by its name (case-sensitive). The provided size indicates how many bytes should be updated. If the provided size is left at zero, the entire contents of given variable will be updated. This function should only be used inside ProgramBegin and ProgramEnd call block.

7.1.1.516 RandomSequenceGenerate()

```
AFTERWARP_API uint32_t RandomSequenceGenerate (
    _Inout_ uint64_t * state )
```

Generates next random number in the sequence from the specified context.

7.1.1.517 RandomSequenceGenerate64()

```
AFTERWARP_API uint64_t RandomSequenceGenerate64 (
    _Inout_ uint64_t * state )
```

Returns 64-bit random number by generating two 32-bit numbers.

7.1.1.518 RandomSequenceGenerateDouble()

```
AFTERWARP_API double RandomSequenceGenerateDouble (
    _Inout_ uint64_t * state )
```

Generates a random value in range [0, 1) with 53 bits of precision, using relatively linear distribution.

7.1.1.519 RandomSequenceGenerateFloat()

```
AFTERWARP_API float RandomSequenceGenerateFloat (
    _Inout_ uint64_t * state )
```

Generates a random value in range [0, 1) with 23 bits of precision, using relatively linear distribution.

7.1.1.520 RandomSequenceGenerateGaussian()

```
AFTERWARP_API float RandomSequenceGenerateGaussian (
    _Inout_ uint64_t * state )
```

Generates a random value using Gaussian distribution with mean 0 and standard deviation of 1.

7.1.1.521 RandomSequenceGenerateRanged()

```
AFTERWARP_API int32_t RandomSequenceGenerateRanged (
    _Inout_ uint64_t * state,
    int32_t range )
```

Generates a random value in range [0, Range) using relatively linear distribution.

7.1.1.522 RandomSequenceInit()

```
AFTERWARP_API void RandomSequenceInit (
    _Out_ uint64_t * state )
```

Initializes random number generator state using local time.

7.1.1.523 RandomSequenceInitBySeed()

```
AFTERWARP_API void RandomSequenceInitBySeed (
    _Out_ uint64_t * state,
    uint64_t seed )
```

Initializes random number generator state with a given seed to generate a reproducible sequence of random numbers.

7.1.1.524 RayCreate()

```
AFTERWARP_API void RayCreate (
    Vector * origin,
    Vector * direction,
    PointF const * position,
    PointF const * surfaceSize,
    Matrix const * viewInverse,
    Matrix const * projection )
```

Constructs a ray's origin and direction from the given 2D surface position, surface size, inverse of the 3D view matrix and the projection matrix.

7.1.1.525 RayIntersectCubeVolume()

```
AFTERWARP_API LibraryBool RayIntersectCubeVolume (
    Vector const * origin,
    Vector const * direction,
    Matrix const * world,
    float * distance )
```

Tests whether the current ray intersects with one of triangles of minimalistic cube volume, whose vertices are first transformed by the given world matrix, returning distance to intersection point.

7.1.1.526 RayIntersectPlane()

```
AFTERWARP_API LibraryBool RayIntersectPlane (
    Vector const * origin,
    Vector const * direction,
    Vector const * planePoint,
    Vector const * planeNormal,
    Vector * intersection,
    float * distance )
```

Tests whether the current ray intersects with a plane specified by a point on the plane and its surface normal, returning the actual point of intersection and distance from ray's origin.

7.1.1.527 RayIntersectTriangle()

```
AFTERWARP_API LibraryBool RayIntersectTriangle (
    Vector const * origin,
    Vector const * direction,
    Vector const * vertex1,
    Vector const * vertex2,
    Vector const * vertex3,
    LibraryBool backFacing,
    PointF * intersection,
    float * distance )
```

Tests whether the current ray intersects with the given front or back facing triangle and if so, calculates the resulting barycentric coordinates and distance from origin to triangle.

7.1.1.528 SamplerBind()

```
AFTERWARP_API LibraryBool SamplerBind (
    struct Sampler_t * sampler,
    uint32_t channel )
```

Binds sampler object to a particular channel.

7.1.1.529 SamplerCreate()

```
AFTERWARP_API struct Sampler_t * SamplerCreate (
    struct Device_t * device,
    SamplerState const * samplerState )
```

Creates a new instance of sampler object.

7.1.1.530 SamplerDestroy()

```
AFTERWARP_API void SamplerDestroy (
    struct Sampler_t * sampler )
```

Releases given instance of sampler object.

7.1.1.531 SamplerGetDevice()

```
AFTERWARP_API struct Device_t * SamplerGetDevice (
    struct Sampler_t * sampler )
```

Returns device associated with the given sampler.

7.1.1.532 SamplerGetState()

```
AFTERWARP_API void SamplerGetState (
    struct Sampler_t * sampler,
    SamplerState * samplerState )
```

Retrieves parameters associated with the sampler object.

7.1.1.533 SamplerSetState()

```
AFTERWARP_API LibraryBool SamplerSetState (
    struct Sampler_t * sampler,
    SamplerState const * samplerState )
```

Updates parameters associated with the sampler object.

7.1.1.534 SamplerUnbind()

```
AFTERWARP_API void SamplerUnbind (
    struct Sampler_t * sampler,
    uint32_t channel )
```

Unbinds sampler object from the particular channel.

7.1.1.535 SceneBegin()

```
AFTERWARP_API LibraryBool SceneBegin (
    struct Scene_t * scene )
```

Activates the appropriate shader program and begins rendering the scene.

7.1.1.536 SceneCreateDepthsNormals()

```
AFTERWARP_API struct Scene_t * SceneCreateDepthsNormals (
    struct Device_t * device )
```

Creates a new instance of depths/normals rendering scene.

7.1.1.537 SceneCreateModeling()

```
AFTERWARP_API struct Scene_t * SceneCreateModeling (
    struct Device_t * device )
```

Creates a new instance of 3D rendering scene.

7.1.1.538 SceneDestroy()

```
AFTERWARP_API void SceneDestroy (
    struct Scene_t * scene )
```

Releases an existing scene rendering instance.

7.1.1.539 SceneEnd()

```
AFTERWARP_API void SceneEnd (
    struct Scene_t * scene )
```

Finishes rendering the scene and deactivates previously activated shader program.

7.1.1.540 SceneGetActiveVertexElements()

```
AFTERWARP_API int32_t SceneGetActiveVertexElements (
    struct Scene_t * scene )
```

Returns currently active index of vertex elements declaration.

7.1.1.541 SceneGetAttributes()

```
AFTERWARP_API SceneAttributes SceneGetAttributes (
    struct Scene_t * scene )
```

Returns attributes that define the rendering behavior of the scene.

7.1.1.542 SceneGetDevice()

```
AFTERWARP_API struct Device_t * SceneGetDevice (
    struct Scene_t * scene )
```

Returns device associated with the given scene.

7.1.1.543 SceneGetInstancesCount()

```
AFTERWARP_API int32_t SceneGetInstancesCount (
    struct Scene_t * scene )
```

Returns maximum number of supported instances.

7.1.1.544 SceneGetLights()

```
AFTERWARP_API struct SceneLights_t * SceneGetLights (
    struct Scene_t * scene )
```

Returns lights container module associated with the scene.

7.1.1.545 SceneGetMaterial()

```
AFTERWARP_API void SceneGetMaterial (
    struct Scene_t * scene,
    ObjectMaterial * material )
```

Returns scene object's material properties.

7.1.1.546 SceneGetParallaxMappingParameters()

```
AFTERWARP_API void SceneGetParallaxMappingParameters (
    struct Scene_t * scene,
    ParallaxMappingParameters * parameters )
```

Returns parameters that define Parallax Mapping technique is performed.

7.1.1.547 SceneGetProgram()

```
AFTERWARP_API struct Program_t * SceneGetProgram (
    struct Scene_t * scene )
```

Returns currently active shader program handle.

7.1.1.548 SceneGetProjection()

```
AFTERWARP_API void SceneGetProjection (
    struct Scene_t * scene,
    Matrix * projection )
```

Returns projection matrix.

7.1.1.549 SceneGetSampler()

```
AFTERWARP_API struct Sampler_t * SceneGetSampler (
    struct Scene_t * scene,
    SceneSamplerType type )
```

Returns sampler of the given type associated with the scene.

7.1.1.550 SceneGetShadowCastingAtlas()

```
AFTERWARP_API struct ShadowCastingAtlas_t * SceneGetShadowCastingAtlas (
    struct Scene_t * scene )
```

Returns shadow casting atlas associated with the scene.

7.1.1.551 SceneGetTexture()

```
AFTERWARP_API struct Texture_t * SceneGetTexture (
    struct Scene_t * scene,
    SceneTextureType type )
```

Returns texture of the given type associated with the scene.

7.1.1.552 SceneGetTextureCabinet()

```
AFTERWARP_API struct TextureCabinet_t * SceneGetTextureCabinet (
    struct Scene_t * scene )
```

Returns an associated texture cabinet.

7.1.1.553 SceneGetToneMappingCoefficients()

```
AFTERWARP_API void SceneGetToneMappingCoefficients (
    struct Scene_t * scene,
    Vector4 * coefficients )
```

Returns tone-mapping RGB luma coefficients and maximum allowed level of white. These parameters are used only when "glassy" (OIT) technique is being performed.

7.1.1.554 SceneGetVertexElementCount()

```
AFTERWARP_API int32_t SceneGetVertexElementCount (
    struct Scene_t * scene,
    int32_t index )
```

Returns number of items in vertex element declaration associated with the given index.

7.1.1.555 SceneGetVertexElements()

```
AFTERWARP_API VertexElement const * SceneGetVertexElements (
    struct Scene_t * scene,
    int32_t index )
```

Returns items in vertex element declaration associated with the given index.

7.1.1.556 SceneGetView()

```
AFTERWARP_API void SceneGetView (
    struct Scene_t * scene,
    Matrix * view )
```

Returns view matrix.

7.1.1.557 SceneGetWorld()

```
AFTERWARP_API void SceneGetWorld (
    struct Scene_t * scene,
    Matrix * world )
```

Returns object world matrix.

7.1.1.558 SceneInstances()

```
AFTERWARP_API LibraryBool SceneInstances (
    struct Scene_t * scene,
    Matrix const transforms[],
    FloatColor const colors[],
    int32_t count )
```

Supplies instance information to the pipeline. The maximum number of elements must not exceed the value returned by `SceneGetInstancesCount()` function.

7.1.1.559 SceneLightsAdd()

```
AFTERWARP_API SceneLight * SceneLightsAdd (
    struct SceneLights_t * sceneLights )
```

Adds a new light with default parameters.

7.1.1.560 SceneLightsClear()

```
AFTERWARP_API void SceneLightsClear (
    struct SceneLights_t * sceneLights )
```

Removes all existing lights.

7.1.1.561 SceneLightsCreate()

```
AFTERWARP_API struct SceneLights_t * SceneLightsCreate (
    struct Device_t * device )
```

Creates a new instance of 3D scene light module.

7.1.1.562 SceneLightsDestroy()

```
AFTERWARP_API void SceneLightsDestroy (
    struct SceneLights_t * sceneLights )
```

Releases given instance of 3D scene light module.

7.1.1.563 SceneLightsErase()

```
AFTERWARP_API LibraryBool SceneLightsErase (
    struct SceneLights_t * sceneLights,
    int32_t index )
```

Erases light with the given index.

7.1.1.564 SceneLightsExecute()

```
AFTERWARP_API LibraryBool SceneLightsExecute (
    struct SceneLights_t * sceneLights,
    Matrix const * view,
    Matrix const * projection )
```

Calculates cluster frustums, then performs light culling for each of the clusters. This populates an internal GPU buffer containing light indices for each of the clusters.

7.1.1.565 SceneLightsGetClusters()

```
AFTERWARP_API void SceneLightsGetClusters (
    struct SceneLights_t * sceneLights,
    Point * clusters )
```

Returns number of clusters per width and height of viewable area.

7.1.1.566 SceneLightsGetClusterSize()

```
AFTERWARP_API int32_t SceneLightsGetClusterSize (
    struct SceneLights_t * sceneLights )
```

Returns size of individual lighting clusters.

7.1.1.567 SceneLightsGetCount()

```
AFTERWARP_API int32_t SceneLightsGetCount (
    struct SceneLights_t * sceneLights )
```

Returns total number of light sources.

7.1.1.568 SceneLightsGetCullingMode()

```
AFTERWARP_API ClustersCullingMode SceneLightsGetCullingMode (
    struct SceneLights_t * sceneLights )
```

Returns mode used for packing culled lights in clusters.

7.1.1.569 SceneLightsGetDepthSlices()

```
AFTERWARP_API int32_t SceneLightsGetDepthSlices (
    struct SceneLights_t * sceneLights )
```

Returns number of cluster slices per viewable depth.

7.1.1.570 SceneLightsGetDevice()

```
AFTERWARP_API struct Device_t * SceneLightsGetDevice (
    struct SceneLights_t * sceneLights )
```

Returns device associated with 3D scene light module.

7.1.1.571 SceneLightsGetElement()

```
AFTERWARP_API SceneLight * SceneLightsGetElement (
    struct SceneLights_t * sceneLights,
    int32_t index )
```

Returns pointer to an existing 3D scene light parameters.

7.1.1.572 SceneLightsGetIndices()

```
AFTERWARP_API struct Buffer_t * SceneLightsGetIndices (
    struct SceneLights_t * sceneLights )
```

Returns light indices that have been calculated for each of the clusters.

7.1.1.573 SceneLightsGetViewSize()

```
AFTERWARP_API void SceneLightsGetViewSize (
    struct SceneLights_t * sceneLights,
    Point * viewSize )
```

Returns size of the viewable area.

7.1.1.574 SceneLightsRenderDebug()

```
AFTERWARP_API LibraryBool SceneLightsRenderDebug (
    struct SceneLights_t * sceneLights,
    LibraryBool intensity )
```

Renders light culling debug information. A call to `SceneLightsExecute()` must be performed before this.

7.1.1.575 SceneLightsSetClusterSize()

```
AFTERWARP_API void SceneLightsSetClusterSize (
    struct SceneLights_t * sceneLights,
    int32_t clusterSize )
```

Changes size of individual lighting clusters.

7.1.1.576 SceneLightsSetCullingMode()

```
AFTERWARP_API void SceneLightsSetCullingMode (
    struct SceneLights_t * sceneLights,
    ClustersCullingMode mode )
```

Changes mode used for packing culled lights in clusters.

7.1.1.577 SceneLightsSetDepthSlices()

```
AFTERWARP_API void SceneLightsSetDepthSlices (
    struct SceneLights_t * sceneLights,
    int32_t depthSlices )
```

Updates number of cluster slices per viewable depth.

7.1.1.578 SceneLightsSetViewSize()

```
AFTERWARP_API void SceneLightsSetViewSize (
    struct SceneLights_t * sceneLights,
    Point const * viewSize )
```

Returns size of the viewable area.

7.1.1.579 SceneMeshAutoDraw()

```
AFTERWARP_API LibraryBool SceneMeshAutoDraw (
    struct SceneMesh_t * sceneMesh,
    struct Scene_t * scene,
    ObjectMaterial const * material,
    FloatColor const * color,
    PrimitiveTopology topology,
    int32_t instanceCount,
    int32_t elementCount,
    int32_t firstIndex,
    int32_t baseVertex,
    uint32_t options,
    _Inout_ int32_t * drawCalls )
```

Issues one or more draw calls for the scene mesh, assigning materials and textures accordingly. If *material* is `NULL`, default material will be used. If *instanceCount* is zero, then non-instancing draw calls will be issued. If *scene* is `NULL`, this does not issue any draw calls, instead counting any tentative draw calls that would be issued; this can be used to determine if any portions of the object would be rendered to decide whether to perform certain pass or not.

7.1.1.580 SceneMeshAutoDrawSliced()

```
AFTERWARP_API LibraryBool SceneMeshAutoDrawSliced (
    struct SceneMesh_t * sceneMesh,
    struct Scene_t * scene,
    ObjectMaterial const * material,
    FloatColor const * color,
    PrimitiveTopology topology,
    int32_t instanceCount,
    uint32_t options,
    _Inout_ int32_t * drawCalls )
```

Issues one or more draw calls for the scene mesh, assigning materials and textures accordingly. If scene mesh is a slice, then renders the portions of the slice accordingly. If *material* is `NULL`, default material will be used. If *instanceCount* is zero, then non-instancing draw call is issued. If *scene* is `NULL`, this does not issue any draw calls, instead counting any tentative draw calls that would be issued; this can be used to determine if any portions of the object would be rendered to decide whether to perform certain pass or not.

7.1.1.581 SceneMeshesAdd()

```
AFTERWARP_API struct SceneMesh_t * SceneMeshesAdd (
    struct SceneMeshes_t * sceneMeshes,
    char const * name,
    ObjectPayload payload,
    Vector const * boundsMin,
    Vector const * boundsMax,
    float scale )
```

Creates a new mesh with the given parameters.

7.1.1.582 SceneMeshesAddFromBuffer()

```
AFTERWARP_API struct SceneMesh_t * SceneMeshesAddFromBuffer (
    struct SceneMeshes_t * sceneMeshes,
    char const * name,
    struct MeshBuffer_t * meshBuffer,
    ObjectPayload payload,
    VertexElement const vertexElements[],
    int32_t vertexElementCount,
    _In_opt_ Vector const * boundsMin,
    _In_opt_ Vector const * boundsMax,
    uint32_t channel,
    uint32_t semanticIndex,
    float scale )
```

Creates a new mesh, loading contents from a given mesh buffer.

7.1.1.583 SceneMeshesAddFromFile()

```
AFTERWARP_API struct SceneMesh_t * SceneMeshesAddFromFile (
    struct SceneMeshes_t * sceneMeshes,
    char const * name,
    char const * fileName,
    ObjectPayload payload,
    VertexElement const vertexElements[],
    int32_t vertexElementCount,
    uint32_t channel,
    uint32_t semanticIndex,
    _Inout_ uint32_t * options,
    float scale,
    MeshLoadSaveFeedback feedback,
    void * feedbackUser,
    char * debug,
    _Inout_ int32_t * debugLength )
```

Creates a new mesh, loading contents from a file on disk, which can be either a Wavefront OBJ file (with ".obj" extension), or a proprietary binary file (with ".mesh" extension). If the source file is in OBJ format, this function will also try to automatically load an accompanying voxel file (same file name, but with ".voxel" extension) and/or latches (".latch" extension), if such is present at the same path.

7.1.1.584 SceneMeshesClear()

```
AFTERWARP_API void SceneMeshesClear (
    struct SceneMeshes_t * sceneMeshes )
```

Removes all existing meshes.

7.1.1.585 SceneMeshesCreate()

```
AFTERWARP_API struct SceneMeshes_t * SceneMeshesCreate (
    struct Device_t * device )
```

Creates new instance of the container associated with a particular device.

7.1.1.586 SceneMeshesDestroy()

```
AFTERWARP_API void SceneMeshesDestroy (
    struct SceneMeshes_t * sceneMeshes )
```

Releases the container and its owned resources.

7.1.1.587 SceneMeshesErase()

```
AFTERWARP_API void SceneMeshesErase (
    struct SceneMeshes_t * sceneMeshes,
    struct SceneMesh_t * sceneMesh )
```

Erases the given mesh.

7.1.1.588 SceneMeshesGetCount()

```
AFTERWARP_API int32_t SceneMeshesGetCount (
    struct SceneMeshes_t * sceneMeshes )
```

Returns number of existing meshes.

7.1.1.589 SceneMeshesGetDevice()

```
AFTERWARP_API struct Device_t * SceneMeshesGetDevice (
    struct SceneMeshes_t * sceneMeshes )
```

Returns device associated with the container.

7.1.1.590 SceneMeshesGetMeshByIndex()

```
AFTERWARP_API struct SceneMesh_t * SceneMeshesGetMeshByIndex (
    struct SceneMeshes_t * sceneMeshes,
    int32_t index )
```

Returns mesh with the given index. Note that adding or removing meshes invalidates all previous mesh indices.

7.1.1.591 SceneMeshesGetMeshByName()

```
AFTERWARP_API struct SceneMesh_t * SceneMeshesGetMeshByName (
    struct SceneMeshes_t * sceneMeshes,
    char const * name )
```

Returns mesh with the given name (case-insensitive) or NULL if no mesh with such name exists.

7.1.1.592 SceneMeshesPayload()

```
AFTERWARP_API struct SceneMesh_t * SceneMeshesPayload (
    struct SceneMeshes_t * sceneMeshes,
    ObjectPayload payload )
```

Returns a mesh that corresponds to the given payload or NULL if such doesn't exist.

7.1.1.593 SceneMeshesSlice()

```
AFTERWARP_API LibraryBool SceneMeshesSlice (
    struct SceneMeshes_t * sceneMeshes,
    struct SceneMesh_t * parent,
    uint8_t type,
    char const * prefix )
```

Splits the given parent mesh into "dummy" sub-meshes by specific meta-tag type. Each sub-mesh splitted this way will refer to parent and one of its tag accordingly. This enables sharing the same 3D model for rendering, while having customized voxel representation and/or latches for each splitted segment.

7.1.1.594 SceneMeshGetBounds()

```
AFTERWARP_API void SceneMeshGetBounds (
    struct SceneMesh_t * sceneMesh,
    Vector * boundsMin,
    Vector * boundsMax )
```

Returns minimum and maximum mesh boundaries.

7.1.1.595 SceneMeshGetLatches()

```
AFTERWARP_API struct SceneMeshLatches_t * SceneMeshGetLatches (
    struct SceneMesh_t * sceneMesh )
```

Returns an integrated collection of latches associated with the 3D mesh.

7.1.1.596 SceneMeshGetMaterials()

```
AFTERWARP_API struct SceneMeshMaterials_t * SceneMeshGetMaterials (
    struct SceneMesh_t * sceneMesh )
```

Returns materials associated with a 3D mesh.

7.1.1.597 SceneMeshGetModel()

```
AFTERWARP_API struct MeshModel_t * SceneMeshGetModel (
    struct SceneMesh_t * sceneMesh )
```

Returns a renderable mesh model.

7.1.1.598 SceneMeshGetName()

```
AFTERWARP_API void SceneMeshGetName (
    struct SceneMesh_t * sceneMesh,
    char * meshName,
    _Inout_ int32_t * meshNameLength )
```

Returns name of the mesh.

7.1.1.599 SceneMeshGetPayload()

```
AFTERWARP_API ObjectPayload SceneMeshGetPayload (
    struct SceneMesh_t * sceneMesh )
```

Returns payload associated with the mesh.

7.1.1.600 SceneMeshGetScale()

```
AFTERWARP_API float SceneMeshGetScale (
    struct SceneMesh_t * sceneMesh )
```

Returns scale of the mesh.

7.1.1.601 SceneMeshGetSize()

```
AFTERWARP_API void SceneMeshGetSize (
    struct SceneMesh_t * sceneMesh,
    Vector * size )
```

Returns size of the mesh.

7.1.1.602 SceneMeshGetSlice()

```
AFTERWARP_API void SceneMeshGetSlice (
    struct SceneMesh_t * sceneMesh,
    _Out_ struct SceneMesh_t ** parent,
    _Out_ struct MeshMetaTag_t ** parentTag )
```

Returns parent mesh that contains the actual 3D model to be rendered and tag that contains information about what portion of parent mesh should be rendered

7.1.1.603 SceneMeshGetTags()

```
AFTERWARP_API struct MeshMetaTags_t * SceneMeshGetTags (
    struct SceneMesh_t * sceneMesh )
```

Returns meta-tags that represent sub-meshes within a 3D mesh.

7.1.1.604 SceneMeshGetVertexElementsIndex()

```
AFTERWARP_API uint8_t SceneMeshGetVertexElementsIndex (
    struct SceneMesh_t * sceneMesh )
```

Returns index of vertex elements that describe the format of mesh vertex buffer. A value of 0xFFu means index is undefined.

7.1.1.605 SceneMeshGetVoxel()

```
AFTERWARP_API struct MeshVoxel_t * SceneMeshGetVoxel (
    struct SceneMesh_t * sceneMesh )
```

Returns voxel representation of the mesh.

7.1.1.606 SceneMeshLatchesAdd()

```
AFTERWARP_API int32_t SceneMeshLatchesAdd (
    struct SceneMeshLatches_t * latches,
    _In_opt_ char const * name,
    int32_t type,
    int32_t group,
    _In_opt_ Vector const * position,
    _In_opt_ Quaternion const * orientation )
```

Adds a new latch to the container and returns its index. In case of memory allocation failure, returns -1.

7.1.1.607 SceneMeshLatchesClear()

```
AFTERWARP_API void SceneMeshLatchesClear (
    struct SceneMeshLatches_t * latches )
```

Removes and releases all existing latches.

7.1.1.608 SceneMeshLatchesCopy()

```
AFTERWARP_API LibraryBool SceneMeshLatchesCopy (
    struct SceneMeshLatches_t * latches,
    struct SceneMeshLatches_t * latchesAnother )
```

Copies latches from another collection.

7.1.1.609 SceneMeshLatchesCreate()

```
AFTERWARP_API struct SceneMeshLatches_t * SceneMeshLatchesCreate (
    void )
```

Creates a new 3D scene mesh latches container.

7.1.1.610 SceneMeshLatchesDestroy()

```
AFTERWARP_API void SceneMeshLatchesDestroy (
    struct SceneMeshLatches_t * latches )
```

Releases an existing instance of 3D scene mesh latches container.

7.1.1.611 SceneMeshLatchesErase()

```
AFTERWARP_API void SceneMeshLatchesErase (
    struct SceneMeshLatches_t * latches,
    int32_t index )
```

Removes an existing latch with the given index.

7.1.1.612 SceneMeshLatchesGetCount()

```
AFTERWARP_API int32_t SceneMeshLatchesGetCount (
    struct SceneMeshLatches_t * latches )
```

Returns total number of existing latches.

7.1.1.613 SceneMeshLatchesGetLatch()

```
AFTERWARP_API LibraryBool SceneMeshLatchesGetLatch (
    struct SceneMeshLatches_t * latches,
    int32_t index,
    SceneMeshLatch * latch )
```

Returns an existing latch with the given index.

7.1.1.614 SceneMeshLatchesGetLatchIndex()

```
AFTERWARP_API int32_t SceneMeshLatchesGetLatchIndex (
    struct SceneMeshLatches_t * latches,
    char const * name )
```

Returns an index of a latch with the given name (case-insensitive) index or -1 if it doesn't exist.

7.1.1.615 SceneMeshLatchesGetWaypointCouple()

```
AFTERWARP_API void SceneMeshLatchesGetWaypointCouple (
    struct SceneMeshLatches_t * latches,
    int32_t group,
    float distance,
    Vector * position,
    Quaternion * orientation )
```

Calculates interpolated position and orientation based on the given distance for the given latch group.

7.1.1.616 SceneMeshLatchesGetWaypointDistance()

```
AFTERWARP_API float SceneMeshLatchesGetWaypointDistance (
    struct SceneMeshLatches_t * latches,
    int32_t group )
```

Returns calculated waypoint traveling distance for the given latch group.

7.1.1.617 SceneMeshLatchesInvalidateWaypoints()

```
AFTERWARP_API void SceneMeshLatchesInvalidateWaypoints (
    struct SceneMeshLatches_t * latches )
```

Triggers recalculation of the interpolated waypoint positions. This should be done when one or more latches have been modified.

7.1.1.618 SceneMeshLatchesLoadFromFile()

```
AFTERWARP_API LibraryBool SceneMeshLatchesLoadFromFile (
    struct SceneMeshLatches_t * latches,
    char const * fileName )
```

Loads a collection of latches from file on disk.

7.1.1.619 SceneMeshLatchesLoadFromFileInMemory()

```
AFTERWARP_API LibraryBool SceneMeshLatchesLoadFromFileInMemory (
    struct SceneMeshLatches_t * latches,
    void * buffer,
    uint32_t bufferSize )
```

Loads a collection of latches from file in memory.

7.1.1.620 SceneMeshLatchesSaveToFile()

```
AFTERWARP_API LibraryBool SceneMeshLatchesSaveToFile (
    struct SceneMeshLatches_t * latches,
    char const * fileName )
```

Saves a collection of latches to file on disk.

7.1.1.621 SceneMeshLatchesSetLatch()

```
AFTERWARP_API LibraryBool SceneMeshLatchesSetLatch (
    struct SceneMeshLatches_t * latches,
    int32_t index,
    SceneMeshLatch const * latch )
```

Updates an existing latch with the given index.

7.1.1.622 SceneMeshLatchesTakeAway()

```
AFTERWARP_API void SceneMeshLatchesTakeAway (
    struct SceneMeshLatches_t * latches,
    struct SceneMeshLatches_t * latchesAnother )
```

Takes away the internal contents from another collection without doing any copying.

7.1.1.623 SceneMeshMaterialAddRange()

```
AFTERWARP_API int32_t SceneMeshMaterialAddRange (
    struct SceneMeshMaterial_t * material,
    SceneMeshMaterialRange const * range )
```

Adds a given range to the list of existing ranges and returns its index. In case of memory allocation failure, returns -1.

7.1.1.624 SceneMeshMaterialClearRanges()

```
AFTERWARP_API void SceneMeshMaterialClearRanges (
    struct SceneMeshMaterial_t * material )
```

Removes all existing ranges.

7.1.1.625 SceneMeshMaterialCommit()

```
AFTERWARP_API LibraryBool SceneMeshMaterialCommit (
    struct SceneMeshMaterial_t * material,
    struct Device_t * device,
    TextureAttributes attributes )
```

Converts existing "scratch" textures into hardware-based textures.

7.1.1.626 SceneMeshMaterialCopy()

```
AFTERWARP_API LibraryBool SceneMeshMaterialCopy (
    struct SceneMeshMaterial_t * material,
    struct SceneMeshMaterial_t * source )
```

Copies contents from another material. In case of textures, creates new instances of textures and copies the contents of textures from the source.

7.1.1.627 SceneMeshMaterialGetName()

```
AFTERWARP_API void SceneMeshMaterialGetName (
    struct SceneMeshMaterial_t * material,
    char * name,
    _Inout_ int32_t * nameLength )
```

Returns material name. The pointed value of `nameLength` defines the maximum number of characters, including null-terminating character, that can be copied. The actual string length is stored in `nameLength`.

7.1.1.628 SceneMeshMaterialGetRange()

```
AFTERWARP_API LibraryBool SceneMeshMaterialGetRange (
    struct SceneMeshMaterial_t * material,
    int32_t index,
    _Out_ SceneMeshMaterialRange * range )
```

Returns a range with the given index.

7.1.1.629 SceneMeshMaterialGetRangeCount()

```
AFTERWARP_API int32_t SceneMeshMaterialGetRangeCount (
    struct SceneMeshMaterial_t * material )
```

Returns number of existing ranges.

7.1.1.630 SceneMeshMaterialGetShading()

```
AFTERWARP_API void SceneMeshMaterialGetShading (
    struct SceneMeshMaterial_t * material,
    SceneMeshMaterialShading * shading )
```

Returns shading parameters of the material.

7.1.1.631 SceneMeshMaterialGetTexture()

```
AFTERWARP_API struct Texture_t * SceneMeshMaterialGetTexture (
    struct SceneMeshMaterial_t * material,
    uint8_t type )
```

Returns a texture associated with the given type or NULL if such does not exist.

7.1.1.632 SceneMeshMaterialReleaseTextures()

```
AFTERWARP_API void SceneMeshMaterialReleaseTextures (
    struct SceneMeshMaterial_t * material )
```

Releases existing textures.

7.1.1.633 SceneMeshMaterialsAdd()

```
AFTERWARP_API int32_t SceneMeshMaterialsAdd (
    struct SceneMeshMaterials_t * materials,
    char const * name )
```

Adds a new material to the scene mesh and returns its index. In case of memory allocation failure, returns -1.

7.1.1.634 SceneMeshMaterialsClear()

```
AFTERWARP_API void SceneMeshMaterialsClear (
    struct SceneMeshMaterials_t * materials )
```

Removes and releases all existing materials.

7.1.1.635 SceneMeshMaterialsCommit()

```
AFTERWARP_API LibraryBool SceneMeshMaterialsCommit (
    struct SceneMeshMaterials_t * materials,
    struct Device_t * device,
    TextureAttributes attributes )
```

Converts existing "scratch" textures into hardware-based textures.

7.1.1.636 SceneMeshMaterialsCopy()

```
AFTERWARP_API LibraryBool SceneMeshMaterialsCopy (
    struct SceneMeshMaterials_t * materials,
    struct SceneMeshMaterials_t * materialsAnother )
```

Copies materials into the collection from another container.

7.1.1.637 SceneMeshMaterialsCreate()

```
AFTERWARP_API struct SceneMeshMaterials_t * SceneMeshMaterialsCreate (
    void )
```

Creates a new 3D scene mesh material container.

7.1.1.638 SceneMeshMaterialsDestroy()

```
AFTERWARP_API void SceneMeshMaterialsDestroy (
    struct SceneMeshMaterials_t * materials )
```

Releases an existing instance of 3D scene mesh material container.

7.1.1.639 SceneMeshMaterialsErase()

```
AFTERWARP_API void SceneMeshMaterialsErase (
    struct SceneMeshMaterials_t * materials,
    int32_t index )
```

Removes a material with the given index.

7.1.1.640 SceneMeshMaterialSetRange()

```
AFTERWARP_API LibraryBool SceneMeshMaterialSetRange (
    struct SceneMeshMaterial_t * material,
    int32_t index,
    SceneMeshMaterialRange const * range )
```

Updates a range with the given index.

7.1.1.641 SceneMeshMaterialSetShading()

```
AFTERWARP_API void SceneMeshMaterialSetShading (
    struct SceneMeshMaterial_t * material,
    SceneMeshMaterialShading const * shading )
```

Updates shading parameters of the material.

7.1.1.642 SceneMeshMaterialSetTexture()

```
AFTERWARP_API LibraryBool SceneMeshMaterialSetTexture (
    struct SceneMeshMaterial_t * material,
    struct Texture_t * texture,
    uint8_t type )
```

Changes texture of the given type. Note: material will take ownership of the given texture instance and relinquish ownership of instance that was previously assigned (if it is not NULL).

7.1.1.643 SceneMeshMaterialsGetCount()

```
AFTERWARP_API int32_t SceneMeshMaterialsGetCount (
    struct SceneMeshMaterials_t * materials )
```

Returns total number of existing materials.

7.1.1.644 SceneMeshMaterialsGetMaterial()

```
AFTERWARP_API struct SceneMeshMaterial_t * SceneMeshMaterialsGetMaterial (
    struct SceneMeshMaterials_t * materials,
    int32_t index )
```

Returns an existing material with the given index, if such exists.

7.1.1.645 SceneMeshMaterialsGetName()

```
AFTERWARP_API void SceneMeshMaterialsGetName (
    struct SceneMeshMaterials_t * materials,
    char * name,
    _Inout_ int32_t * nameLength )
```

Returns name of the library.

7.1.1.646 SceneMeshMaterialsSetName()

```
AFTERWARP_API LibraryBool SceneMeshMaterialsSetName (
    struct SceneMeshMaterials_t * materials,
    char const * name )
```

Changes the name of the library.

7.1.1.647 SceneMeshMaterialsTakeAway()

```
AFTERWARP_API void SceneMeshMaterialsTakeAway (
    struct SceneMeshMaterials_t * materials,
    struct SceneMeshMaterials_t * materialsAnother )
```

Takes away the internal contents from another material container without doing any copying.

7.1.1.648 SceneMeshMaterialsTexturing()

```
AFTERWARP_API LibraryBool SceneMeshMaterialsTexturing (
    struct SceneMeshMaterials_t * materials )
```

Reviews existing materials to determine whether any of them use texturing.

7.1.1.649 SceneMeshSetSlice()

```
AFTERWARP_API void SceneMeshSetSlice (
    struct SceneMesh_t * sceneMesh,
    struct SceneMesh_t * parent,
    struct MeshMetaTag_t * parentTag )
```

Specify the parent mesh and the tag to which this slice would refer to.

7.1.1.650 SceneMeshSetVertexElementsIndex()

```
AFTERWARP_API void SceneMeshSetVertexElementsIndex (
    struct SceneMesh_t * sceneMesh,
    uint8_t vertexElementsIndex )
```

Changes index of vertex elements that describe the format of mesh vertex buffer. A value of 0xFFu means index is undefined.

7.1.1.651 ScenePrepare()

```
AFTERWARP_API LibraryBool ScenePrepare (
    struct Scene_t * scene )
```

Configures rendering parameters for the scene, retrieves light indices from light container module and updates internal buffers required for rendering the scene. This must be called outside of `SceneBegin()` and `SceneEnd()` block, but before starting any rendering. Ambient occlusion texture, shadow casting atlas, view and projection matrices have been assigned prior this call.

7.1.1.652 SceneRendering()

```
AFTERWARP_API LibraryBool SceneRendering (
    struct Scene_t * scene )
```

Indicates that the rendering is currently taking place (inside `SceneBegin()` / `SceneEnd()` block).

7.1.1.653 SceneResetCache()

```
AFTERWARP_API void SceneResetCache (
    struct Scene_t * scene )
```

Clears cached buffers and resources in the scene.

7.1.1.654 SceneSetActiveVertexElements()

```
AFTERWARP_API LibraryBool SceneSetActiveVertexElements (
    struct Scene_t * scene,
    int32_t index )
```

Changes currently active index of vertex elements declaration.

7.1.1.655 SceneSetAttributes()

```
AFTERWARP_API LibraryBool SceneSetAttributes (
    struct Scene_t * scene,
    SceneAttributes attributes )
```

Updates attributes that define the rendering behavior of the scene.

7.1.1.656 SceneSetLights()

```
AFTERWARP_API void SceneSetLights (
    struct Scene_t * scene,
    struct SceneLights_t * sceneLights )
```

Changes light container module associated with the scene. Note: this must be called outside of `SceneBegin()` and `SceneEnd()` block and before calling `ScenePrepare()` function.

7.1.1.657 SceneSetMaterial()

```
AFTERWARP_API LibraryBool SceneSetMaterial (
    struct Scene_t * scene,
    ObjectMaterial const * material )
```

Updates scene object's material properties.

7.1.1.658 SceneSetParallaxMappingParameters()

```
AFTERWARP_API LibraryBool SceneSetParallaxMappingParameters (
    struct Scene_t * scene,
    ParallaxMappingParameters const * parameters )
```

Updates parameters that define Parallax Mapping technique is performed.

7.1.1.659 SceneSetProjection()

```
AFTERWARP_API void SceneSetProjection (
    struct Scene_t * scene,
    Matrix const * projection )
```

Updates projection matrix.

7.1.1.660 SceneSetShadowCastingAtlas()

```
AFTERWARP_API void SceneSetShadowCastingAtlas (
    struct Scene_t * scene,
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Changes shadow casting atlas associated with the scene. Note: this must be called outside of `SceneBegin()` and `SceneEnd()` block and before calling `ScenePrepare()` function.

7.1.1.661 SceneSetTexture()

```
AFTERWARP_API LibraryBool SceneSetTexture (
    struct Scene_t * scene,
    struct Texture_t * texture,
    SceneTextureType type )
```

Updates texture of the given type associated with the scene.

7.1.1.662 SceneSetTextureCabinet()

```
AFTERWARP_API void SceneSetTextureCabinet (
    struct Scene_t * scene,
    struct TextureCabinet_t * textureCabinet )
```

Updates an association with a texture cabinet. Note: this must be called outside of `SceneBegin()` and `SceneEnd()` block and before calling `ScenePrepare()` function.

7.1.1.663 SceneSetToneMappingCoefficients()

```
AFTERWARP_API LibraryBool SceneSetToneMappingCoefficients (
    struct Scene_t * scene,
    Vector4 const * coefficients )
```

Updates tone-mapping RGB luma coefficients and maximum allowed level of white. These parameters are used only when "glassy" (OIT) technique is being performed.

7.1.1.664 SceneSetVertexElements()

```
AFTERWARP_API LibraryBool SceneSetVertexElements (
    struct Scene_t * scene,
    int32_t index,
    VertexElement const vertexElements[],
    int32_t vertexElementCount )
```

Changes vertex element declaration associated with the given index.

7.1.1.665 SceneSetVertexElementsFromTextModeller()

```
AFTERWARP_API LibraryBool SceneSetVertexElementsFromTextModeller (
    struct Scene_t * scene,
    int32_t index )
```

Changes vertex element declaration to match the one of 2D and 3D text rendering module, associated with the given index.

7.1.1.666 SceneSetView()

```
AFTERWARP_API void SceneSetView (
    struct Scene_t * scene,
    Matrix const * view )
```

Updates view matrix.

7.1.1.667 SceneSetWorld()

```
AFTERWARP_API void SceneSetWorld (
    struct Scene_t * scene,
    Matrix const * world )
```

Updates object world matrix.

7.1.1.668 SelectionHighlightBegin()

```
AFTERWARP_API LibraryBool SelectionHighlightBegin (
    struct SelectionHighlight_t * selectionHighlight )
```

Begins rendering the primary textures.

7.1.1.669 SelectionHighlightClear()

```
AFTERWARP_API LibraryBool SelectionHighlightClear (
    struct SelectionHighlight_t * selectionHighlight )
```

Clears the primary rendering textures.

7.1.1.670 SelectionHighlightCreate()

```
AFTERWARP_API struct SelectionHighlight_t * SelectionHighlightCreate (
    struct Device_t * device,
    Point const * size,
    PixelFormat depthStencil,
    int32_t samples,
    LibraryBool grayscale )
```

Creates a new instance of selection highlight module.

7.1.1.671 SelectionHighlightDestroy()

```
AFTERWARP_API void SelectionHighlightDestroy (
    struct SelectionHighlight_t * selectionHighlight )
```

Releases given instance of selection highlight module.

7.1.1.672 SelectionHighlightEnd()

```
AFTERWARP_API void SelectionHighlightEnd (
    struct SelectionHighlight_t * selectionHighlight )
```

Ends rendering to the primary textures.

7.1.1.673 SelectionHighlightFilter()

```
AFTERWARP_API LibraryBool SelectionHighlightFilter (
    struct SelectionHighlight_t * selectionHighlight )
```

Performs final post-filtering of the textures to produce the highlight selection effect.

7.1.1.674 SelectionHighlightGetDepthStencil()

```
AFTERWARP_API PixelFormat SelectionHighlightGetDepthStencil (
    struct SelectionHighlight_t * selectionHighlight )
```

Returns pixel format used for depth/stencil buffer.

7.1.1.675 SelectionHighlightGetDevice()

```
AFTERWARP_API struct Device_t * SelectionHighlightGetDevice (
    struct SelectionHighlight_t * selectionHighlight )
```

Returns device associated with the given module.

7.1.1.676 SelectionHighlightGetGrayscale()

```
AFTERWARP_API LibraryBool SelectionHighlightGetGrayscale (
    struct SelectionHighlight_t * selectionHighlight )
```

Returns whether the technique is performed in a more optimal, grayscale mode, instead of full color.

7.1.1.677 SelectionHighlightGetParameters()

```
AFTERWARP_API void SelectionHighlightGetParameters (
    struct SelectionHighlight_t * selectionHighlight,
    SelectionHighlightParameters * parameters )
```

Returns parameters that define the behavior and characteristics of the technique.

7.1.1.678 SelectionHighlightGetSamples()

```
AFTERWARP_API int32_t SelectionHighlightGetSamples (
    struct SelectionHighlight_t * selectionHighlight )
```

Returns number of samples used for multisampling.

7.1.1.679 SelectionHighlightGetSize()

```
AFTERWARP_API void SelectionHighlightGetSize (
    struct SelectionHighlight_t * selectionHighlight,
    Point * size )
```

Returns size of the rendering surfaces.

7.1.1.680 SelectionHighlightGetTexture()

```
AFTERWARP_API struct Texture_t * SelectionHighlightGetTexture (
    struct SelectionHighlight_t * selectionHighlight,
    SelectionHighlightTextureType type )
```

Returns texture of the given type.

7.1.1.681 SelectionHighlightGetTextureCoordinates()

```
AFTERWARP_API void SelectionHighlightGetTextureCoordinates (
    struct SelectionHighlight_t * selectionHighlight,
    Quad * coords )
```

Returns texture coordinates that can be passed to canvas for rendering the highlight.

7.1.1.682 SelectionHighlightRendering()

```
AFTERWARP_API LibraryBool SelectionHighlightRendering (
    struct SelectionHighlight_t * selectionHighlight )
```

Indicates that the rendering is currently performed to the primary textures.

7.1.1.683 SelectionHighlightSetParameters()

```
AFTERWARP_API LibraryBool SelectionHighlightSetParameters (
    struct SelectionHighlight_t * selectionHighlight,
    SelectionHighlightParameters const * parameters )
```

Updates parameters that define the behavior and characteristics of the technique.

7.1.1.684 SelectionHighlightSetSize()

```
AFTERWARP_API LibraryBool SelectionHighlightSetSize (
    struct SelectionHighlight_t * selectionHighlight,
    Point const * size )
```

Updates size of the rendering surfaces.

7.1.1.685 ShadowCasterBegin()

```
AFTERWARP_API LibraryBool ShadowCasterBegin (
    struct ShadowCaster_t * shadowCaster )
```

Starts rendering to shadow caster's texture to build a shadow map.

7.1.1.686 ShadowCasterClear()

```
AFTERWARP_API LibraryBool ShadowCasterClear (
    struct ShadowCaster_t * shadowCaster )
```

Clears the integrated textures.

7.1.1.687 ShadowCasterEnd()

```
AFTERWARP_API void ShadowCasterEnd (
    struct ShadowCaster_t * shadowCaster )
```

Finishes rendering to shadow caster's texture.

7.1.1.688 ShadowCasterFilter()

```
AFTERWARP_API LibraryBool ShadowCasterFilter (
    struct ShadowCaster_t * shadowCaster )
```

Performs filtering on the shadow map and accomodates it on the atlas.

7.1.1.689 ShadowCasterGetAtlas()

```
AFTERWARP_API struct ShadowCastingAtlas_t * ShadowCasterGetAtlas (
    struct ShadowCaster_t * shadowCaster )
```

Returns pointer to an owner atlas.

7.1.1.690 ShadowCasterGetDevice()

```
AFTERWARP_API struct Device_t * ShadowCasterGetDevice (
    struct ShadowCaster_t * shadowCaster )
```

Returns device associated with the given shadow shadowCaster.

7.1.1.691 ShadowCasterGetPosition()

```
AFTERWARP_API void ShadowCasterGetPosition (
    struct ShadowCaster_t * shadowCaster,
    Point * position )
```

Returns position of the shadow map in parent atlas.

7.1.1.692 ShadowCasterGetSize()

```
AFTERWARP_API void ShadowCasterGetSize (
    struct ShadowCaster_t * shadowCaster,
    Point * size )
```

Returns size of the shadow map.

7.1.1.693 ShadowCasterGetTexture()

```
AFTERWARP_API struct Texture_t * ShadowCasterGetTexture (
    struct ShadowCaster_t * shadowCaster,
    int32_t index )
```

Returns one of the textures currently used by the shadow caster.

7.1.1.694 ShadowCasterGetViewProjection()

```
AFTERWARP_API void ShadowCasterGetViewProjection (
    struct ShadowCaster_t * shadowCaster,
    Matrix * viewProjection )
```

Returns a combined view and projection matrices of the shadow caster.

7.1.1.695 ShadowCasterRendering()

```
AFTERWARP_API LibraryBool ShadowCasterRendering (
    struct ShadowCaster_t * shadowCaster )
```

Indicates whether rendering is taking place.

7.1.1.696 ShadowCasterSetViewProjection()

```
AFTERWARP_API void ShadowCasterSetViewProjection (
    struct ShadowCaster_t * shadowCaster,
    Matrix const * viewProjection )
```

Updates a combined view and projection matrices of the shadow caster.

7.1.1.697 ShadowCastingAtlasAdd()

```
AFTERWARP_API struct ShadowCaster_t * ShadowCastingAtlasAdd (
    struct ShadowCastingAtlas_t * shadowCastingAtlas,
    Point const * size,
    LibraryBool shared )
```

Creates a new shadow caster with the given texture size. Shared parameter determines whether the caster should share its shadow map textures with others to reduce memory footprint. For "progressive" rendering, where more portions are added to shadow map in next frames, the shadow caster must own its shadow map.

7.1.1.698 ShadowCastingAtlasBorderFill()

```
AFTERWARP_API LibraryBool ShadowCastingAtlasBorderFill (
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Clears the surface of shadow mapping atlas texture using border color to avoid edge artifacts. This is typically required after shadow parameters have been changed.

7.1.1.699 ShadowCastingAtlasClear()

```
AFTERWARP_API void ShadowCastingAtlasClear (
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Removes all existing shadow casters.

7.1.1.700 ShadowCastingAtlasCreate()

```
AFTERWARP_API struct ShadowCastingAtlas_t * ShadowCastingAtlasCreate (
    struct Device_t * device,
    Point const * size,
    TechniqueShadows technique,
    int32_t samples,
    int32_t padding )
```

Creates a new instance of shadow caster and map management module.

7.1.1.701 ShadowCastingAtlasDestroy()

```
AFTERWARP_API void ShadowCastingAtlasDestroy (
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Releases given instance of shadow caster and map management module.

7.1.1.702 ShadowCastingAtlasErase()

```
AFTERWARP_API void ShadowCastingAtlasErase (
    struct ShadowCastingAtlas_t * shadowCastingAtlas,
    struct ShadowCaster_t * caster )
```

Removes an existing shadow caster.

7.1.1.703 ShadowCastingAtlasGetCaster()

```
AFTERWARP_API struct ShadowCaster_t * ShadowCastingAtlasGetCaster (
    struct ShadowCastingAtlas_t * shadowCastingAtlas,
    int32_t index )
```

Returns shadow caster for the given index.

7.1.1.704 ShadowCastingAtlasGetCount()

```
AFTERWARP_API int32_t ShadowCastingAtlasGetCount (
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Returns number of existing shadow casters.

7.1.1.705 ShadowCastingAtlasGetDevice()

```
AFTERWARP_API struct Device_t * ShadowCastingAtlasGetDevice (
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Returns device associated with the given textures collection.

7.1.1.706 ShadowCastingAtlasGetPadding()

```
AFTERWARP_API int32_t ShadowCastingAtlasGetPadding (
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Returns padding used to accomodate shadow maps.

7.1.1.707 ShadowCastingAtlasGetParameters()

```
AFTERWARP_API void ShadowCastingAtlasGetParameters (
    struct ShadowCastingAtlas_t * shadowCastingAtlas,
    ShadowParameters * parameters )
```

Returns parameters that define how shadows are rendered.

7.1.1.708 ShadowCastingAtlasGetSamples()

```
AFTERWARP_API int32_t ShadowCastingAtlasGetSamples (
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Returns number of samples used for shadow map multisampling.

7.1.1.709 ShadowCastingAtlasGetSize()

```
AFTERWARP_API void ShadowCastingAtlasGetSize (
    struct ShadowCastingAtlas_t * shadowCastingAtlas,
    Point * size )
```

Returns size of the shadow-casting shadowCastingAtlas.

7.1.1.710 ShadowCastingAtlasGetTechnique()

```
AFTERWARP_API TechniqueShadows ShadowCastingAtlasGetTechnique (
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Returns shadow rendering technique.

7.1.1.711 ShadowCastingAtlasGetTexture()

```
AFTERWARP_API struct Texture_t * ShadowCastingAtlasGetTexture (
    struct ShadowCastingAtlas_t * shadowCastingAtlas )
```

Returns shadow mapping atlas texture containing existing shadow maps.

7.1.1.712 ShadowCastingAtlasSetParameters()

```
AFTERWARP_API LibraryBool ShadowCastingAtlasSetParameters (
    struct ShadowCastingAtlas_t * shadowCastingAtlas,
    ShadowParameters const * parameters )
```

Updates shadow rendering parameters.

7.1.1.713 SineAccelerate()

```
AFTERWARP_API float SineAccelerate (
    float value )
```

Transforms value in range of [0, 1] using "accelerating" sine wave. The curve starts at zero with almost no "velocity", while in the end incrementing almost linearly.

7.1.1.714 SineCycle()

```
AFTERWARP_API float SineCycle (
    float value )
```

Transforms value in range of [0, 1] to go through full sine curve in range [0 -> 1 -> 0]. Note that the curve goes from zero to one and then back to zero.

7.1.1.715 SineDecelerate()

```
AFTERWARP_API float SineDecelerate (
    float value )
```

Transforms value in range of [0, 1] using "decelerating" sine wave. The curve starts at zero, incrementing almost linearly, while in the end going to almost a complete stop.

7.1.1.716 SineTransform()

```
AFTERWARP_API float SineTransform (
    float value )
```

Transforms value in range of [0, 1] by using sine wave (kind of "accelerate then decelerate").

7.1.1.717 SineTwoCycle()

```
AFTERWARP_API float SineTwoCycle (
    float value )
```

Transforms value in range of [0, 1] to go full cycle through sine function (0 -> 1 -> 0 -> -1 -> 0).

7.1.1.718 SmootherStep()

```
AFTERWARP_API float SmootherStep (
    float value1,
    float value2,
    float theta )
```

Interpolates between two values using Hermite curve after applying saturation, where theta parameter is specified in [0, 1] range. This uses Perlin's improvement, where 1st and 2nd order derivatives at positions 0 and 1 are zero.

7.1.1.719 SmoothStep()

```
AFTERWARP_API float SmoothStep (
    float value1,
    float value2,
    float theta )
```

Interpolates between two values using simplified Hermite curve after applying saturation, where theta parameter is specified in [0, 1] range.

7.1.1.720 SpatialFogCreate()

```
AFTERWARP_API struct SpatialFog_t * SpatialFogCreate (
    struct Device_t * device )
```

Creates a new instance of spatial fog module.

7.1.1.721 SpatialFogDestroy()

```
AFTERWARP_API void SpatialFogDestroy (
    struct SpatialFog_t * spatialFog )
```

Releases given instance of spatial fog module.

7.1.1.722 SpatialFogExecute()

```
AFTERWARP_API LibraryBool SpatialFogExecute (
    struct SpatialFog_t * spatialFog,
    struct Texture_t * destination,
    struct Texture_t * linearDepths )
```

Computes fog and applies it to the destination high-dynamic range (HDR) color texture. The module requires an existing linear depths texture (optionally multisampled) to be provided.

7.1.1.723 SpatialFogExecuteGlassy()

```
AFTERWARP_API LibraryBool SpatialFogExecuteGlassy (
    struct SpatialFog_t * spatialFog,
    struct Texture_t * destination,
    struct Texture_t * linearDepths )
```

Computes fog and applies it to the destination "glassy" composition texture. The module requires an existing linear depths texture (optionally multisampled) to be provided.

7.1.1.724 SpatialFogGetDepthDistance()

```
AFTERWARP_API DepthFogDistance SpatialFogGetDepthDistance (
    struct SpatialFog_t * spatialFog )
```

Returns distance calculation formula for the fog.

7.1.1.725 SpatialFogGetDevice()

```
AFTERWARP_API struct Device_t * SpatialFogGetDevice (
    struct SpatialFog_t * spatialFog )
```

Returns device associated with the given module.

7.1.1.726 SpatialFogGetFormula()

```
AFTERWARP_API FogFormula SpatialFogGetFormula (
    struct SpatialFog_t * spatialFog )
```

Returns formula used for the fog.

7.1.1.727 SpatialFogGetParameters()

```
AFTERWARP_API void SpatialFogGetParameters (
    struct SpatialFog_t * spatialFog,
    FogParameters * parameters )
```

Returns parameters that define spatial fog characteristics.

7.1.1.728 SpatialFogGetProjection()

```
AFTERWARP_API void SpatialFogGetProjection (
    struct SpatialFog_t * spatialFog,
    Matrix * projection )
```

Returns projection transformation matrix.

7.1.1.729 SpatialFogGetView()

```
AFTERWARP_API void SpatialFogGetView (
    struct SpatialFog_t * spatialFog,
    Matrix * view )
```

Returns view transformation matrix.

7.1.1.730 SpatialFogSetDepthDistance()

```
AFTERWARP_API LibraryBool SpatialFogSetDepthDistance (
    struct SpatialFog_t * spatialFog,
    DepthFogDistance depthDistance )
```

Changes distance calculation formula for the fog.

7.1.1.731 SpatialFogSetFormula()

```
AFTERWARP_API LibraryBool SpatialFogSetFormula (
    struct SpatialFog_t * spatialFog,
    FogFormula formula )
```

Changes formula used for the fog.

7.1.1.732 SpatialFogSetParameters()

```
AFTERWARP_API LibraryBool SpatialFogSetParameters (
    struct SpatialFog_t * spatialFog,
    FogParameters const * parameters )
```

Changes parameters that define spatial fog characteristics.

7.1.1.733 SpatialFogSetProjection()

```
AFTERWARP_API LibraryBool SpatialFogSetProjection (
    struct SpatialFog_t * spatialFog,
    Matrix const * projection )
```

Updates projection transformation matrix.

7.1.1.734 SpatialFogSetView()

```
AFTERWARP_API LibraryBool SpatialFogSetView (
    struct SpatialFog_t * spatialFog,
    Matrix const * view )
```

Updates view transformation matrix.

7.1.1.735 SubtractColors()

```
AFTERWARP_API Color SubtractColors (
    Color color1,
    Color color2 )
```

Subtracts two 32-bit RGBA color values clamping the resulting values.

7.1.1.736 SurfaceClear()

```
AFTERWARP_API void SurfaceClear (
    struct Surface_t * surface )
```

Clears the entire surface with zeros.

7.1.1.737 SurfaceClearWith()

```
AFTERWARP_API LibraryBool SurfaceClearWith (
    struct Surface_t * surface,
    Color color )
```

Clears the entire surface with a given color. This does pixel format conversion when appropriate.

7.1.1.738 SurfaceConvertFormat()

```
AFTERWARP_API LibraryBool SurfaceConvertFormat (
    struct Surface_t * surface,
    PixelFormat format )
```

Converts surface from its current format to another one.

7.1.1.739 SurfaceCopyFrom()

```
AFTERWARP_API LibraryBool SurfaceCopyFrom (
    struct Surface_t * surface,
    struct Surface_t * source )
```

Copies entire contents from a source surface to the current one. If the current surface has size and/or pixel format not specified, these will be copied from the source surface as well. If current surface is not empty, then its pixel format will not be modified - in this case, pixel format conversion may occur. This function will try to ensure that current surface size matches the source surface and if this cannot be achieved, it will fail.

7.1.1.740 SurfaceCopyRect()

```
AFTERWARP_API LibraryBool SurfaceCopyRect (
    struct Surface_t * surface,
    struct Surface_t * source,
    Rect const * sourceRect,
    Point const * destPos )
```

Copies a portion of source surface to the current one according to specified source rectangle and destination position. If source rectangle is empty (or NULL), then the entire source surface will be copied. This function does the appropriate clipping and pixel format conversion. It does not change current surface size or pixel format. Note that "premultiplied alpha" flag of either current or source surfaces is completely ignored.

7.1.1.741 SurfaceCreate()

```
AFTERWARP_API struct Surface_t * SurfaceCreate (
    int32_t width,
    int32_t height,
    PixelFormat format )
```

Creates a new instance of raster surface.

7.1.1.742 SurfaceCreateFromFile()

```
AFTERWARP_API struct Surface_t * SurfaceCreateFromFile (
    char const * fileName,
    AlphaFormatRequest formatRequest )
```

Creates a new instance of raster surface containing image loaded from external file. The preference for premultiplied or non-premultiplied alpha-channel is merely a suggestion, which can influence how the image is loaded (depending on underlying implementation). This function typically supports most common file formats such as BMP, PNG, JPEG, but may also support other formats like GIF and TIFF.

7.1.1.743 SurfaceCreateFromFileInMemory()

```
AFTERWARP_API struct Surface_t * SurfaceCreateFromFileInMemory (
    void const * fileContents,
    uint32_t contentSize,
    char const * extension,
    AlphaFormatRequest formatRequest )
```

Creates a new instance of raster surface containing image loaded from file that has been preloaded into memory. The internal file format is determined by "extension" parameter, which should contain a valid file extension such as .png (case-insensitive). The preference for premultiplied or non-premultiplied alpha-channel is merely a suggestion, which can influence how the image is loaded (depending on underlying implementation). This function typically supports most common file formats such as BMP, PNG, JPEG, but may also support other formats like GIF and TIFF.

7.1.1.744 SurfaceDestroy()

```
AFTERWARP_API void SurfaceDestroy (
    struct Surface_t * surface )
```

Releases given instance of raster surface.

7.1.1.745 SurfaceEmpty()

```
AFTERWARP_API LibraryBool SurfaceEmpty (
    struct Surface_t * surface )
```

Checks whether the surface has width or height set to zero, has undefined pixel format, unknown pitch, unknown bytes per pixel, or access bits set to NULL. If this returns false, then the surface can be considered valid.

7.1.1.746 SurfaceFlip()

```
AFTERWARP_API LibraryBool SurfaceFlip (
    struct Surface_t * surface )
```

Flips the visible image on surface vertically.

7.1.1.747 SurfaceGetBits()

```
AFTERWARP_API void * SurfaceGetBits (
    struct Surface_t * surface )
```

Returns pointer to top/left corner of pixel data contained by this surface, with horizontal rows arranged linearly from top to bottom.

7.1.1.748 SurfaceGetByteSize()

```
AFTERWARP_API uint32_t SurfaceGetByteSize (
    struct Surface_t * surface )
```

Returns surface size in bytes.

7.1.1.749 SurfaceGetBytesPerPixel()

```
AFTERWARP_API int32_t SurfaceGetBytesPerPixel (
    struct Surface_t * surface )
```

Returns number of bytes each pixel occupies.

7.1.1.750 SurfaceGetFormat()

```
AFTERWARP_API PixelFormat SurfaceGetFormat (
    struct Surface_t * surface )
```

Returns pixel format in which pixels are stored.

7.1.1.751 SurfaceGetHeight()

```
AFTERWARP_API int32_t SurfaceGetHeight (
    struct Surface_t * surface )
```

Returns surface height in pixels.

7.1.1.752 SurfaceGetPitch()

```
AFTERWARP_API uint32_t SurfaceGetPitch (
    struct Surface_t * surface )
```

Returns number of bytes each horizontal row of pixels occupies. This may differ from an actual calculated number and may include unused or even protected memory locations, which should be considered read-only and be skipped.

7.1.1.753 SurfaceGetPixel()

```
AFTERWARP_API Color SurfaceGetPixel (
    struct Surface_t * surface,
    int32_t x,
    int32_t y )
```

Reads a single pixel from the surface.

7.1.1.754 SurfaceGetPixelBilinear()

```
AFTERWARP_API Color SurfaceGetPixelBilinear (
    struct Surface_t * surface,
    float x,
    float y )
```

Retrieves pixel from floating-point coordinates, interpolating linearly between four neighbor pixels as necessary to get an accurate match. This can be used for limitless stretching, to get color values that lie between individual pixels and slowly change from one pixel to another.

7.1.1.755 SurfaceGetPixelPtr()

```
AFTERWARP_API void * SurfaceGetPixelPtr (
    struct Surface_t * surface,
    int32_t x,
    int32_t y )
```

Provides pointer to the pixel data at the given coordinates.

7.1.1.756 SurfaceGetPremultipliedAlpha()

```
AFTERWARP_API LibraryBool SurfaceGetPremultipliedAlpha (
    struct Surface_t * surface )
```

Indicates whether the pixels in this surface have their alpha premultiplied or not. This is just an informative parameter; to actually convert pixels from one mode to another, use [SurfacePremultiplyAlpha\(\)](#) and [SurfaceUnpremultiplyAlpha\(\)](#) methods.

7.1.1.757 SurfaceGetScanline()

```
AFTERWARP_API void * SurfaceGetScanline (
    struct Surface_t * surface,
    int32_t index )
```

Provides pointer to a first pixel at the given scanline index (that is, row number).

7.1.1.758 SurfaceGetWidth()

```
AFTERWARP_API int32_t SurfaceGetWidth (
    struct Surface_t * surface )
```

Returns surface width in pixels.

7.1.1.759 SurfaceHasAlphaChannel()

```
AFTERWARP_API LibraryBool SurfaceHasAlphaChannel (
    struct Surface_t * surface )
```

Processes the whole surface to determine whether it has meaningful alpha-channel. A surface that has all its pixels with alpha-channel set to fully translucent or fully opaque (but not mixed) is considered lacking alpha-channel. On the other hand, a surface that has at least one pixel with alpha-channel value different than any other pixel, is considered to have alpha-channel. This is useful to determine whether the surface can be stored in one of pixel formats lacking alpha-channel, to avoid losing any transparency information.

7.1.1.760 SurfaceInvert()

```
AFTERWARP_API LibraryBool SurfaceInvert (
    struct Surface_t * surface )
```

Inverts colors in the surface.

7.1.1.761 SurfaceMakeSignedDistanceField()

```
AFTERWARP_API LibraryBool SurfaceMakeSignedDistanceField (
    struct Surface_t * surface,
    struct Surface_t * source,
    float spread,
    Point const * destPos,
    Rect const * sourceRect )
```

Calculates signed distance field from alpha values of the source.

7.1.1.762 SurfaceMirror()

```
AFTERWARP_API LibraryBool SurfaceMirror (
    struct Surface_t * surface )
```

Mirrors the visible image on surface horizontally.

7.1.1.763 SurfacePremultiplyAlpha()

```
AFTERWARP_API LibraryBool SurfacePremultiplyAlpha (
    struct Surface_t * surface )
```

Processes the whole surface, premultiplying each pixel's red, green and blue values by the corresponding alpha-channel value, resulting in image with premultiplied alpha. Note that this is an irreversible process, during which some color information is lost permanently (smaller alpha values contribute to bigger information loss). This is generally useful to prepare the image for generating mipmaps and/or alpha-blending, to get more accurate visual results.

7.1.1.764 SurfaceResetAlpha()

```
AFTERWARP_API LibraryBool SurfaceResetAlpha (
    struct Surface_t * surface,
    LibraryBool opaque )
```

Processes surface pixels, setting alpha-channel to either fully translucent or fully opaque.

7.1.1.765 SurfaceResize()

```
AFTERWARP_API LibraryBool SurfaceResize (
    struct Surface_t * surface,
    int32_t width,
    int32_t height,
    PixelFormat format )
```

Redefines surface size to the specified width, height and pixel format, discarding previous contents.

7.1.1.766 SurfaceSaveToFile()

```
AFTERWARP_API LibraryBool SurfaceSaveToFile (
    struct Surface_t * surface,
    char const * fileName,
    int32_t quality )
```

Saves the contents of the surface to external file. This function typically supports most common file formats such as BMP, PNG, JPEG, but may also support other formats like GIF and TIFF. "Quality" parameter, in case of formats such as JPEG, determines the compression ratio (between 0 and 100).

7.1.1.767 SurfaceSaveToFileInMemory()

```
AFTERWARP_API uint32_t SurfaceSaveToFileInMemory (
    struct Surface_t * surface,
    void * fileContents,
    uint32_t fileContentSize,
    char const * extension,
    int32_t quality )
```

Saves the contents of surface to a file in memory and returns the size of resulting file. This function will save up to "fileContentSize" bytes to the provided pointer, even if it means that not a complete file will be saved. If "fileContents" pointer is NULL, then this function will provide an estimate size required to store the file in memory. This function typically supports most common file formats such as BMP, PNG, JPEG, but may also support other formats like GIF and TIFF. `quality` parameter, in case of formats such as JPEG, determines the compression ratio (between 0 and 100).

7.1.1.768 SurfaceSetPixel()

```
AFTERWARP_API void SurfaceSetPixel (
    struct Surface_t * surface,
    int32_t x,
    int32_t y,
    Color color )
```

Writes a single pixel to the surface.

7.1.1.769 SurfaceSetPremultipliedAlpha()

```
AFTERWARP_API void SurfaceSetPremultipliedAlpha (
    struct Surface_t * surface,
    LibraryBool premultipliedAlpha )
```

Changes attribute of premultiplied alpha in the surface. This is just an informative parameter; to actually convert pixels from one mode to another, use [SurfacePremultiplyAlpha\(\)](#) and [SurfaceUnpremultiplyAlpha\(\)](#) methods.

7.1.1.770 SurfaceShrinkFrom()

```
AFTERWARP_API LibraryBool SurfaceShrinkFrom (
    struct Surface_t * surface,
    struct Surface_t * source )
```

This function works similarly to [SurfaceCopyFrom\(\)](#), except that it produces image with half of size, averaging each four pixels to one. This is specifically useful to generate mipmaps.

7.1.1.771 SurfaceStretchRect()

```
AFTERWARP_API LibraryBool SurfaceStretchRect (
    struct Surface_t * surface,
    struct Surface_t * source,
    RectF const * destRect,
    RectF const * sourceRect )
```

This function works similarly to [SurfaceCopyRect\(\)](#), but provides stretching and/or shrinking. That is, it copies source surface rectangle onto destination rectangle with point filtering. Clipping and pixel format conversion is done as necessary.

7.1.1.772 SurfaceStretchRectBilinear()

```
AFTERWARP_API LibraryBool SurfaceStretchRectBilinear (
    struct Surface_t * surface,
    struct Surface_t * source,
    RectF const * destRect,
    RectF const * sourceRect )
```

This function works similarly to [SurfaceCopyRect\(\)](#), but provides bilinear stretching. That is, it copies source surface rectangle onto destination rectangle with linear filtering. Clipping and pixel format conversion is done as necessary. Note that this function is meant for stretching only; shrinking although will also work, but result in inaccurate results as shrinking requires calculating average of variable number of pixels depending on shrink ratio.

7.1.1.773 SurfaceUnpremultiplyAlpha()

```
AFTERWARP_API LibraryBool SurfaceUnpremultiplyAlpha (
    struct Surface_t * surface )
```

Processes the whole surface, dividing each pixel by its alpha-value, resulting in image with non-premultiplied alpha. During this process, some color information may be lost due to precision issues. This can be useful to obtain original pixel information from image that has been previously premultiplied; however, this does not recover lost information during premultiplication process. For instance, pixels that had alpha value of zero and were premultiplied, lose all information and cannot be recovered; pixels with alpha value of 128 (that is, 50% opaque) lose half of their precision and after "unpremultiply" process will have values multiples of 2.

7.1.1.774 SwapChainBegin()

```
AFTERWARP_API LibraryBool SwapChainBegin (
    struct SwapChain_t * swapChain )
```

Begins rendering on the swap-chain.

7.1.1.775 SwapChainCreate()

```
AFTERWARP_API struct SwapChain_t * SwapChainCreate (
    struct Device_t * device,
    UntypedHandle windowHandle,
    Point const * size,
    PixelFormat format,
    PixelFormat depthStencil,
    int32_t multisamples,
    LibraryBool vsync )
```

Creates a new rendering swap-chain associated with the given device and window handle. Once the swap-chain has been created for a particular window under OpenGL-based backends, it would not be possible to remove this association without re-creating the window.

7.1.1.776 SwapChainDestroy()

```
AFTERWARP_API void SwapChainDestroy (
    struct SwapChain_t * swapChain )
```

Releases given instance of rendering swap-chain.

7.1.1.777 SwapChainEnd()

```
AFTERWARP_API void SwapChainEnd (
    struct SwapChain_t * swapChain )
```

Finishes rendering on the swap-chain and flips the buffers.

7.1.1.778 SwapChainGetDepthStencil()

```
AFTERWARP_API PixelFormat SwapChainGetDepthStencil (
    struct SwapChain_t * swapChain )
```

Returns pixel format of swap-chain depth/stencil surface.

7.1.1.779 SwapChainGetDevice()

```
AFTERWARP_API struct Device_t * SwapChainGetDevice (
    struct SwapChain_t * swapChain )
```

Returns device associated with the given swap-chain.

7.1.1.780 SwapChainGetFormat()

```
AFTERWARP_API PixelFormat SwapChainGetFormat (
    struct SwapChain_t * swapChain )
```

Returns pixel format of swap-chain rendering surface.

7.1.1.781 SwapChainGetHeight()

```
AFTERWARP_API int32_t SwapChainGetHeight (
    struct SwapChain_t * swapChain )
```

Returns swap-chain height.

7.1.1.782 SwapChainGetMultisamples()

```
AFTERWARP_API int32_t SwapChainGetMultisamples (
    struct SwapChain_t * swapChain )
```

Returns number of samples of swap-chain surfaces.

7.1.1.783 SwapChainGetVSync()

```
AFTERWARP_API LibraryBool SwapChainGetVSync (
    struct SwapChain_t * swapChain )
```

Returns whether to wait for a vertical retrace.

7.1.1.784 SwapChainGetWidth()

```
AFTERWARP_API int32_t SwapChainGetWidth (
    struct SwapChain_t * swapChain )
```

Returns swap-chain width.

7.1.1.785 SwapChainGetWindowHandle()

```
AFTERWARP_API UntypedHandle SwapChainGetWindowHandle (
    struct SwapChain_t * swapChain )
```

Returns handle of the associated window.

7.1.1.786 SwapChainResize()

```
AFTERWARP_API LibraryBool SwapChainResize (
    struct SwapChain_t * swapChain,
    Point const * size )
```

Resizes the swap-chain.

7.1.1.787 TextModellerClear()

```
AFTERWARP_API void TextModellerClear (
    struct TextModeller_t * textModeller )
```

Clears GPU buffers for rendering 3D text.

7.1.1.788 TextModellerCopyToMeshBuffer()

```
AFTERWARP_API LibraryBool TextModellerCopyToMeshBuffer (
    struct TextModeller_t * textModeller,
    struct MeshBuffer_t * meshBuffer )
```

Copies the contents of existing 3D buffers into a mesh buffer.

7.1.1.789 TextModellerCreate()

```
AFTERWARP_API struct TextModeller_t * TextModellerCreate (
    struct Device_t * device )
```

Creates new instance of vector-based text rendering module.

7.1.1.790 TextModellerDestroy()

```
AFTERWARP_API void TextModellerDestroy (
    struct TextModeller_t * textModeller )
```

Releases an existing instance of vector-based text rendering module.

7.1.1.791 TextModellerDraw()

```
AFTERWARP_API LibraryBool TextModellerDraw (
    struct TextModeller_t * textModeller,
    Vector const * position,
    char const * text,
    float depth,
    ColorPair const * colors,
    TextAlignment alignHoriz,
    TextAlignment alignVert,
    TextAlignment alignDepth,
    LibraryBool alignByShape,
    TextRenderModifiers const * modifiers )
```

Renders a volumetric 3D text at the given position, orientation and alignment. The text is oriented in XZ plane with front side oriented towards positive Y axis.

7.1.1.792 TextModellerDrawCurved()

```
AFTERWARP_API LibraryBool TextModellerDrawCurved (
    struct TextModeller_t * textModeller,
    Vector const * position,
    char const * text,
    float radius,
    float angle,
    float depth,
    ColorPair const * colors,
    TextAlignment alignDepth,
    TextRenderModifiers const * modifiers )
```

Renders a curved volumetric 3D text at the given position, orientation and alignment. The text is oriented in XZ plane with front side oriented towards positive Y axis, and the rotation is performed around Y axis. Note: if angle is "NaN", then the text is centered vertically according to the curvature; that is, if the radius would converge to infinity, the text would converge to look like if it wasn't curved.

7.1.1.793 TextModellerDrawCurvedToCanvas()

```
AFTERWARP_API LibraryBool TextModellerDrawCurvedToCanvas (
    struct TextModeller_t * textModeller,
    struct Canvas_t * canvas,
    PointF const * position,
    char const * text,
    float radius,
    float angle,
    ColorPair const * colors,
    TextRenderModifiers const * modifiers,
    BlendingEffect effect )
```

Draws a curved 2D text at the given position with specific alignment directly to canvas. Note: if angle is "NaN", then the text is centered vertically according to the curvature; that is, if the radius would converge to infinity, the text would converge to look like if it wasn't curved.

7.1.1.794 TextModellerDrawCurvedToCanvasBuffer()

```
AFTERWARP_API LibraryBool TextModellerDrawCurvedToCanvasBuffer (
    struct TextModeller_t * textModeller,
    struct CanvasBuffer_t * buffer,
    PointF const * position,
    char const * text,
    float radius,
    float angle,
    ColorPair const * colors,
    TextRenderModifiers const * modifiers )
```

Draws a curved 2D text at the given position with specific alignment to a canvas buffer. Note: if angle is "NaN", then the text is centered vertically according to the curvature; that is, if the radius would converge to infinity, the text would converge to look like if it wasn't curved.

7.1.1.795 TextModellerDrawDepthCurved()

```
AFTERWARP_API LibraryBool TextModellerDrawDepthCurved (
    struct TextModeller_t * textModeller,
    Vector const * position,
    char const * text,
    float radius,
    float angle,
    float depth,
    ColorPair const * colors,
    TextRenderModifiers const * modifiers )
```

Renders a curved volumetric 3D text at the given position, orientation and alignment. The text is oriented in XZ plane with front side oriented towards positive Y axis, and the rotation is performed around Z axis. Note: if angle is "NaN", then the text is centered vertically according to the curvature; that is, if the radius would converge to infinity, the text would converge to look like if it wasn't curved.

7.1.1.796 TextModellerDrawToCanvas()

```
AFTERWARP_API LibraryBool TextModellerDrawToCanvas (
    struct TextModeller_t * textModeller,
    struct Canvas_t * canvas,
    PointF const * position,
    char const * text,
    ColorPair const * colors,
    TextAlignment alignHoriz,
    TextAlignment alignVert,
    LibraryBool alignByShape,
    TextRenderModifiers const * modifiers,
    BlendingEffect effect )
```

Draws 2D text at the given position with specific alignment directly to canvas.

7.1.1.797 TextModellerDrawToCanvasBuffer()

```
AFTERWARP_API LibraryBool TextModellerDrawToCanvasBuffer (
    struct TextModeller_t * textModeller,
    struct CanvasBuffer_t * buffer,
    PointF const * position,
    char const * text,
    ColorPair const * colors,
    TextAlignment alignHoriz,
    TextAlignment alignVert,
    LibraryBool alignByShape,
    TextRenderModifiers const * modifiers )
```

Draws 2D text at the given position with specific alignment to a canvas buffer.

7.1.1.798 TextModellerExtent()

```
AFTERWARP_API void TextModellerExtent (
    struct TextModeller_t * textModeller,
    char const * text,
    PointF * extent,
    TextRenderModifiers const * modifiers )
```

Returns the logical dimensions that the given text will occupy when rendered.

7.1.1.799 TextModellerExtentByShape()

```
AFTERWARP_API void TextModellerExtentByShape (
    struct TextModeller_t * textModeller,
    char const * text,
    RectF * rect,
    TextRenderModifiers const * modifiers )
```

Calculates a physical shape area that a given text occupies when rendered at zero position.

7.1.1.800 TextModellerGetDevice()

```
AFTERWARP_API struct Device_t * TextModellerGetDevice (
    struct TextModeller_t * textModeller )
```

Returns graphics device associated with the text rendering module.

7.1.1.801 TextModellerGetFontProvider()

```
AFTERWARP_API UntypedHandle TextModellerGetFontProvider (
    struct TextModeller_t * textModeller )
```

Returns currently associated font provider.

7.1.1.802 TextModellerGetTransform()

```
AFTERWARP_API void TextModellerGetTransform (
    struct TextModeller_t * textModeller,
    Matrix * transform )
```

Returns current 3D transformation matrix.

7.1.1.803 TextModellerPrepare()

```
AFTERWARP_API LibraryBool TextModellerPrepare (
    struct TextModeller_t * textModeller )
```

Prepares and fills GPU buffers with an existing 3D text rendering data.

7.1.1.804 TextModellerRects()

```
AFTERWARP_API int32_t TextModellerRects (
    struct TextModeller_t * textModeller,
    PointF * extent,
    _Out_opt_ TextEntryRect rects[],
    char const * text,
    TextRenderModifiers const * modifiers )
```

Provides information regarding individual character position and sizes for the given text string when rendered. This can be useful for components such as text edit box, for highlighting and selecting different characters. Returns the actual width and height of the text.

7.1.1.805 TextModellerRender()

```
AFTERWARP_API LibraryBool TextModellerRender (
    struct TextModeller_t * textModeller,
    struct Program_t * program,
    uint32_t channel )
```

Issues draw call with existing GPU buffers attached.

7.1.1.806 TextModellerReset()

```
AFTERWARP_API void TextModellerReset (
    struct TextModeller_t * textModeller )
```

Releases existing resources, character records and buffers.

7.1.1.807 TextModellerSetFontParameters()

```
AFTERWARP_API UntypedHandle TextModellerSetFontParameters (
    struct TextModeller_t * textModeller,
    FontParameters const * parameters )
```

Searches for an existing font provider with the given parameters and if such is not found, creates a new one. Returns the appropriate font provider or NULL if provider could not be created.

7.1.1.808 TextModellerSetFontProvider()

```
AFTERWARP_API void TextModellerSetFontProvider (
    struct TextModeller_t * textModeller,
    UntypedHandle provider )
```

Changes currently associated font provider. The parameter must match one of the results returned by `TextModellerSetFontParameters` function.

7.1.1.809 TextModellerSetTransform()

```
AFTERWARP_API void TextModellerSetTransform (
    struct TextModeller_t * textModeller,
    Matrix const * transform )
```

Changes current 3D transformation matrix.

7.1.1.810 TextRendererCreate()

```
AFTERWARP_API struct TextRenderer_t * TextRendererCreate (
    struct Canvas_t * canvas,
    Point const * textureSize,
    PixelFormat pixelFormat,
    LibraryBool mipMapping )
```

Creates new instance of text renderer.

7.1.1.811 TextRendererDestroy()

```
AFTERWARP_API void TextRendererDestroy (
    struct TextRenderer_t * textRenderer )
```

Releases given instance of text renderer.

7.1.1.812 TextRendererDraw()

```
AFTERWARP_API void TextRendererDraw (
    struct TextRenderer_t * textRenderer,
    PointF const * position,
    char const * text,
    ColorPair const * colors,
    float alpha,
    TextRenderModifiers const * modifiers )
```

Draws text at the given position.

7.1.1.813 TextRendererDrawAligned()

```
AFTERWARP_API void TextRendererDrawAligned (
    struct TextRenderer_t * textRenderer,
    PointF const * position,
    char const * text,
    ColorPair const * colors,
    TextAlignment horizAlign,
    TextAlignment vertAlign,
    float alpha,
    LibraryBool alignToPixels,
    TextRenderModifiers const * modifiers )
```

Draws text at the given position with specific alignment.

7.1.1.814 TextRendererDrawAlignedByPixels()

```
AFTERWARP_API void TextRendererDrawAlignedByPixels (
    struct TextRenderer_t * textRenderer,
    PointF const * position,
    char const * text,
    ColorPair const * colors,
    TextAlignment horizAlign,
    TextAlignment vertAlign,
    float alpha,
    LibraryBool alignToPixels,
    TextRenderModifiers const * modifiers )
```

Draws text at the given position with specific alignment by actual visible pixels.

7.1.1.815 TextRendererDrawCentered()

```
AFTERWARP_API void TextRendererDrawCentered (
    struct TextRenderer_t * textRenderer,
    PointF const * position,
    char const * text,
    ColorPair const * colors,
    float alpha,
    LibraryBool alignToPixels,
    TextRenderModifiers const * modifiers )
```

Draws text centered around the specified position.

7.1.1.816 TextRendererDrawCenteredByPixels()

```
AFTERWARP_API void TextRendererDrawCenteredByPixels (
    struct TextRenderer_t * textRenderer,
    PointF const * position,
    char const * text,
    ColorPair const * colors,
    float alpha,
    LibraryBool alignToPixels,
    TextRenderModifiers const * modifiers )
```

Draws text centered around the specified position by actual visible pixels.

7.1.1.817 TextRendererExtent()

```
AFTERWARP_API void TextRendererExtent (
    struct TextRenderer_t * textRenderer,
    char const * text,
    PointF * dimensions,
    TextRenderModifiers const * modifiers )
```

Returns the logical dimensions that the given text will occupy when rendered.

7.1.1.818 TextRendererExtentByPixels()

```
AFTERWARP_API void TextRendererExtentByPixels (
    struct TextRenderer_t * textRenderer,
    char const * text,
    RectF * rect,
    TextRenderModifiers const * modifiers )
```

Calculates the actual rectangle in pixels that the given text will occupy when rendered.

7.1.1.819 TextRendererGetCanvas()

```
AFTERWARP_API struct Canvas_t * TextRendererGetCanvas (
    struct TextRenderer_t * textRenderer )
```

Returns canvas class associated with the text renderer.

7.1.1.820 TextRendererGetFontParameters()

```
AFTERWARP_API LibraryBool TextRendererGetFontParameters (
    struct TextRenderer_t * textRenderer,
    FontParameters * parameters )
```

Retrieves current font settings from the text renderer.

7.1.1.821 TextRendererRects()

```
AFTERWARP_API int32_t TextRendererRects (
    struct TextRenderer_t * textRenderer,
    PointF * extent,
    _Out_opt_ TextEntryRect rects[],
    char const * text,
    TextRenderModifiers const * modifiers )
```

Provides information regarding individual character position and sizes for the given text string when rendered. This can be useful for components such as text edit box, for highlighting and selecting different characters. The actual width and height of the text is, optionally, provided as well. Returns actual number of elements returned. Calling this function with `rects` set to NULL would calculate how many elements need to be allocated.

7.1.1.822 TextRendererSetFontParameters()

```
AFTERWARP_API LibraryBool TextRendererSetFontParameters (
    struct TextRenderer_t * textRenderer,
    FontParameters const * parameters )
```

Specifies new font settings for the text renderer.

7.1.1.823 TextureAttach()

```
AFTERWARP_API LibraryBool TextureAttach (
    struct Texture_t * texture,
    struct Texture_t * attachment,
    int32_t layer,
    int32_t mipLevel )
```

Attaches another drawable texture with the current texture into a common MRT rendering stack. Drawable textures will be available to shaders in the same order of attachment.

7.1.1.824 TextureBegin()

```
AFTERWARP_API LibraryBool TextureBegin (
    struct Texture_t * texture,
    int32_t layer,
    int32_t mipLevel )
```

Starts rendering on the drawable texture.

7.1.1.825 TextureBind()

```
AFTERWARP_API LibraryBool TextureBind (
    struct Texture_t * texture,
    uint32_t channel )
```

Binds texture to the specified rendering channel.

7.1.1.826 TextureCabinetBegin()

```
AFTERWARP_API LibraryBool TextureCabinetBegin (
    struct TextureCabinet_t * textureCabinet,
    TextureCabinetPass pass )
```

Begins rendering the given pass.

7.1.1.827 TextureCabinetClear()

```
AFTERWARP_API LibraryBool TextureCabinetClear (
    struct TextureCabinet_t * textureCabinet,
    TextureCabinetPass pass,
    _In_opt_ FloatColor const * color )
```

Clears the content of textures related to the given pass. Note: the given color is used only in main/color pass.

7.1.1.828 TextureCabinetCreate()

```
AFTERWARP_API struct TextureCabinet_t * TextureCabinetCreate (
    struct Device_t * device,
    Point const * size,
    TextureFidelity fidelity,
    int32_t samples,
    PixelFormat depthStencil )
```

Creates a new instance of collection of drawable textures for performing 3D scene rendering.

7.1.1.829 TextureCabinetDestroy()

```
AFTERWARP_API void TextureCabinetDestroy (
    struct TextureCabinet_t * textureCabinet )
```

Releases given instance of drawable textures collection.

7.1.1.830 TextureCabinetEnd()

```
AFTERWARP_API void TextureCabinetEnd (
    struct TextureCabinet_t * textureCabinet )
```

Ends rendering the pass that was previously started.

7.1.1.831 TextureCabinetFilter()

```
AFTERWARP_API LibraryBool TextureCabinetFilter (
    struct TextureCabinet_t * textureCabinet,
    TextureCabinetFilterType filter,
    Matrix const * projection )
```

Performs processing and filtering of the textures according to the filter type. For occlusion, performs depth linearization and computes ambient occlusion. For bloom, downscales the color texture, applies blur to compute glare, combines the effects and then performs HDR tone-mapping. For order-independent transparency (OIT or "glassy") pass, performs compositioning of the textures. This step must be performed outside of `TextureCabinetBegin()` and `TextureCabinetEnd()` block after finishing each corresponding rendering pass but before starting next one. For performance reasons, the filtering should be performed as late as possible after finishing previous pass (e.g. after doing some CPU calculations) to enable GPU to do work in parallel. Note: OIT rendering pass does not require projection matrix.

7.1.1.832 TextureCabinetGetAmbientOcclusionParameters()

```
AFTERWARP_API void TextureCabinetGetAmbientOcclusionParameters (
    struct TextureCabinet_t * textureCabinet,
    AmbientOcclusionParameters * parameters )
```

Returns ambient occlusion parameters.

7.1.1.833 TextureCabinetGetAttributes()

```
AFTERWARP_API TextureCabinetAttributes TextureCabinetGetAttributes (
    struct TextureCabinet_t * textureCabinet )
```

Returns rendering attributes of the collection.

7.1.1.834 TextureCabinetGetDepthStencil()

```
AFTERWARP_API PixelFormat TextureCabinetGetDepthStencil (
    struct TextureCabinet_t * textureCabinet )
```

Returns format used for depth/stencil buffers.

7.1.1.835 TextureCabinetGetDevice()

```
AFTERWARP_API struct Device_t * TextureCabinetGetDevice (
    struct TextureCabinet_t * textureCabinet )
```

Returns device associated with the given textures collection.

7.1.1.836 TextureCabinetGetFidelity()

```
AFTERWARP_API TextureFidelity TextureCabinetGetFidelity (
    struct TextureCabinet_t * textureCabinet )
```

Returns level of fidelity used for choosing integrated pixel formats.

7.1.1.837 TextureCabinetGetFinalTexture()

```
AFTERWARP_API struct Texture_t * TextureCabinetGetFinalTexture (
    struct TextureCabinet_t * textureCabinet )
```

Returns final color texture that can contains the scene.

7.1.1.838 TextureCabinetGetSamples()

```
AFTERWARP_API int32_t TextureCabinetGetSamples (
    struct TextureCabinet_t * textureCabinet )
```

Returns number of samples used in the textures.

7.1.1.839 TextureCabinetGetSize()

```
AFTERWARP_API void TextureCabinetGetSize (
    struct TextureCabinet_t * textureCabinet,
    Point * size )
```

Returns the size of the integrated textures.

7.1.1.840 TextureCabinetGetTexture()

```
AFTERWARP_API struct Texture_t * TextureCabinetGetTexture (
    struct TextureCabinet_t * textureCabinet,
    TextureCabinetType type )
```

Returns texture from the container of the given type.

7.1.1.841 TextureCabinetGetToneMappingBloom()

```
AFTERWARP_API void TextureCabinetGetToneMappingBloom (
    struct TextureCabinet_t * textureCabinet,
    ToneMappingBloom * parameters )
```

Returns tone-mapping and bloom parameters.

7.1.1.842 TextureCabinetPresent()

```
AFTERWARP_API LibraryBool TextureCabinetPresent (
    struct TextureCabinet_t * textureCabinet )
```

Renders the final color texture on the current rendering surface.

7.1.1.843 TextureCabinetRendering()

```
AFTERWARP_API LibraryBool TextureCabinetRendering (
    struct TextureCabinet_t * textureCabinet )
```

Indicates that one of the passes is currently being rendered.

7.1.1.844 TextureCabinetResolve()

```
AFTERWARP_API LibraryBool TextureCabinetResolve (
    struct TextureCabinet_t * textureCabinet )
```

Renders a multisample HDR color texture into a final color texture.

7.1.1.845 TextureCabinetRetrieveGlassyMetrics()

```
AFTERWARP_API LibraryBool TextureCabinetRetrieveGlassyMetrics (
    struct TextureCabinet_t * textureCabinet,
    uint32_t * values )
```

Retrieves metrics when using glassy rendering; five values are returned: hash table size, fragments drawn, fragments written, fragments read, max depth. Note: this should be called as little often as possible (ideally, no more than a couple of times per second) to avoid unnecessary rendering pipeline stalls as the values are read asynchronously from GPU.

7.1.1.846 TextureCabinetSetAmbientOcclusionParameters()

```
AFTERWARP_API void TextureCabinetSetAmbientOcclusionParameters (
    struct TextureCabinet_t * textureCabinet,
    AmbientOcclusionParameters const * parameters )
```

Updates ambient occlusion parameters.

7.1.1.847 TextureCabinetSetAttributes()

```
AFTERWARP_API LibraryBool TextureCabinetSetAttributes (
    struct TextureCabinet_t * textureCabinet,
    TextureCabinetAttributes attributes )
```

7.1.1.848 TextureCabinetSetSize()

```
AFTERWARP_API LibraryBool TextureCabinetSetSize (
    struct TextureCabinet_t * textureCabinet,
    Point const * size )
```

Updates the size of integrated textures and buffers.

7.1.1.849 TextureCabinetSetToneMappingBloom()

```
AFTERWARP_API void TextureCabinetSetToneMappingBloom (
    struct TextureCabinet_t * textureCabinet,
    ToneMappingBloom const * parameters )
```

Updates tone-mapping and bloom parameters.

7.1.1.850 TextureClear()

```
AFTERWARP_API LibraryBool TextureClear (
    struct Texture_t * texture )
```

Clears entire texture surface and fills pixels with zeros.

7.1.1.851 TextureClearWith()

```
AFTERWARP_API LibraryBool TextureClearWith (
    struct Texture_t * texture,
    FloatColor const * color,
    int32_t layer,
    int32_t mipLevel )
```

Clears texture surface with the given color. This must be called outside of device and texture `TextureBegin()` / `TextureEnd()` blocks.

7.1.1.852 TextureCopy()

```
AFTERWARP_API LibraryBool TextureCopy (
    struct Texture_t * texture,
    struct Texture_t * source,
    int32_t destLayer,
    Point const * destPos,
    int32_t srcLayer,
    Rect const * sourceRect,
    int32_t destMipLevel,
    int32_t srcMipLevel )
```

Copies a portion of source texture to the current one according to specified source rectangle and destination position. If source rectangle is empty, then the entire source texture will be copied. This function does the appropriate clipping and pixel format conversion (albeit at significant performance cost). Note that `premultipliedAlpha` attribute of either current or source texture is completely ignored.

7.1.1.853 TextureCopyFromSurface()

```
AFTERWARP_API LibraryBool TextureCopyFromSurface (
    struct Texture_t * texture,
    struct Surface_t * surface,
    int32_t layer,
    Point const * destPos,
    Rect const * sourceRect,
    int32_t mipLevel )
```

Copies a portion of source surface to the current texture according to specified source rectangle and destination position. If source rectangle is empty, then the entire source surface will be copied. This function does the appropriate clipping and pixel format conversion. Note that `premultipliedAlpha` attribute of either current or source surface is completely ignored.

7.1.1.854 TextureCopyToSurface()

```
AFTERWARP_API LibraryBool TextureCopyToSurface (
    struct Texture_t * texture,
    struct Surface_t * surface,
    int32_t layer,
    Point const * destPos,
    Rect const * sourceRect,
    int32_t mipLevel )
```

Copies a portion of current texture to the destination surface according to specified source rectangle and destination position. If source rectangle is empty, then the entire texture surface will be copied. This function does the appropriate clipping and pixel format conversion. Note that `premultipliedAlpha` attribute of either current or source surface is completely ignored.

7.1.1.855 TextureCreate()

```
AFTERWARP_API struct Texture_t * TextureCreate (
    struct Device_t * device,
    TextureParameters const * parameters )
```

Creates a new instance of texture object.

7.1.1.856 TextureCreateFromFile()

```
AFTERWARP_API struct Texture_t * TextureCreateFromFile (
    struct Device_t * device,
    char const * fileName,
    PixelFormat format,
    TextureAttributes attributes )
```

Creates a new instance of texture object loading its contents from an external file. This function typically supports most common file formats such as BMP, PNG, JPEG, but may also support other formats like GIF and TIFF.

7.1.1.857 TextureCreateFromFileInMemory()

```
AFTERWARP_API struct Texture_t * TextureCreateFromFileInMemory (
    struct Device_t * device,
    void const * fileContents,
    uint32_t contentSize,
    char const * extension,
    PixelFormat format,
    TextureAttributes attributes )
```

Creates a new instance of texture object loading its contents from an image file that has been preloaded into memory. The internal file format is determined by "extension" parameter, which should contain a valid file extension such as `.png` (case-insensitive). This function typically supports most common file formats such as BMP, PNG, JPEG, but may also support other formats like GIF and TIFF.

7.1.1.858 TextureCreateFromFileNormalsAndOcclusion()

```
AFTERWARP_API struct Texture_t * TextureCreateFromFileNormalsAndOcclusion (
    struct Device_t * device,
    char const * fileNameNormalMap,
    char const * fileNameOcclusionMap,
    TextureAttributes attributes )
```

Creates texture containing normal map and occlusion map from external files on disk. Both normal and occlusion map must have equal sizes.

7.1.1.859 TextureCreateFromFileParallax()

```
AFTERWARP_API struct Texture_t * TextureCreateFromFileParallax (
    struct Device_t * device,
    char const * fileName,
    PixelFormat format,
    TextureAttributes attributes )
```

Creates texture containing displacement map for Parallax Mapping technique from an external file on disk.

7.1.1.860 TextureCreateFromSurface()

```
AFTERWARP_API struct Texture_t * TextureCreateFromSurface (
    struct Device_t * device,
    struct Surface_t * surface,
    TextureAttributes attributes )
```

Creates a new instance of texture object loading its contents from an existing surface.

7.1.1.861 TextureDestroy()

```
AFTERWARP_API void TextureDestroy (
    struct Texture_t * texture )
```

Releases given instance of texture object.

7.1.1.862 TextureDetach()

```
AFTERWARP_API void TextureDetach (
    struct Texture_t * texture )
```

Removes all previous drawable texture attachments.

7.1.1.863 TextureEnd()

```
AFTERWARP_API LibraryBool TextureEnd (
    struct Texture_t * texture )
```

Finishes rendering on the drawable texture.

7.1.1.864 TextureGenerateMipMaps()

```
AFTERWARP_API LibraryBool TextureGenerateMipMaps (
    struct Texture_t * texture )
```

Generates texture's mipmaps from its base (level = 0) image.

7.1.1.865 TextureGetDevice()

```
AFTERWARP_API struct Device_t * TextureGetDevice (
    struct Texture_t * texture )
```

Returns device associated with the given texture.

7.1.1.866 TextureGetParameters()

```
AFTERWARP_API void TextureGetParameters (
    struct Texture_t * texture,
    TextureParameters * parameters )
```

Retrieves current texture parameters.

7.1.1.867 TextureGetPlatformHandle()

```
AFTERWARP_API UntypedHandle TextureGetPlatformHandle (
    struct Texture_t * texture )
```

Returns platform-specific texture handle.

7.1.1.868 TextureLoadFromFile()

```
AFTERWARP_API LibraryBool TextureLoadFromFile (
    struct Texture_t * texture,
    char const * fileName,
    int32_t layer,
    Point const * destPos,
    Rect const * sourceRect,
    int32_t mipLevel )
```

Loads image from disk and copies it to the specified location and parameters.

7.1.1.869 TextureResetCache()

```
AFTERWARP_API void TextureResetCache (
    struct Texture_t * texture )
```

Resets any cache associated with the texture, which means purging any temporary resources. This also removes all existing texture attachments.

7.1.1.870 TextureRetrieve()

```
AFTERWARP_API LibraryBool TextureRetrieve (
    struct Texture_t * texture,
    void * content,
    uint32_t pitch,
    int32_t layer,
    Rect const * rect,
    int32_t mipLevel )
```

Retrieves content from the given location of texture's owned memory to the user pointer. The data is copied in its original format and source rectangle, if specified, must be within valid bounds (no clipping is performed).

7.1.1.871 TextureSaveToFile()

```
AFTERWARP_API LibraryBool TextureSaveToFile (
    struct Texture_t * texture,
    char const * fileName,
    int32_t quality,
    int32_t layer,
    Rect const * sourceRect,
    int32_t mipLevel )
```

Saves the appropriate portion of texture to external file.

7.1.1.872 TextureUnbind()

```
AFTERWARP_API LibraryBool TextureUnbind (
    struct Texture_t * texture,
    uint32_t channel )
```

Unbinds texture from the specified rendering channel.

7.1.1.873 TextureUpdate()

```
AFTERWARP_API LibraryBool TextureUpdate (
    struct Texture_t * texture,
    void const * content,
    uint32_t pitch,
    int32_t layer,
    Rect const * rect,
    int32_t mipLevel )
```

Copies content from user pointer to texture's owned memory at the given location. The data is assumed to match texture's pixel format and destination rectangle, if specified, must be within valid bounds (no clipping is performed).

7.1.1.874 TimerCreate()

```
AFTERWARP_API struct Timer_t * TimerCreate (
    void )
```

Creates new instance of timer that can be used in multimedia interactive applications to ensure fixed processing rate and/or variable processing rate (e.g. as fast as possible) with a fixed-rate counter.

7.1.1.875 TimerDestroy()

```
AFTERWARP_API void TimerDestroy (
    struct Timer\_t * timer )
```

Releases given timer instance.

7.1.1.876 TimerExtractTokens()

```
AFTERWARP_API int32_t TimerExtractTokens (
    struct Timer\_t * timer )
```

Extracts existing fixed-rate token counter from the timer.

7.1.1.877 TimerGetFrameRate()

```
AFTERWARP_API float TimerGetFrameRate (
    struct Timer\_t * timer )
```

Returns calculated average frame rate in frames per second. This value is updated roughly two times per second and can only be used for informative purposes (e.g. displaying frame rate in the application). For precise real-time indications it is recommended to use [TimerGetLatency\(\)](#).

7.1.1.878 TimerGetLatency()

```
AFTERWARP_API uint64_t TimerGetLatency (
    struct Timer\_t * timer )
```

Returns time (in microseconds) calculated between previous and current frames. This can be a direct indication of rendering performance as it tells how much time it took to render (and possibly, process) the frame.

7.1.1.879 TimerGetSkippedTimeSlices()

```
AFTERWARP_API int64_t TimerGetSkippedTimeSlices (
    struct Timer\_t * timer )
```

Returns total number of skipped time slices.

7.1.1.880 TimerGetSpeed()

```
AFTERWARP_API double TimerGetSpeed (
    struct Timer\_t * timer )
```

Returns currently set multimedia timer speed (in terms of frames per second).

7.1.1.881 TimerGetTimeSlice()

```
AFTERWARP_API int64_t TimerGetTimeSlice (
    struct Timer_t * timer )
```

Returns current time slice.

7.1.1.882 TimerNextSlice()

```
AFTERWARP_API LibraryBool TimerNextSlice (
    struct Timer_t * timer )
```

Ensures that next processing time slice is reached. Returns whether the method was executed within the correct time slice (in other words, no timer slices have been lost, which could result in stuttering).

7.1.1.883 TimerReset()

```
AFTERWARP_API void TimerReset (
    struct Timer_t * timer )
```

Resets internal structures of the timer, restarts the timing calculations and sets current time slice to zero. This can be useful when an unscheduled and/or a time-consuming task was executed.

7.1.1.884 TimerSetSpeed()

```
AFTERWARP_API void TimerSetSpeed (
    struct Timer_t * timer,
    double speed )
```

Sets new multimedia timer processing speed (in terms of frames per second). Default speed is 60 frames per second.

7.1.1.885 TimerTrimSkippedTimeSlices()

```
AFTERWARP_API void TimerTrimSkippedTimeSlices (
    struct Timer_t * timer,
    int64_t slicesToTrim )
```

Reduces number of skipped time slices by the given quantity. If the quantity is larger than the actual number of skipped time slices, resets that number to zero.

7.1.1.886 TimerUpdate()

```
AFTERWARP_API void TimerUpdate (
    struct Timer_t * timer )
```

Updates latency, calculates average frame rate and updates fixed-rate token counter.

7.1.1.887 TimerUpdateNextSlice()

```
AFTERWARP_API LibraryBool TimerUpdateNextSlice (
    struct Timer_t * timer )
```

Ensures that next processing time slice is reached, updates latency and calculates average frame rate. Returns whether the method was executed within the correct time slice (in other words, no timer slices have been lost, which could result in stuttering). Updates fixed-rate token counter.

7.1.1.888 UnpremultiplyAlpha()

```
AFTERWARP_API Color UnpremultiplyAlpha (
    Color color )
```

Takes 32-bit RGBA color with premultiplied alpha channel and divides each of its red, green, and blue components by alpha, resulting in unpremultiplied alpha color.

7.1.1.889 VertexElementsEstimatePitch()

```
AFTERWARP_API uint32_t VertexElementsEstimatePitch (
    uint32_t channel,
    VertexElement const vertexElements[],
    int32_t vertexElementCount )
```

Makes an estimation of pitch for the given channel based on existing element offsets.

7.1.1.890 VolumeCalculateNearFarPlanes()

```
AFTERWARP_API void VolumeCalculateNearFarPlanes (
    Matrix const * worldView,
    float * nearPlane,
    float * farPlane )
```

Calculates near and far planes for a bounding box transformed by the given world/view matrix.

7.1.1.891 VolumeCalculateVisibleFrame()

```
AFTERWARP_API void VolumeCalculateVisibleFrame (
    Matrix const * worldViewProjection,
    PointF const * surfaceSize,
    RectF * visibleFrame )
```

Calculates visible surface frame for a bounding box transformed by the given world/view/projection matrix.

7.1.1.892 WidgetAcceptKey()

```
AFTERWARP_API void WidgetAcceptKey (
    struct Widget_t * widget,
    KeyEvent event,
    Key key,
    UnicodeChar charCode )
```

Makes the widget receive and process keyboard input.

7.1.1.893 WidgetAcceptMouse()

```
AFTERWARP_API void WidgetAcceptMouse (
    struct Widget_t * widget,
    MouseEvent event,
    MouseButton button,
    PointF const * position )
```

Makes the widget receive and process mouse input.

7.1.1.894 WidgetAccomodate()

```
AFTERWARP_API void WidgetAccomodate (
    struct Widget_t * widget )
```

Schedules widget repositioning during next update cycle.

7.1.1.895 WidgetBringToFront()

```
AFTERWARP_API void WidgetBringToFront (
    struct Widget_t * widget )
```

Puts the widget on front of other widgets.

7.1.1.896 WidgetCreate()

```
AFTERWARP_API struct Widget_t * WidgetCreate (
    struct Widget_t * manager,
    char const * className,
    char const * instanceName,
    ObjectPayload payload )
```

Creates a new widget of the given class type (case-insensitive), assigns it requested instance name (case-insensitive) and optional payload. The widget is commonly accessed by its instance name, which must be unique. Possible values for "className" are: Widget, Label, Button, Edit, RadioButton, CheckBox, GaugeEdit, SpinEdit, Placer, Panel, TabSheets, Window and SketchPad.

7.1.1.897 WidgetDestroy()

```
AFTERWARP_API void WidgetDestroy (
    struct Widget_t * widget )
```

Releases an existing widget instance.

7.1.1.898 WidgetFindAt()

```
AFTERWARP_API struct Widget_t * WidgetFindAt (
    struct Widget_t * widget,
    PointF const * position )
```

Searches through children hierarchy and returns widget that is pointed by the given local position. If no child is found, the same widget is returned. This is particularly useful for mouse hit test.

7.1.1.899 WidgetGetChildByIndex()

```
AFTERWARP_API struct Widget_t * WidgetGetChildByIndex (
    struct Widget_t * widget,
    int32_t index )
```

Returns an existing widget's child with the given index.

7.1.1.900 WidgetGetChildCount()

```
AFTERWARP_API int32_t WidgetGetChildCount (
    struct Widget_t * widget )
```

Returns number of existing widget's children.

7.1.1.901 WidgetGetChildIndex()

```
AFTERWARP_API int32_t WidgetGetChildIndex (
    struct Widget_t * widget,
    struct Widget_t * child )
```

Attempts to find a widget among given parent's widget children and returns its index. If the given child is not found among parent's widget children, returns -1.

7.1.1.902 WidgetGetExternalEvent()

```
AFTERWARP_API void WidgetGetExternalEvent (
    struct Widget_t * widget,
    WidgetExternalEvent * event,
    void ** user )
```

Retrieves currently set external event callback.

7.1.1.903 WidgetGetManager()

```
AFTERWARP_API struct Widget_t * WidgetGetManager (
    struct Widget_t * widget )
```

Returns manager associated with the widget.

7.1.1.904 WidgetGetParent()

```
AFTERWARP_API struct Widget_t * WidgetGetParent (
    struct Widget_t * widget )
```

Returns widget's parent.

7.1.1.905 WidgetGetPayload()

```
AFTERWARP_API ObjectPayload WidgetGetPayload (
    struct Widget_t * widget )
```

Returns widget's payload.

7.1.1.906 WidgetGetProperty()

```
AFTERWARP_API LibraryBool WidgetGetProperty (
    struct Widget_t * widget,
    char const * propertyName,
    WidgetPropertyType valueType,
    void * value )
```

Retrieves an existing property and performs type conversion (if possible). This function will not work for string type of data, for which `WidgetGetPropertyAsString` should be used instead.

7.1.1.907 WidgetGetPropertyAsString()

```
AFTERWARP_API LibraryBool WidgetGetPropertyAsString (
    struct Widget_t * widget,
    char const * propertyName,
    char * value,
    _Inout_ int32_t * valueLength )
```

Returns an existing property as string. The pointed value of `valueLength` defines the maximum number of characters, including null-terminating character, that can be copied. The actual string length is stored in `valueLength`.

7.1.1.908 WidgetGetPropertyCount()

```
AFTERWARP_API int32_t WidgetGetPropertyCount (
    struct Widget_t * widget )
```

Returns number of existing properties.

7.1.1.909 WidgetInvalidate()

```
AFTERWARP_API void WidgetInvalidate (
    struct Widget_t * widget )
```

Schedules widget repaint during next update cycle.

7.1.1.910 WidgetInvokeEvent()

```
AFTERWARP_API LibraryBool WidgetInvokeEvent (
    struct Widget_t * widget,
    char const * eventName )
```

Invokes a widget's event.

7.1.1.911 WidgetLocalToScreen()

```
AFTERWARP_API void WidgetLocalToScreen (
    struct Widget_t * widget,
    PointF * screenPos,
    PointF const * localPos )
```

Converts local widget position to screen coordinates.

7.1.1.912 WidgetManagerBatchCount()

```
AFTERWARP_API int32_t WidgetManagerBatchCount (
    struct Widget_t * widget )
```

Returns total number of canvas batches during UI rendering.

7.1.1.913 WidgetManagerCreate()

```
AFTERWARP_API struct Widget_t * WidgetManagerCreate (
    struct Application_t * application,
    struct TextRenderer_t * textRenderer,
    char const * instanceName,
    ObjectPayload payload,
    PixelFormat format,
    int32_t multisamples,
    uint32_t attributes )
```

Creates a new widget manager, which facilitates working with and rendering widgets. The widget manager should be the topmost parent for all other widgets. The manager itself is also considered a widget.

7.1.1.914 WidgetManagerGetApplication()

```
AFTERWARP_API struct Application_t * WidgetManagerGetApplication (
    struct Widget_t * widget )
```

Returns application associated with the widget manager.

7.1.1.915 WidgetManagerGetAttributes()

```
AFTERWARP_API uint32_t WidgetManagerGetAttributes (
    struct Widget_t * widget )
```

Returns attributes used when the manager was created.

7.1.1.916 WidgetManagerGetFormat()

```
AFTERWARP_API PixelFormat WidgetManagerGetFormat (
    struct Widget_t * widget )
```

Returns pixel format of composition textures.

7.1.1.917 WidgetManagerGetMultisamples()

```
AFTERWARP_API int32_t WidgetManagerGetMultisamples (
    struct Widget_t * widget )
```

Returns multisamples of rendering composition texture.

7.1.1.918 WidgetManagerGetTextRenderer()

```
AFTERWARP_API struct TextRenderer_t * WidgetManagerGetTextRenderer (
    struct Widget_t * widget )
```

Returns application associated with the widget manager.

7.1.1.919 WidgetManagerGetTexture()

```
AFTERWARP_API struct Texture_t * WidgetManagerGetTexture (
    struct Widget_t * widget,
    WidgetManagerTextureType textureType )
```

Returns one of textures used for the composition.

7.1.1.920 WidgetManagerGetWidgetByName()

```
AFTERWARP_API struct Widget_t * WidgetManagerGetWidgetByName (
    struct Widget_t * widget,
    char const * name )
```

Returns an existing widget with the given name (case-insensitive).

7.1.1.921 WidgetManagerGetWidgetByPayload()

```
AFTERWARP_API struct Widget_t * WidgetManagerGetWidgetByPayload (
    struct Widget_t * widget,
    ObjectPayload payload )
```

Returns an existing widget with the given payload.

7.1.1.922 WidgetManagerPresent()

```
AFTERWARP_API LibraryBool WidgetManagerPresent (
    struct Widget_t * widget )
```

Renders widget manager's area and all controls within.

7.1.1.923 WidgetPropertyIdentify()

```
AFTERWARP_API LibraryBool WidgetPropertyIdentify (
    struct Widget_t * widget,
    _Inout_ WidgetProperty * property )
```

Returns name, type, behavior and possibly a value of an existing property. If property name is provided, the property is looked by its name. If location is provided, information about the property at that location is retrieved and location is updated to point to next entry from the list of existing properties. If the property name and location are not provided, then a first property is retrieved and location is updated to point to the next one. When the last property is retrieved from the list, the returned location will be set to "undefined". The contents of the property are also retrieved, unless it is a string that is long enough not to fit within 182 characters, in which case string length will be set to 255 with remaining data bytes set to zero; in this case, the string has to be retrieved using WidgetGetPropertyAsString function.

7.1.1.924 WidgetScreenToLocal()

```
AFTERWARP_API void WidgetScreenToLocal (
    struct Widget_t * widget,
    PointF * localPos,
    PointF const * screenPos )
```

Converts screen coordinates to local widget position.

7.1.1.925 WidgetSendToBack()

```
AFTERWARP_API void WidgetSendToBack (
    struct Widget_t * widget )
```

Puts the widget behind all other widgets.

7.1.1.926 WidgetSetExternalEvent()

```
AFTERWARP_API void WidgetSetExternalEvent (
    struct Widget_t * widget,
    WidgetExternalEvent event,
    void * user )
```

Specifies a new external event callback.

7.1.1.927 WidgetSetParent()

```
AFTERWARP_API LibraryBool WidgetSetParent (
    struct Widget_t * widget,
    struct Widget_t * parent )
```

Changes widget's parent. When widget's parent is assigned, the parent is responsible for a lifetime of the widget; that is, when a widget is released, all of its child hierarchy is released as well.

7.1.1.928 WidgetSetProperty()

```
AFTERWARP_API LibraryBool WidgetSetProperty (
    struct Widget_t * widget,
    char const * propertyName,
    WidgetPropertyType valueType,
    void const * value,
    LibraryBool invokeChanged )
```

Creates a new property value or updates an existing property value. If `value` is `NULL`, an existing property will be deleted.

7.1.1.929 WidgetUpdate()

```
AFTERWARP_API LibraryBool WidgetUpdate (
    struct Widget_t * widget,
    double latency )
```

Updates widget and its children. `latency` must be a number of milliseconds elapsed since previous call to update; this is necessary to ensure proper animations in the widgets. Returns `true` if the application needs to be repainted or `false` otherwise.

7.2 Afterwarp.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Afterwarp Framework (https://afterwarp.io).
00003  * Copyright (c) 2015 - 2025 Dr. Yuriy Kotsarenko. All rights reserved.
00004  *
00005  * Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in
00006  * compliance with the License. You may obtain a copy of the License at
00007  * http://www.apache.org/licenses/LICENSE-2.0
00008  *
00009  * Unless required by applicable law or agreed to in writing, software distributed under the License
00010  * is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
00011  * implied.
00012  * See the License for the specific language governing permissions and limitations under the License.
00013  */
00014 // Afterwarp.h
00015 #pragma once
00016
00017 #include "Afterwarp.Structs.h"
00018
00019 #ifdef __cplusplus
00020     extern "C" {
00021 #endif
00022
00023 // The following SAL-style annotations are used by an automated header translation tool to determine
00024 // proper
00025 // conventions for languages such as C#. For C/C++ applications, these are just a syntactic sugar and
00026 // should
00027 // be ignored.
00028 #if !defined(_MSC_VER) || !defined(_In_)
00029     #define _In_
00030     #define _Out_
00031     #define _Inout_
00032     #define _In_opt_
00033     #define _Out_opt_
00034     #define _Inout_opt_
00035 #endif
00036
00037 // Library functions.
00038 AFTERWARP_API LibraryBool LibraryGetVersion(int32_t* major, int32_t* minor, uint32_t* build);
00039
00040 AFTERWARP_API void LibrarySerialCode(char const* serial, int32_t length);
```



```

00042
00043 // SwapChain functions.
00044
00048 AFTERWARP_API struct SwapChain_t* SwapChainCreate(struct Device_t* device, UntypedHandle windowHandle,
00049     Point const* size, PixelFormat format, PixelFormat depthStencil, int32_t multisamples,
00050     LibraryBool vsync);
00051
00053 AFTERWARP_API void SwapChainDestroy(struct SwapChain_t* swapChain);
00054
00056 AFTERWARP_API struct Device_t* SwapChainGetDevice(struct SwapChain_t* swapChain);
00057
00059 AFTERWARP_API UntypedHandle SwapChainGetWindowHandle(struct SwapChain_t* swapChain);
00060
00062 AFTERWARP_API int32_t SwapChainGetWidth(struct SwapChain_t* swapChain);
00063
00065 AFTERWARP_API int32_t SwapChainGetHeight(struct SwapChain_t* swapChain);
00066
00068 AFTERWARP_API PixelFormat SwapChainGetFormat(struct SwapChain_t* swapChain);
00069
00071 AFTERWARP_API PixelFormat SwapChainGetDepthStencil(struct SwapChain_t* swapChain);
00072
00074 AFTERWARP_API int32_t SwapChainGetMultisamples(struct SwapChain_t* swapChain);
00075
00077 AFTERWARP_API LibraryBool SwapChainGetVSync(struct SwapChain_t* swapChain);
00078
00080 AFTERWARP_API LibraryBool SwapChainResize(struct SwapChain_t* swapChain, Point const* size);
00081
00083 AFTERWARP_API LibraryBool SwapChainBegin(struct SwapChain_t* swapChain);
00084
00086 AFTERWARP_API void SwapChainEnd(struct SwapChain_t* swapChain);
00087
00088 // Device functions.
00089
00091 AFTERWARP_API struct Device_t* DeviceCreate(struct Application_t* application, DeviceTechnology
    technology,
00092     DeviceAttributes attributes);
00093
00096 AFTERWARP_API struct Device_t* DeviceCreateWrapped(struct Application_t* application,
00097     DeviceTechnology technology, DeviceAttributes attributes, UntypedHandle platformDevice);
00098
00100 AFTERWARP_API void DeviceDestroy(struct Device_t* device);
00101
00103 AFTERWARP_API UntypedHandle DeviceGetPlatformDevice(struct Device_t* device);
00104
00106 AFTERWARP_API DeviceAttributes DeviceGetAttributes(struct Device_t* device);
00107
00109 AFTERWARP_API DeviceBehavior DeviceGetBehavior(struct Device_t* device);
00110
00112 AFTERWARP_API DeviceLegacyBits DeviceGetLegacyBits(struct Device_t* device);
00113
00115 AFTERWARP_API DeviceTechnology DeviceGetTechnology(struct Device_t* device);
00116
00118 AFTERWARP_API DevicePlatform DeviceGetPlatform(struct Device_t* device);
00119
00124 AFTERWARP_API uint32_t DeviceGetTechVersion(struct Device_t* device);
00125
00131 AFTERWARP_API uint32_t DeviceGetTechFeatureVersion(struct Device_t* device);
00132
00134 AFTERWARP_API void DeviceResetCache(struct Device_t* device);
00135
00137 AFTERWARP_API LibraryBool DeviceClear(struct Device_t* device, uint8_t clearLayers,
00138     FloatColor const* clearColor, float clearDepth, uint32_t clearStencil);
00139
00141 AFTERWARP_API LibraryBool DeviceSetRenderingState(struct Device_t* device, RenderingState const*
    state);
00142
00144 AFTERWARP_API void DeviceGetRenderingState(struct Device_t* device, RenderingState* state);
00145
00147 AFTERWARP_API LibraryBool DeviceSetViewport(struct Device_t* device, Rect const* viewport);
00148
00150 AFTERWARP_API void DeviceGetViewport(struct Device_t* device, Rect* viewport);
00151
00153 AFTERWARP_API LibraryBool DeviceSetScissor(struct Device_t* device, Rect const* scissor);
00154
00156 AFTERWARP_API void DeviceGetScissor(struct Device_t* device, Rect* scissor);
00157
00159 AFTERWARP_API void DeviceMemoryBarrier(struct Device_t* device, uint32_t barrierBits);
00160
00161 // Buffer functions.
00162
00165 AFTERWARP_API struct Buffer_t* BufferCreate(struct Device_t* device, BufferDataType dataType,
00166     BufferAccessType accessType, uint32_t size, uint32_t pitch, PixelFormat format, void const*
    initialData);
00167
00169 AFTERWARP_API void BufferDestroy(struct Buffer_t* buffer);
00170
00172 AFTERWARP_API struct Device_t* BufferGetDevice(struct Buffer_t* buffer);

```

```

00173
00175 AFTERWARP_API UntypedHandle BufferGetPlatformHandle(struct Buffer_t* buffer);
00176
00178 AFTERWARP_API BufferDataType BufferGetDataType(struct Buffer_t* buffer);
00179
00181 AFTERWARP_API BufferAccessType BufferGetAccessType(struct Buffer_t* buffer);
00182
00184 AFTERWARP_API uint32_t BufferGetSize(struct Buffer_t* buffer);
00185
00187 AFTERWARP_API uint32_t BufferGetPitch(struct Buffer_t* buffer);
00188
00190 AFTERWARP_API PixelFormat BufferGetFormat(struct Buffer_t* buffer);
00191
00196 AFTERWARP_API LibraryBool BufferUpdate(struct Buffer_t* buffer, void const* data, uint32_t offset,
00197     uint32_t size);
00198
00201 AFTERWARP_API LibraryBool BufferRetrieve(struct Buffer_t* buffer, void* data, uint32_t offset,
00202     uint32_t size);
00203
00206 AFTERWARP_API LibraryBool BufferCopy(struct Buffer_t* buffer, struct Buffer_t* bufferSource,
00207     uint32_t offsetDest, uint32_t offsetSource, uint32_t size);
00208
00209 // Program types and functions.
00210
00212 AFTERWARP_API struct Program_t* ProgramCreate(struct Device_t* device,
00213     _In_opt_ VertexElement const vertexElements[], int32_t vertexElementCount,
00214     _In_opt_ ProgramElement const programElements[], int32_t programElementCount,
00215     _In_opt_ ProgramVariable const programVariables[], int32_t programVariableCount,
00216     _In_opt_ char const* shaderVertex, int32_t shaderVertexLength,
00217     _In_opt_ char const* shaderHull, int32_t shaderHullLength,
00218     _In_opt_ char const* shaderDomain, int32_t shaderDomainLength,
00219     _In_opt_ char const* shaderGeometry, int32_t shaderGeometryLength,
00220     _In_opt_ char const* shaderPixel, int32_t shaderPixelLength);
00221
00223 AFTERWARP_API void ProgramDestroy(struct Program_t* program);
00224
00225 // TODO: expose getters for shaders, vertex and program elements.
00226
00228 AFTERWARP_API struct Device_t* ProgramGetDevice(struct Program_t* program);
00229
00234 AFTERWARP_API LibraryBool ProgramUpdateByIndex(struct Program_t* program, int32_t variableIndex,
00235     void const* variableData, uint32_t size);
00236
00241 AFTERWARP_API LibraryBool ProgramUpdateByName(struct Program_t* program, char const* variableName,
00242     void const* variableData, uint32_t size);
00243
00249 AFTERWARP_API LibraryBool ProgramBind(struct Program_t* program, struct Buffer_t* buffer, uint32_t
00250     channel,
00251     uint32_t offset);
00253 AFTERWARP_API void ProgramUnbind(struct Program_t* program, struct Buffer_t* buffer, uint32_t
00254     channel);
00256 AFTERWARP_API void ProgramResetBindings(struct Program_t* program);
00257
00260 AFTERWARP_API void ProgramResetCache(struct Program_t* program);
00261
00264 AFTERWARP_API LibraryBool ProgramCommit(struct Program_t* program);
00265
00267 AFTERWARP_API LibraryBool ProgramBegin(struct Program_t* program);
00268
00270 AFTERWARP_API void ProgramEnd(struct Program_t* program);
00271
00273 AFTERWARP_API LibraryBool ProgramDraw(struct Program_t* program, PrimitiveTopology topology,
00274     int32_t vertexCount, int32_t baseVertex);
00275
00277 AFTERWARP_API LibraryBool ProgramDrawIndexed(struct Program_t* program, PrimitiveTopology topology,
00278     int32_t indexCount, int32_t firstIndex, int32_t baseVertex);
00279
00281 AFTERWARP_API LibraryBool ProgramDrawInstances(struct Program_t* program, PrimitiveTopology topology,
00282     int32_t vertexCount, int32_t instanceCount, int32_t baseVertex);
00283
00285 AFTERWARP_API LibraryBool ProgramDrawInstancesIndexed(struct Program_t* program, PrimitiveTopology
00286     topology,
00287     int32_t indexCount, int32_t instanceCount, int32_t firstIndex, int32_t baseVertex);
00289 AFTERWARP_API LibraryBool ProgramSetPatchVertices(struct Program_t* program, int32_t patchVertices);
00290
00291 // ComputeProgram types and functions.
00292
00293 // TODO: expose getter for program elements.
00294
00296 AFTERWARP_API struct ComputeProgram_t* ComputeProgramCreate(struct Device_t* device,
00297     _In_opt_ ProgramElement const programElements[], int32_t programElementCount,
00298     _In_opt_ char const* shader, int32_t shaderLength);
00299
00301 AFTERWARP_API void ComputeProgramDestroy(struct ComputeProgram_t* program);

```

```

00302
00304 AFTERWARP_API struct Device_t* ComputeProgramGetDevice(struct ComputeProgram_t* program);
00305
00309 AFTERWARP_API LibraryBool ComputeProgramBindBuffer(struct ComputeProgram_t* program,
00310     struct Buffer_t* buffer, uint32_t channel, uint32_t offset);
00311
00314 AFTERWARP_API void ComputeProgramUnbindBuffer(struct ComputeProgram_t* program, struct Buffer_t*
    buffer,
00315     uint32_t channel);
00316
00318 AFTERWARP_API LibraryBool ComputeProgramBindTexture(struct ComputeProgram_t* program,
00319     struct Texture_t* texture, ComputeBindTextureFormat const* bindFormat);
00320
00322 AFTERWARP_API void ComputeProgramUnbindTexture(struct ComputeProgram_t* program, struct Texture_t*
    texture,
00323     ComputeBindTextureFormat const* bindFormat);
00324
00327 AFTERWARP_API void ComputeProgramResetBindings(struct ComputeProgram_t* program);
00328
00331 AFTERWARP_API LibraryBool ComputeProgramCommit(struct ComputeProgram_t* program);
00332
00334 AFTERWARP_API LibraryBool ComputeProgramBegin(struct ComputeProgram_t* program);
00335
00337 AFTERWARP_API void ComputeProgramEnd(struct ComputeProgram_t* program);
00338
00340 AFTERWARP_API LibraryBool ComputeProgramDispatch(struct ComputeProgram_t* program, int32_t groupsX,
00341     int32_t groupsY, int32_t groupsZ);
00342
00343 // Sampler functions.
00344
00346 AFTERWARP_API struct Sampler_t* SamplerCreate(struct Device_t* device, SamplerState const*
    samplerState);
00347
00349 AFTERWARP_API void SamplerDestroy(struct Sampler_t* sampler);
00350
00352 AFTERWARP_API struct Device_t* SamplerGetDevice(struct Sampler_t* sampler);
00353
00355 AFTERWARP_API void SamplerGetState(struct Sampler_t* sampler, SamplerState* samplerState);
00356
00358 AFTERWARP_API LibraryBool SamplerSetState(struct Sampler_t* sampler, SamplerState const*
    samplerState);
00359
00361 AFTERWARP_API LibraryBool SamplerBind(struct Sampler_t* sampler, uint32_t channel);
00362
00364 AFTERWARP_API void SamplerUnbind(struct Sampler_t* sampler, uint32_t channel);
00365
00366 // Texture functions.
00367
00369 AFTERWARP_API struct Texture_t* TextureCreate(struct Device_t* device, TextureParameters const*
    parameters);
00370
00374 AFTERWARP_API struct Texture_t* TextureCreateFromFile(struct Device_t* device, char const* fileName,
00375     PixelFormat format, TextureAttributes attributes);
00376
00378 AFTERWARP_API struct Texture_t* TextureCreateFromFileParallax(struct Device_t* device, char const*
    fileName,
00379     PixelFormat format, TextureAttributes attributes);
00380
00383 AFTERWARP_API struct Texture_t* TextureCreateFromFileNormalsAndOcclusion(struct Device_t* device,
00384     char const* fileNameNormalMap, char const* fileNameOcclusionMap, TextureAttributes attributes);
00385
00390 AFTERWARP_API struct Texture_t* TextureCreateFromFileInMemory(struct Device_t* device,
00391     void const* fileContents, uint32_t contentSize, char const* extension, PixelFormat format,
00392     TextureAttributes attributes);
00393
00395 AFTERWARP_API struct Texture_t* TextureCreateFromSurface(struct Device_t* device, struct Surface_t*
    surface,
00396     TextureAttributes attributes);
00397
00399 AFTERWARP_API void TextureDestroy(struct Texture_t* texture);
00400
00402 AFTERWARP_API struct Device_t* TextureGetDevice(struct Texture_t* texture);
00403
00405 AFTERWARP_API UntypedHandle TextureGetPlatformHandle(struct Texture_t* texture);
00406
00408 AFTERWARP_API void TextureGetParameters(struct Texture_t* texture, TextureParameters* parameters);
00409
00413 AFTERWARP_API LibraryBool TextureUpdate(struct Texture_t* texture, void const* content, uint32_t
    pitch,
00414     int32_t layer, Rect const* rect, int32_t mipLevel);
00415
00419 AFTERWARP_API LibraryBool TextureRetrieve(struct Texture_t* texture, void* content, uint32_t pitch,
00420     int32_t layer, Rect const* rect, int32_t mipLevel);
00421
00427 AFTERWARP_API LibraryBool TextureCopy(struct Texture_t* texture, struct Texture_t* source,
00428     int32_t destLayer, Point const* destPos, int32_t srcLayer, Rect const* sourceRect, int32_t
    destMipLevel,

```

```

00429     int32_t srcMipLevel);
00430
00435 AFTERWARP_API LibraryBool TextureCopyFromSurface(struct Texture_t* texture, struct Surface_t* surface,
00436     int32_t layer, Point const* destPos, Rect const* sourceRect, int32_t mipLevel);
00437
00442 AFTERWARP_API LibraryBool TextureCopyToSurface(struct Texture_t* texture, struct Surface_t* surface,
00443     int32_t layer, Point const* destPos, Rect const* sourceRect, int32_t mipLevel);
00444
00446 AFTERWARP_API LibraryBool TextureLoadFromFile(struct Texture_t* texture, char const* fileName,
00447     int32_t layer, Point const* destPos, Rect const* sourceRect, int32_t mipLevel);
00448
00450 AFTERWARP_API LibraryBool TextureSaveToFile(struct Texture_t* texture, char const* fileName,
00451     int32_t quality, int32_t layer, Rect const* sourceRect, int32_t mipLevel);
00452
00454 AFTERWARP_API LibraryBool TextureClear(struct Texture_t* texture);
00455
00458 AFTERWARP_API LibraryBool TextureClearWith(struct Texture_t* texture, FloatColor const* color,
00459     int32_t layer, int32_t mipLevel);
00460
00462 AFTERWARP_API LibraryBool TextureBind(struct Texture_t* texture, uint32_t channel);
00463
00465 AFTERWARP_API LibraryBool TextureUnbind(struct Texture_t* texture, uint32_t channel);
00466
00469 AFTERWARP_API LibraryBool TextureAttach(struct Texture_t* texture, struct Texture_t* attachment,
00470     int32_t layer, int32_t mipLevel);
00471
00473 AFTERWARP_API void TextureDetach(struct Texture_t* texture);
00474
00476 AFTERWARP_API LibraryBool TextureBegin(struct Texture_t* texture, int32_t layer, int32_t mipLevel);
00477
00479 AFTERWARP_API LibraryBool TextureEnd(struct Texture_t* texture);
00480
00482 AFTERWARP_API LibraryBool TextureGenerateMipMaps(struct Texture_t* texture);
00483
00486 AFTERWARP_API void TextureResetCache(struct Texture_t* texture);
00487
00488 // Surface functions.
00489
00491 AFTERWARP_API struct Surface_t* SurfaceCreate(int32_t width, int32_t height, PixelFormat format);
00492
00497 AFTERWARP_API struct Surface_t* SurfaceCreateFromFile(char const* fileName,
00498     AlphaFormatRequest formatRequest);
00499
00506 AFTERWARP_API struct Surface_t* SurfaceCreateFromFileInMemory(void const* fileContents,
00507     uint32_t contentSize, char const* extension, AlphaFormatRequest formatRequest);
00508
00510 AFTERWARP_API void SurfaceDestroy(struct Surface_t* surface);
00511
00514 AFTERWARP_API void* SurfaceGetBits(struct Surface_t* surface);
00515
00519 AFTERWARP_API uint32_t SurfaceGetPitch(struct Surface_t* surface);
00520
00522 AFTERWARP_API int32_t SurfaceGetBytesPerPixel(struct Surface_t* surface);
00523
00525 AFTERWARP_API int32_t SurfaceGetWidth(struct Surface_t* surface);
00526
00528 AFTERWARP_API int32_t SurfaceGetHeight(struct Surface_t* surface);
00529
00531 AFTERWARP_API uint32_t SurfaceGetByteSize(struct Surface_t* surface);
00532
00534 AFTERWARP_API PixelFormat SurfaceGetFormat(struct Surface_t* surface);
00535
00539 AFTERWARP_API LibraryBool SurfaceGetPremultipliedAlpha(struct Surface_t* surface);
00540
00544 AFTERWARP_API void SurfaceSetPremultipliedAlpha(struct Surface_t* surface, LibraryBool
    premultipliedAlpha);
00545
00547 AFTERWARP_API void* SurfaceGetScanline(struct Surface_t* surface, int32_t index);
00548
00550 AFTERWARP_API void* SurfaceGetPixelPtr(struct Surface_t* surface, int32_t x, int32_t y);
00551
00555 AFTERWARP_API LibraryBool SurfaceEmpty(struct Surface_t* surface);
00556
00558 AFTERWARP_API LibraryBool SurfaceResize(struct Surface_t* surface, int32_t width, int32_t height,
00559     PixelFormat format);
00560
00562 AFTERWARP_API LibraryBool SurfaceConvertFormat(struct Surface_t* surface, PixelFormat format);
00563
00565 AFTERWARP_API Color SurfaceGetPixel(struct Surface_t* surface, int32_t x, int32_t y);
00566
00568 AFTERWARP_API void SurfaceSetPixel(struct Surface_t* surface, int32_t x, int32_t y, Color color);
00569
00573 AFTERWARP_API Color SurfaceGetPixelBilinear(struct Surface_t* surface, float x, float y);
00574
00580 AFTERWARP_API LibraryBool SurfaceCopyRect(struct Surface_t* surface, struct Surface_t* source,
00581     Rect const* sourceRect, Point const* destPos);
00582

```

```

00588 AFTERWARP_API LibraryBool SurfaceCopyFrom(struct Surface_t* surface, struct Surface_t* source);
00589
00591 AFTERWARP_API void SurfaceClear(struct Surface_t* surface);
00592
00594 AFTERWARP_API LibraryBool SurfaceClearWith(struct Surface_t* surface, Color color);
00595
00597 AFTERWARP_API LibraryBool SurfaceResetAlpha(struct Surface_t* surface, LibraryBool opaque);
00598
00605 AFTERWARP_API LibraryBool SurfaceHasAlphaChannel(struct Surface_t* surface);
00606
00612 AFTERWARP_API LibraryBool SurfacePremultiplyAlpha(struct Surface_t* surface);
00613
00621 AFTERWARP_API LibraryBool SurfaceUnpremultiplyAlpha(struct Surface_t* surface);
00622
00624 AFTERWARP_API LibraryBool SurfaceMirror(struct Surface_t* surface);
00625
00627 AFTERWARP_API LibraryBool SurfaceFlip(struct Surface_t* surface);
00628
00630 AFTERWARP_API LibraryBool SurfaceInvert(struct Surface_t* surface);
00631
00634 AFTERWARP_API LibraryBool SurfaceShrinkFrom(struct Surface_t* surface, struct Surface_t* source);
00635
00639 AFTERWARP_API LibraryBool SurfaceStretchRect(struct Surface_t* surface, struct Surface_t* source,
00640 RectF const* destRect, RectF const* sourceRect);
00641
00647 AFTERWARP_API LibraryBool SurfaceStretchRectBilinear(struct Surface_t* surface, struct Surface_t*
00648 source,
00649 RectF const* destRect, RectF const* sourceRect);
00651 AFTERWARP_API LibraryBool SurfaceMakeSignedDistanceField(struct Surface_t* surface,
00652 struct Surface_t* source, float spread, Point const* destPos, Rect const* sourceRect);
00653
00657 AFTERWARP_API LibraryBool SurfaceSaveToFile(struct Surface_t* surface, char const* fileName,
00658 int32_t quality);
00659
00666 AFTERWARP_API uint32_t SurfaceSaveToFileInMemory(struct Surface_t* surface, void* fileContents,
00667 uint32_t fileContentSize, char const* extension, int32_t quality);
00668
00669 // TODO: expose canvas for surface.
00670
00671 // CanvasBuffer functions.
00672
00675 AFTERWARP_API struct CanvasBuffer_t* CanvasBufferCreate(void);
00676
00678 AFTERWARP_API void CanvasBufferDestroy(struct CanvasBuffer_t* buffer);
00679
00681 AFTERWARP_API int32_t CanvasBufferGetVertexCount(struct CanvasBuffer_t* buffer);
00682
00684 AFTERWARP_API LibraryBool CanvasBufferSetVertexCount(struct CanvasBuffer_t* buffer, int32_t
00685 vertexCount);
00687 AFTERWARP_API int32_t CanvasBufferGetIndexCount(struct CanvasBuffer_t* buffer);
00688
00690 AFTERWARP_API LibraryBool CanvasBufferSetIndexCount(struct CanvasBuffer_t* buffer, int32_t
00691 indexCount);
00693 AFTERWARP_API void CanvasBufferClear(struct CanvasBuffer_t* buffer);
00694
00696 AFTERWARP_API void CanvasBufferClearAndShrink(struct CanvasBuffer_t* buffer);
00697
00699 AFTERWARP_API PointF* CanvasBufferGetVertices(struct CanvasBuffer_t* buffer);
00700
00702 AFTERWARP_API Color* CanvasBufferGetColors(struct CanvasBuffer_t* buffer);
00703
00705 AFTERWARP_API int32_t* CanvasBufferGetIndices(struct CanvasBuffer_t* buffer);
00706
00708 AFTERWARP_API void CanvasBufferGetExtents(struct CanvasBuffer_t* buffer, RectF* extents);
00709
00710 // PathBroker functions.
00711
00713 AFTERWARP_API struct PathBroker_t* PathBrokerCreate(void);
00714
00716 AFTERWARP_API void PathBrokerDestroy(struct PathBroker_t* broker);
00717
00720 AFTERWARP_API LibraryBool PathBrokerFill(struct PathBroker_t* broker, struct CanvasBuffer_t* buffer,
00721 PathElement const path[], int32_t pathLength, ColorRect const* colors, LibraryBool quality);
00722
00724 AFTERWARP_API LibraryBool PathBrokerStroke(struct PathBroker_t* broker, struct CanvasBuffer_t* buffer,
00725 PathElement const path[], int32_t pathLength, float thickness, Color color, PathJoint joints,
00726 LineCaps caps, float miterLimit, float smoothStep);
00727
00729 AFTERWARP_API void PathBrokerReset(struct PathBroker_t* broker);
00730
00731 // Canvas functions.
00732
00734 AFTERWARP_API struct Canvas_t* CanvasCreate(struct Device_t* device);
00735

```

```

00737 AFTERWARP_API void CanvasDestroy(struct Canvas_t* canvas);
00738
00740 AFTERWARP_API struct Device_t* CanvasGetDevice(struct Canvas_t* canvas);
00741
00743 AFTERWARP_API LibraryBool CanvasBegin(struct Canvas_t* canvas);
00744
00746 AFTERWARP_API void CanvasEnd(struct Canvas_t* canvas);
00747
00749 AFTERWARP_API void CanvasPixels(struct Canvas_t* canvas, PointF const positions[], Color const
    colors[],
00750     int32_t elementCount, BlendingEffect effect);
00751
00753 AFTERWARP_API void CanvasLines(struct Canvas_t* canvas, PointF const vertices[], Color const colors[],
00754     int32_t const indices[], int32_t vertexCount, int32_t primitiveCount, BlendingEffect effect);
00755
00757 AFTERWARP_API void CanvasTriangles(struct Canvas_t* canvas, PointF const vertices[], Color const
    colors[],
00758     int32_t const indices[], int32_t vertexCount, int32_t primitiveCount, BlendingEffect effect);
00759
00761 AFTERWARP_API void CanvasTexturedTriangles(struct Canvas_t* canvas, struct Texture_t* texture,
00762     PointF const vertices[], PointF const texCoords[], Color const colors[], int32_t const indices[],
00763     int32_t vertexCount, int32_t primitiveCount, BlendingEffect effect);
00764
00766 AFTERWARP_API void CanvasPixel(struct Canvas_t* canvas, PointF const* position, Color color,
00767     BlendingEffect effect);
00768
00770 AFTERWARP_API void CanvasLine(struct Canvas_t* canvas, PointF const* srcPos, PointF const* destPos,
00771     ColorPair const* colors, BlendingEffect effect);
00772
00774 AFTERWARP_API void CanvasThickLine(struct Canvas_t* canvas, PointF const* srcPos, PointF const*
    destPos,
00775     ColorPair const* colors, float thickness, BlendingEffect effect);
00776
00778 AFTERWARP_API void CanvasLineEllipse(struct Canvas_t* canvas, PointF const* origin, PointF const*
    radius,
00779     int32_t steps, Color color, BlendingEffect effect);
00780
00782 AFTERWARP_API void CanvasLineCircle(struct Canvas_t* canvas, PointF const* origin, float radius,
00783     int32_t steps, Color color, BlendingEffect effect);
00784
00786 AFTERWARP_API void CanvasThickLineEllipse(struct Canvas_t* canvas, PointF const* origin,
00787     PointF const* radius, int32_t steps, Color color, float thickness, BlendingEffect effect);
00788
00790 AFTERWARP_API void CanvasThickLineCircle(struct Canvas_t* canvas, PointF const* origin, float radius,
00791     int32_t steps, Color color, float thickness, BlendingEffect effect);
00792
00794 AFTERWARP_API void CanvasLineTriangle(struct Canvas_t* canvas, PointF const* vertex1, PointF const*
    vertex2,
00795     PointF const* vertex3, Color color1, Color color2, Color color3, BlendingEffect effect);
00796
00798 AFTERWARP_API void CanvasThickLineTriangle(struct Canvas_t* canvas, PointF const* vertex1,
00799     PointF const* vertex2, PointF const* vertex3, Color color1, Color color2, Color color3, float
    thickness,
00800     BlendingEffect effect);
00801
00803 AFTERWARP_API void CanvasLineQuad(struct Canvas_t* canvas, Quad const* vertices, ColorRect const*
    colors,
00804     BlendingEffect effect);
00805
00807 AFTERWARP_API void CanvasThickLineQuad(struct Canvas_t* canvas, Quad const* vertices,
00808     ColorRect const* colors, float thickness, BlendingEffect effect);
00809
00815 AFTERWARP_API void CanvasLineHexagon(struct Canvas_t* canvas, Color color1, Color color2, Color
    color3,
00816     Color color4, Color color5, Color color6, BlendingEffect effect);
00817
00823 AFTERWARP_API void CanvasLineHexagonGrad(struct Canvas_t* canvas, ColorRect const* colors,
00824     BlendingEffect effect);
00825
00830 AFTERWARP_API void CanvasThickLineHexagon(struct Canvas_t* canvas, Color color1, Color color2, Color
    color3,
00831     Color color4, Color color5, Color color6, float thickness, BlendingEffect effect);
00832
00837 AFTERWARP_API void CanvasThickLineHexagonGrad(struct Canvas_t* canvas, ColorRect const* colors,
00838     float thickness, BlendingEffect effect);
00839
00840 // TODO: lineArc and thickLineArc
00841
00843 AFTERWARP_API void CanvasTriangle(struct Canvas_t* canvas, PointF const* vertex1, PointF const*
    vertex2,
00844     PointF const* vertex3, Color color1, Color color2, Color color3, BlendingEffect effect);
00845
00847 AFTERWARP_API void CanvasQuad(struct Canvas_t* canvas, Quad const* vertices, ColorRect const* colors,
00848     BlendingEffect effect);
00849
00851 AFTERWARP_API void CanvasFillRect(struct Canvas_t* canvas, RectF const* rect, ColorRect const* colors,
00852     BlendingEffect effect);

```



```

00853
00855 AFTERWARP_API void CanvasFrameRect(struct Canvas_t* canvas, RectF const* rect, ColorRect const*
    colors,
00856     float thickness, BlendingEffect effect);
00857
00858 // NOTE: Other fillRect(), frameRect(), horizLine() and vertLine() overloads are not exposed.
00859
00861 AFTERWARP_API void CanvasFillRoundRect(struct Canvas_t* canvas, RectF const* rect, ColorRect const*
    colors,
00862     float radius, int32_t steps, BlendingEffect effect);
00863
00865 AFTERWARP_API void CanvasFrameRoundRect(struct Canvas_t* canvas, RectF const* rect, ColorRect const*
    colors,
00866     float radius, float thickness, int32_t steps, BlendingEffect effect);
00867
00869 AFTERWARP_API void CanvasFillRoundRectTop(struct Canvas_t* canvas, RectF const* rect,
    ColorRect const* colors, float radius, int32_t steps, BlendingEffect effect);
00870
00871 AFTERWARP_API void CanvasFillRoundRectBottom(struct Canvas_t* canvas, RectF const* rect,
    ColorRect const* colors, float radius, int32_t steps, BlendingEffect effect);
00872
00873 AFTERWARP_API void CanvasFillRoundRectTopInverse(struct Canvas_t* canvas, RectF const* rect,
    ColorRect const* colors, float radius, int32_t steps, BlendingEffect effect);
00874
00875 AFTERWARP_API void CanvasFillRoundRectBottomInverse(struct Canvas_t* canvas, RectF const* rect,
    ColorRect const* colors, float radius, int32_t steps, BlendingEffect effect);
00876
00877 AFTERWARP_API void CanvasHighlight(struct Canvas_t* canvas, RectF const* rect, float cornerRadius,
    float luma, float distance, int32_t steps);
00878
00880 AFTERWARP_API void CanvasHexagon(struct Canvas_t* canvas, Color color1, Color color2, Color color3,
    Color color4, Color color5, Color color6, BlendingEffect effect);
00881
00882 AFTERWARP_API void CanvasHexagonGrad(struct Canvas_t* canvas, ColorRect const* colors,
    BlendingEffect effect);
00883
00884 AFTERWARP_API void CanvasArc(struct Canvas_t* canvas, PointF const* origin, PointF const* radius,
    float initAngle, float endAngle, int32_t steps, Color colorInside, Color colorOutside,
    BlendingEffect effect);
00885
00886 AFTERWARP_API void CanvasArcGrad(struct Canvas_t* canvas, PointF const* origin, PointF const* radius,
    float initAngle, float endAngle, int32_t steps, ColorRect const* colors, BlendingEffect effect);
00887
00888 AFTERWARP_API void CanvasEllipse(struct Canvas_t* canvas, PointF const* origin, PointF const* radius,
    int32_t steps, ColorRect const* colors, BlendingEffect effect);
00889
00890 AFTERWARP_API void CanvasRibbonGrad(struct Canvas_t* canvas, PointF const* origin,
    PointF const* insideRadius, PointF const* outsideRadius, int32_t steps, ColorRect const* colors,
    BlendingEffect effect);
00891
00892 AFTERWARP_API void CanvasRibbonTri(struct Canvas_t* canvas, PointF const* origin,
    PointF const* insideRadius, PointF const* outsideRadius, int32_t steps, ColorPair const* color1,
    ColorPair const* color2, ColorPair const* color3, BlendingEffect effect);
00893
00894 AFTERWARP_API void CanvasRibbon(struct Canvas_t* canvas, PointF const* origin, PointF const*
    insideRadius,
00895     PointF const* outsideRadius, int32_t steps, Color color, BlendingEffect effect);
00896
00897 AFTERWARP_API void CanvasTapeGrad(struct Canvas_t* canvas, PointF const* origin, PointF const*
    insideRadius,
00898     PointF const* outsideRadius, float initAngle, float endAngle, int32_t steps, ColorRect const*
    colors,
00899     BlendingEffect effect);
00900
00901 AFTERWARP_API void CanvasTapeTri(struct Canvas_t* canvas, PointF const* origin, PointF const*
    insideRadius,
00902     PointF const* outsideRadius, float initAngle, float endAngle, int32_t steps, ColorPair const*
    color1,
00903     ColorPair const* color2, ColorPair const* color3, BlendingEffect effect);
00904
00905 AFTERWARP_API void CanvasTape(struct Canvas_t* canvas, PointF const* origin, PointF const*
    insideRadius,
00906     PointF const* outsideRadius, float initAngle, float endAngle, int32_t steps, Color color,
    BlendingEffect effect);
00907
00908 AFTERWARP_API void CanvasRectWithHole(struct Canvas_t* canvas, RectF const* rect, PointF const*
    holeOrigin,
00909     PointF const* holeRadius, ColorPair const* colors, int32_t steps, BlendingEffect effect);
00910
00911 AFTERWARP_API void CanvasDrawBuffer(struct Canvas_t* canvas, struct CanvasBuffer_t* buffer,
    BlendingEffect effect);
00912
00913 AFTERWARP_API void CanvasTexturedTriangle(struct Canvas_t* canvas, struct Texture_t* texture,
    PointF const* vertex1, PointF const* vertex2, PointF const* vertex3, PointF const* texCoord1,
    PointF const* texCoord2, PointF const* texCoord3, Color color1, Color color2, Color color3,
    BlendingEffect effect);
00914
00915 AFTERWARP_API void CanvasTexturedTriangleRegion(struct Canvas_t* canvas, struct Texture_t* texture,
    PointF const* vertex1, PointF const* vertex2, PointF const* vertex3, PointF const* texCoord1,
    PointF const* texCoord2, PointF const* texCoord3, Color color1, Color color2, Color color3,

```

```

00963     BlendingEffect effect);
00964
00967 AFTERWARP_API void CanvasTexturedQuad(struct Canvas_t* canvas, struct Texture_t* texture,
00968     Quad const* vertices, Quad const* texCoords, ColorRect const* colors, BlendingEffect effect);
00969
00972 AFTERWARP_API void CanvasTexturedQuadRegion(struct Canvas_t* canvas, struct Texture_t* texture,
00973     Quad const* vertices, Quad const* texCoords, ColorRect const* colors, BlendingEffect effect);
00974
00976 AFTERWARP_API void CanvasTexturedRoundRect(struct Canvas_t* canvas, struct Texture_t* texture,
00977     RectF const* rect, Quad const* texCoords, ColorRect const* colors, float radius, int32_t steps,
00978     BlendingEffect effect);
00979
00982 AFTERWARP_API void CanvasTexturedRoundRectRegion(struct Canvas_t* canvas, struct Texture_t* texture,
00983     RectF const* rect, Quad const* texCoords, ColorRect const* colors, float radius, int32_t steps,
00984     BlendingEffect effect);
00985
00987 AFTERWARP_API void CanvasQuadImage(struct Canvas_t* canvas, struct ImageAtlas_t* imageAtlas,
00988     Quad const* vertices, int32_t regionIndex, ColorRect const* colors, RectF const* sourceRect,
00989     uint32_t modifiers, BlendingEffect effect);
00990
00992 AFTERWARP_API void CanvasSetWorld(struct Canvas_t* canvas, Matrix const* world);
00993
00995 AFTERWARP_API void CanvasGetWorld(struct Canvas_t* canvas, Matrix* world);
00996
00998 AFTERWARP_API void CanvasSetViewProjection(struct Canvas_t* canvas, Matrix const* viewProjection);
00999
01001 AFTERWARP_API void CanvasGetViewProjection(struct Canvas_t* canvas, Matrix* viewProjection);
01002
01004 AFTERWARP_API void CanvasSetTransform(struct Canvas_t* canvas, Matrix3x2 const* transform);
01005
01007 AFTERWARP_API void CanvasGetTransform(struct Canvas_t* canvas, Matrix3x2* transform);
01008
01011 AFTERWARP_API void CanvasSetSignedDistanceField(struct Canvas_t* canvas,
01012     SignedDistanceField const* signedDistanceField);
01013
01015 AFTERWARP_API void CanvasGetSignedDistanceField(struct Canvas_t* canvas,
01016     SignedDistanceField* signedDistanceField);
01017
01019 AFTERWARP_API CanvasContextState CanvasGetContextState(struct Canvas_t* canvas);
01020
01022 AFTERWARP_API LibraryBool CanvasSetContextState(struct Canvas_t* canvas,
01023     CanvasContextState contextState);
01024
01026 AFTERWARP_API LibraryBool CanvasSetSamplerState(struct Canvas_t* canvas,
01027     CanvasSamplerState const* samplerState);
01028
01030 AFTERWARP_API LibraryBool CanvasGetSamplerState(struct Canvas_t* canvas,
01031     CanvasSamplerState* samplerState);
01032
01034 AFTERWARP_API void CanvasResetSamplerState(struct Canvas_t* canvas);
01035
01037 AFTERWARP_API void CanvasSetAttributes(struct Canvas_t* canvas, uint32_t attributes);
01038
01040 AFTERWARP_API uint32_t CanvasGetAttributes(struct Canvas_t* canvas);
01041
01043 AFTERWARP_API void CanvasFlush(struct Canvas_t* canvas);
01044
01046 AFTERWARP_API void CanvasReset(struct Canvas_t* canvas);
01047
01051 AFTERWARP_API int32_t CanvasGetBatchCount(struct Canvas_t* canvas);
01052
01054 AFTERWARP_API void CanvasGetClipRect(struct Canvas_t* canvas, Rect* clipRect);
01055
01057 AFTERWARP_API LibraryBool CanvasSetClipRect(struct Canvas_t* canvas, Rect const* rect);
01058
01059 // ImageAtlas functions.
01060
01062 AFTERWARP_API struct ImageAtlas_t* ImageAtlasCreate(struct Device_t* device);
01063
01065 AFTERWARP_API void ImageAtlasDestroy(struct ImageAtlas_t* atlas);
01066
01068 AFTERWARP_API struct Device_t* ImageAtlasGetDevice(struct ImageAtlas_t* atlas);
01069
01071 AFTERWARP_API int32_t ImageAtlasTextureCount(struct ImageAtlas_t* atlas);
01072
01074 AFTERWARP_API struct Texture_t* ImageAtlasTexture(struct ImageAtlas_t* atlas, int32_t textureIndex);
01075
01077 AFTERWARP_API int32_t ImageAtlasRegionCount(struct ImageAtlas_t* atlas);
01078
01080 AFTERWARP_API LibraryBool ImageAtlasRegion(struct ImageAtlas_t* atlas, int32_t regionIndex,
01081     ImageRegion* region);
01082
01084 AFTERWARP_API int32_t ImageAtlasCreateRegion(struct ImageAtlas_t* atlas, Rect const* rect,
01085     int32_t textureIndex);
01086
01088 AFTERWARP_API void ImageAtlasRemoveRegion(struct ImageAtlas_t* atlas, int32_t regionIndex);
01089

```



```

01091 AFTERWARP_API void ImageAtlasClearRegions(struct ImageAtlas_t* atlas);
01092
01094 AFTERWARP_API void ImageAtlasMakeRegions(struct ImageAtlas_t* atlas, Point const* patternSize,
01095     Point const* visibleSize, int32_t patternCount);
01096
01098 AFTERWARP_API void ImageAtlasClearTextures(struct ImageAtlas_t* atlas);
01099
01101 AFTERWARP_API void ImageAtlasRemoveTexture(struct ImageAtlas_t* atlas, int32_t textureIndex);
01102
01105 AFTERWARP_API int32_t ImageAtlasCreateTexture(struct ImageAtlas_t* atlas, Point const* size,
01106     PixelFormat format, TextureAttributes attributes);
01107
01110 AFTERWARP_API int32_t ImageAtlasPackRegion(struct ImageAtlas_t* atlas, Point const* size, int32_t
padding);
01111
01115 AFTERWARP_API int32_t ImageAtlasPackSurface(struct ImageAtlas_t* atlas, struct Surface_t* surface,
01116     Rect const* sourceRect, int32_t padding);
01117
01118 // TextRenderer functions.
01119
01121 AFTERWARP_API struct TextRenderer_t* TextRendererCreate(struct Canvas_t* canvas, Point const*
textureSize,
01122     PixelFormat pixelFormat, LibraryBool mipMapping);
01123
01125 AFTERWARP_API void TextRendererDestroy(struct TextRenderer_t* textRenderer);
01126
01128 AFTERWARP_API struct Canvas_t* TextRendererGetCanvas(struct TextRenderer_t* textRenderer);
01129
01131 AFTERWARP_API LibraryBool TextRendererSetFontParameters(struct TextRenderer_t* textRenderer,
01132     FontParameters const* parameters);
01133
01135 AFTERWARP_API LibraryBool TextRendererGetFontParameters(struct TextRenderer_t* textRenderer,
01136     FontParameters* parameters);
01137
01139 AFTERWARP_API void TextRendererExtent(struct TextRenderer_t* textRenderer, char const* text,
01140     PointF* dimensions, TextRenderModifiers const* modifiers);
01141
01143 AFTERWARP_API void TextRendererExtentByPixels(struct TextRenderer_t* textRenderer, char const* text,
01144     RectF* rect, TextRenderModifiers const* modifiers);
01145
01151 AFTERWARP_API int32_t TextRendererRects(struct TextRenderer_t* textRenderer, PointF* extent,
01152     _Out_opt_ TextEntryRect rects[], char const* text, TextRenderModifiers const* modifiers);
01153
01155 AFTERWARP_API void TextRendererDraw(struct TextRenderer_t* textRenderer, PointF const* position,
01156     char const* text, ColorPair const* colors, float alpha, TextRenderModifiers const* modifiers);
01157
01159 AFTERWARP_API void TextRendererDrawAligned(struct TextRenderer_t* textRenderer, PointF const*
position,
01160     char const* text, ColorPair const* colors, TextAlignment horizAlign, TextAlignment vertAlign, float
alpha,
01161     LibraryBool alignToPixels, TextRenderModifiers const* modifiers);
01162
01164 AFTERWARP_API void TextRendererDrawCentered(struct TextRenderer_t* textRenderer, PointF const*
position,
01165     char const* text, ColorPair const* colors, float alpha, LibraryBool alignToPixels,
01166     TextRenderModifiers const* modifiers);
01167
01169 AFTERWARP_API void TextRendererDrawAlignedByPixels(struct TextRenderer_t* textRenderer,
01170     PointF const* position, char const* text, ColorPair const* colors, TextAlignment horizAlign,
01171     TextAlignment vertAlign, float alpha, LibraryBool alignToPixels, TextRenderModifiers const*
modifiers);
01172
01174 AFTERWARP_API void TextRendererDrawCenteredByPixels(struct TextRenderer_t* textRenderer,
01175     PointF const* position, char const* text, ColorPair const* colors, float alpha, LibraryBool
alignToPixels,
01176     TextRenderModifiers const* modifiers);
01177
01178 // TextModeller functions.
01179
01181 AFTERWARP_API struct TextModeller_t* TextModellerCreate(struct Device_t* device);
01182
01184 AFTERWARP_API void TextModellerDestroy(struct TextModeller_t* textModeller);
01185
01187 AFTERWARP_API struct Device_t* TextModellerGetDevice(struct TextModeller_t* textModeller);
01188
01191 AFTERWARP_API UntypedHandle TextModellerSetFontParameters(struct TextModeller_t* textModeller,
01192     FontParameters const* parameters);
01193
01195 AFTERWARP_API UntypedHandle TextModellerGetFontProvider(struct TextModeller_t* textModeller);
01196
01199 AFTERWARP_API void TextModellerSetFontProvider(struct TextModeller_t* textModeller, UntypedHandle
provider);
01200
01202 AFTERWARP_API void TextModellerExtent(struct TextModeller_t* textModeller, char const* text, PointF*
extent,
01203     TextRenderModifiers const* modifiers);
01204

```

```

01206 AFTERWARP_API void TextModellerExtentByShape(struct TextModeller_t* textModeller, char const* text,
01207 RectF* rect, TextRenderModifiers const* modifiers);
01208
01212 AFTERWARP_API int32_t TextModellerRects(struct TextModeller_t* textModeller, PointF* extent,
01213 _Out_opt_ TextEntryRect rects[], char const* text, TextRenderModifiers const* modifiers);
01214
01216 AFTERWARP_API LibraryBool TextModellerDrawToCanvas(struct TextModeller_t* textModeller,
01217 struct Canvas_t* canvas, PointF const* position, char const* text, ColorPair const* colors,
01218 TextAlignment alignHoriz, TextAlignment alignVert, LibraryBool alignByShape,
01219 TextRenderModifiers const* modifiers, BlendingEffect effect);
01220
01222 AFTERWARP_API LibraryBool TextModellerDrawToCanvasBuffer(struct TextModeller_t* textModeller,
01223 struct CanvasBuffer_t* buffer, PointF const* position, char const* text, ColorPair const* colors,
01224 TextAlignment alignHoriz, TextAlignment alignVert, LibraryBool alignByShape,
01225 TextRenderModifiers const* modifiers);
01226
01230 AFTERWARP_API LibraryBool TextModellerDrawCurvedToCanvas(struct TextModeller_t* textModeller,
01231 struct Canvas_t* canvas, PointF const* position, char const* text, float radius, float angle,
01232 ColorPair const* colors, TextRenderModifiers const* modifiers, BlendingEffect effect);
01233
01237 AFTERWARP_API LibraryBool TextModellerDrawCurvedToCanvasBuffer(struct TextModeller_t* textModeller,
01238 struct CanvasBuffer_t* buffer, PointF const* position, char const* text, float radius, float angle,
01239 ColorPair const* colors, TextRenderModifiers const* modifiers);
01240
01243 AFTERWARP_API LibraryBool TextModellerDraw(struct TextModeller_t* textModeller, Vector const*
01244 position,
01245 char const* text, float depth, ColorPair const* colors, TextAlignment alignHoriz, TextAlignment
01246 alignVert,
01247 TextAlignment alignDepth, LibraryBool alignByShape, TextRenderModifiers const* modifiers);
01248
01252 AFTERWARP_API LibraryBool TextModellerDrawCurved(struct TextModeller_t* textModeller,
01253 Vector const* position, char const* text, float radius, float angle, float depth, ColorPair const*
01254 colors,
01255 TextAlignment alignDepth, TextRenderModifiers const* modifiers);
01256
01261 AFTERWARP_API LibraryBool TextModellerDrawDepthCurved(struct TextModeller_t* textModeller,
01262 Vector const* position, char const* text, float radius, float angle, float depth, ColorPair const*
01263 colors,
01264 TextRenderModifiers const* modifiers);
01265
01266 AFTERWARP_API void TextModellerClear(struct TextModeller_t* textModeller);
01267
01269 AFTERWARP_API LibraryBool TextModellerPrepare(struct TextModeller_t* textModeller);
01270
01272 AFTERWARP_API LibraryBool TextModellerRender(struct TextModeller_t* textModeller, struct Program_t*
01273 program,
01274 uint32_t channel);
01275
01276 AFTERWARP_API LibraryBool TextModellerCopyToMeshBuffer(struct TextModeller_t* textModeller,
01277 struct MeshBuffer_t* meshBuffer);
01278
01280 AFTERWARP_API void TextModellerReset(struct TextModeller_t* textModeller);
01281
01283 AFTERWARP_API void TextModellerGetTransform(struct TextModeller_t* textModeller, Matrix* transform);
01284
01286 AFTERWARP_API void TextModellerSetTransform(struct TextModeller_t* textModeller, Matrix const*
01287 transform);
01288
01288 // Grapher functions.
01289
01291 AFTERWARP_API struct Grapher_t* GrapherCreate(struct Device_t* device);
01292
01294 AFTERWARP_API void GrapherDestroy(struct Grapher_t* grapher);
01295
01297 AFTERWARP_API struct Device_t* GrapherGetDevice(struct Grapher_t* grapher);
01298
01300 AFTERWARP_API LibraryBool GrapherBegin(struct Grapher_t* grapher);
01301
01303 AFTERWARP_API void GrapherEnd(struct Grapher_t* grapher);
01304
01307 AFTERWARP_API void GrapherPoints(struct Grapher_t* grapher, Vector4 const vertices[], Color const
01308 colors[],
01309 float const angles[], int32_t elementCount, PointShape shape);
01310
01311 AFTERWARP_API void GrapherLines(struct Grapher_t* grapher, Vector4 const vertices[], Color const
01312 colors[],
01313 int32_t const indices[], int32_t vertexCount, int32_t indexCount, LineCaps caps);
01314
01315 AFTERWARP_API void GrapherPoint(struct Grapher_t* grapher, Vector const* position, Color color, float
01316 size,
01317 PointShape shape, float angle);
01318
01319 AFTERWARP_API void GrapherLine(struct Grapher_t* grapher, Vector const* position1, Vector const*
01320 position2,
01321 Color color1, Color color2, float thickness1, float thickness2, LineCaps caps);
01322
01324 AFTERWARP_API void GrapherArrow(struct Grapher_t* grapher, PointF const* targetSize, Vector const*

```

```

    origin,
01325     Vector const* destination, Color color, float thickness, float size, LineCaps caps);
01326
01328 AFTERWARP_API void GrapherBoundingBox(struct Grapher_t* grapher, Matrix const* volume, Color color,
01329     float thickness, float length, LineCaps caps);
01330
01332 AFTERWARP_API void GrapherDottedLine(struct Grapher_t* grapher, Vector const* position1,
01333     Vector const* position2, Color color1, Color color2, float thickness1, float thickness2, float
    sparsity,
01334     PointShape shape, float angle);
01335
01338 AFTERWARP_API void GrapherFlush(struct Grapher_t* grapher);
01339
01343 AFTERWARP_API void GrapherReset(struct Grapher_t* grapher);
01344
01346 AFTERWARP_API void GrapherGetTransform(struct Grapher_t* grapher, Matrix* transform);
01347
01349 AFTERWARP_API void GrapherSetTransform(struct Grapher_t* grapher, Matrix const* transform);
01350
01354 AFTERWARP_API int32_t GrapherGetBatchCount(struct Grapher_t* grapher);
01355
01356 // MeshModel functions.
01357
01359 AFTERWARP_API struct MeshModel_t* MeshModelCreate(struct Device_t* device, int32_t vertexCount,
01360     uint32_t vertexElementPitch, int32_t indexCount, uint32_t channel, void const* initialVertexData,
01361     void const* initialIndexData);
01362
01364 AFTERWARP_API void MeshModelDestroy(struct MeshModel_t* meshModel);
01365
01368 AFTERWARP_API LibraryBool MeshModelRenderable(struct MeshModel_t* meshModel);
01369
01372 AFTERWARP_API void MeshModelDismantle(struct MeshModel_t* meshModel);
01373
01375 AFTERWARP_API void MeshModelTakeAway(struct MeshModel_t* meshModel, struct MeshModel_t*
    meshModelAnother);
01376
01378 AFTERWARP_API int32_t MeshModelGetVertexCount(struct MeshModel_t* meshModel);
01379
01381 AFTERWARP_API int32_t MeshModelGetIndexCount(struct MeshModel_t* meshModel);
01382
01384 AFTERWARP_API uint32_t MeshModelGetChannel(struct MeshModel_t* meshModel);
01385
01387 AFTERWARP_API struct Buffer_t* MeshModelGetVertexBuffer(struct MeshModel_t* meshModel);
01388
01390 AFTERWARP_API struct Buffer_t* MeshModelGetIndexBuffer(struct MeshModel_t* meshModel);
01391
01393 AFTERWARP_API LibraryBool MeshModelDraw(struct MeshModel_t* meshModel, struct Program_t* program,
01394     PrimitiveTopology topology, int32_t elementCount, int32_t firstIndex, int32_t baseVertex,
01395     LibraryBool postUnbind);
01396
01398 AFTERWARP_API LibraryBool MeshModelDrawInstances(struct MeshModel_t* meshModel, struct Program_t*
    program,
01399     int32_t instanceCount, PrimitiveTopology topology, int32_t elementCount, int32_t firstIndex,
01400     int32_t baseVertex, LibraryBool postUnbind);
01401
01402 // MeshMetaTag functions.
01403
01405 AFTERWARP_API struct MeshMetaTags_t* MeshMetaTagGetParent(struct MeshMetaTag_t* tag);
01406
01410 AFTERWARP_API void MeshMetaTagGetName(struct MeshMetaTag_t* tag, char* name, _Inout_ int32_t*
    nameLength);
01411
01413 AFTERWARP_API uint8_t MeshMetaTagGetType(struct MeshMetaTag_t* tag);
01414
01416 AFTERWARP_API void MeshMetaTagGetBounds(struct MeshMetaTag_t* tag, Vector* boundsMin, Vector*
    boundsMax);
01417
01419 AFTERWARP_API int32_t MeshMetaTagGetPortionCount(struct MeshMetaTag_t* tag);
01420
01422 AFTERWARP_API LibraryBool MeshMetaTagPortionGet(struct MeshMetaTag_t* tag, MeshMetaTagPortion*
    portion,
01423     int32_t index);
01424
01426 AFTERWARP_API LibraryBool MeshMetaTagPortionAdd(struct MeshMetaTag_t* tag,
01427     MeshMetaTagPortion const* portion);
01428
01430 AFTERWARP_API void MeshMetaTagPortionErase(struct MeshMetaTag_t* tag, int32_t index);
01431
01433 AFTERWARP_API void MeshMetaTagPortionsClear(struct MeshMetaTag_t* tag);
01434
01436 AFTERWARP_API LibraryBool MeshMetaTagPortionsCopy(struct MeshMetaTag_t* tag, struct MeshMetaTag_t*
    source);
01437
01438 // MeshMetaTags functions.
01439
01441 AFTERWARP_API struct MeshMetaTags_t* MeshMetaTagsCreate(void);
01442

```

```

01444 AFTERWARP_API void MeshMetaTagsDestroy(struct MeshMetaTags_t* tags);
01445
01447 AFTERWARP_API int32_t MeshMetaTagsCount(struct MeshMetaTags_t* tags);
01448
01450 AFTERWARP_API struct MeshMetaTag_t* MeshMetaTagsGetByIndex(struct MeshMetaTags_t* tags, int32_t
index);
01451
01453 AFTERWARP_API struct MeshMetaTag_t* MeshMetaTagsGetByName(struct MeshMetaTags_t* tags, char const*
name,
01454     uint8_t type);
01455
01457 AFTERWARP_API struct MeshMetaTag_t* MeshMetaTagsAdd(struct MeshMetaTags_t* tags, char const* name,
01458     uint8_t type);
01459
01461 AFTERWARP_API void MeshMetaTagsErase(struct MeshMetaTags_t* tags, int32_t index);
01462
01464 AFTERWARP_API void MeshMetaTagsClear(struct MeshMetaTags_t* tags);
01465
01467 AFTERWARP_API LibraryBool MeshMetaTagsCopy(struct MeshMetaTags_t* tags, struct MeshMetaTags_t*
tagsAnother);
01468
01470 AFTERWARP_API void MeshMetaTagsTakeAway(struct MeshMetaTags_t* tags, struct MeshMetaTags_t*
tagsAnother);
01471
01472 // MeshBuffer functions.
01473
01475 AFTERWARP_API struct MeshBuffer_t* MeshBufferCreate(void);
01476
01478 AFTERWARP_API void MeshBufferDestroy(struct MeshBuffer_t* meshBuffer);
01479
01481 AFTERWARP_API MeshBufferEntry* MeshBufferGetVertices(struct MeshBuffer_t* meshBuffer);
01482
01484 AFTERWARP_API int32_t MeshBufferGetVertexCount(struct MeshBuffer_t* meshBuffer);
01485
01487 AFTERWARP_API LibraryBool MeshBufferSetVertexCount(struct MeshBuffer_t* meshBuffer, int32_t
vertexCount);
01488
01490 AFTERWARP_API int32_t* MeshBufferGetIndices(struct MeshBuffer_t* meshBuffer);
01491
01493 AFTERWARP_API int32_t MeshBufferGetIndexCount(struct MeshBuffer_t* meshBuffer);
01494
01496 AFTERWARP_API LibraryBool MeshBufferSetIndexCount(struct MeshBuffer_t* meshBuffer, int32_t
indexCount);
01497
01499 AFTERWARP_API struct MeshModel_t* MeshBufferCreateModel(struct MeshBuffer_t* meshBuffer,
01500     struct Device_t* device, VertexElement const vertexElements[], int32_t vertexElementCount,
01501     int32_t firstVertex, int32_t vertexCount, int32_t firstIndex, int32_t indexCount, uint32_t channel);
01502
01506 AFTERWARP_API LibraryBool MeshBufferSuperEllipse(struct MeshBuffer_t* meshBuffer, int32_t
longitudeSections,
01507     int32_t latitudeSections, Vector const* origin, Vector const* radius, float initLongitudeAngle,
01508     float endLongitudeAngle, float longitudeShape, float initLatitudeAngle, float endLatitudeAngle,
01509     float latitudeShape, PointF const* initTexCoord, PointF const* endTexCoord, FloatColor const* color,
01510     LibraryBool indicesClockwise);
01511
01513 AFTERWARP_API LibraryBool MeshBufferGeosphere(struct MeshBuffer_t* meshBuffer, int32_t subdivisions,
01514     FloatColor const* color, LibraryBool flatNormals, LibraryBool indicesClockwise);
01515
01518 AFTERWARP_API LibraryBool MeshBufferCylinder(struct MeshBuffer_t* meshBuffer, int32_t radialSections,
01519     int32_t heightSections, Vector const* origin, Vector const* horizAxis, Vector const* vertAxis,
01520     Vector const* heightAxis, float initAngle, float endAngle, float shape, PointF const* initTexCoord,
01521     PointF const* endTexCoord, float bendAngle, float lateralAngle, FloatColor const* color,
01522     LibraryBool indicesClockwise);
01523
01525 AFTERWARP_API LibraryBool MeshBufferConeSimple(struct MeshBuffer_t* meshBuffer, int32_t sections,
01526     Vector const* origin, float radius, float height, FloatColor const* color, LibraryBool
indicesClockwise);
01527
01531 AFTERWARP_API LibraryBool MeshBufferDisc(struct MeshBuffer_t* meshBuffer, int32_t radialSections,
01532     int32_t innerSections, Vector const* origin, Vector const* horizAxis, Vector const* vertAxis,
01533     Vector const* normal, float initAngle, float endAngle, float shape, float initRadius,
01534     PointF const* initTexCoord, PointF const* endTexCoord, float lateralAngle, FloatColor const* color,
01535     LibraryBool indicesClockwise);
01536
01538 AFTERWARP_API LibraryBool MeshBufferPlane(struct MeshBuffer_t* meshBuffer, int32_t horizSections,
01539     int32_t vertSections, Vector const* origin, Vector const* horizAxis, Vector const* vertAxis,
01540     Vector const* normal, PointF const* initTexCoord, PointF const* endTexCoord, FloatColor const*
color,
01541     LibraryBool indicesClockwise);
01542
01544 AFTERWARP_API LibraryBool MeshBufferCube(struct MeshBuffer_t* meshBuffer, int32_t horizSections,
01545     int32_t vertSections, int32_t depthSections, Vector const* origin, Vector const* horizAxis,
01546     Vector const* vertAxis, Vector const* depthAxis, PointF const* initTexCoord, PointF const*
endTexCoord,
01547     FloatColor const* color, LibraryBool indicesClockwise);
01548
01551 AFTERWARP_API LibraryBool MeshBufferCubeMinimal(struct MeshBuffer_t* meshBuffer, Vector const* origin,

```

```

01552     Vector const* size, FloatColor const* color, LibraryBool indicesClockwise);
01553
01555 AFTERWARP_API LibraryBool MeshBufferCubeRound(struct MeshBuffer_t* meshBuffer, Vector const* origin,
01556     Vector const* size, float roundness, FloatColor const* color, LibraryBool indicesClockwise);
01557
01560 AFTERWARP_API LibraryBool MeshBufferTorus(struct MeshBuffer_t* meshBuffer, int32_t outerSections,
01561     int32_t innerSections, Vector const* origin, Vector const* horizAxis, Vector const* vertAxis,
01562     float initOuterAngle, float endOuterAngle, float outerShape, float initInnerAngle, float
01563     endInnerAngle,
01564     float innerShape, PointF const* innerRadius, PointF const* initTexCoord, PointF const* endTexCoord,
01565     float lateralAngle, FloatColor const* color, LibraryBool indicesClockwise);
01566
01568 AFTERWARP_API LibraryBool MeshBufferTorusKnot(struct MeshBuffer_t* meshBuffer, int32_t outerSections,
01569     int32_t innerSections, Vector const* origin, int32_t p, int32_t q, float initOuterAngle,
01570     float endOuterAngle, float outerRadius, float initInnerAngle, float endInnerAngle, float innerShape,
01571     PointF const* innerRadius, PointF const* initTexCoord, PointF const* endTexCoord, float
01572     lateralAngle,
01573     FloatColor const* color, LibraryBool indicesClockwise);
01574
01576 AFTERWARP_API LibraryBool MeshBufferSupertoroid(struct MeshBuffer_t* meshBuffer, int32_t
01577     outerSections,
01578     int32_t innerSections, Vector const* origin, float initOuterAngle, float endOuterAngle, float
01579     outerRadius,
01580     float outerShape, float initInnerAngle, float endInnerAngle, float innerRadius, float innerShape,
01581     PointF const* initTexCoord, PointF const* endTexCoord, FloatColor const* color,
01582     LibraryBool indicesClockwise);
01583
01583 AFTERWARP_API LibraryBool MeshBufferFrustumVolume(struct MeshBuffer_t* meshBuffer,
01584     Matrix const* viewProjectionInverse, LibraryBool depthClipNegative);
01585
01589 AFTERWARP_API LibraryBool MeshBufferExtrusion(struct MeshBuffer_t* meshBuffer, PathElement const
01590     path[],
01591     int32_t pathLength, float depth, PointF const* texCoords, FloatColor const colors[], LibraryBool
01592     quality);
01593
01593 AFTERWARP_API void MeshBufferTransformVertices(struct MeshBuffer_t* meshBuffer, int32_t firstVertex,
01594     int32_t vertexCount);
01595
01598 AFTERWARP_API LibraryBool MeshBufferCalculateFlatNormals(struct MeshBuffer_t* meshBuffer, float
01599     epsilon);
01600
01602 AFTERWARP_API void MeshBufferCalculateNormals(struct MeshBuffer_t* meshBuffer, int32_t firstVertex,
01603     int32_t vertexCount, int32_t firstIndex, int32_t indexCount);
01604
01608 AFTERWARP_API void MeshBufferCalculateNormalsWeld(struct MeshBuffer_t* meshBuffer, int32_t
01609     firstVertex,
01610     int32_t vertexCount, int32_t firstIndex, int32_t indexCount, float weldEpsilon);
01611
01613 AFTERWARP_API void MeshBufferInvertNormals(struct MeshBuffer_t* meshBuffer, int32_t firstVertex,
01614     int32_t vertexCount);
01615
01617 AFTERWARP_API void MeshBufferInvertIndexOrder(struct MeshBuffer_t* meshBuffer, int32_t firstIndex,
01618     int32_t indexCount);
01619
01621 AFTERWARP_API void MeshBufferCalculateBounds(struct MeshBuffer_t* meshBuffer, Vector* vertexMin,
01622     Vector* vertexMax, int32_t firstVertex, int32_t vertexCount);
01623
01625 AFTERWARP_API void MeshBufferCentralize(struct MeshBuffer_t* meshBuffer, int32_t firstVertex,
01626     int32_t vertexCount);
01627
01630 AFTERWARP_API void MeshBufferEliminateUnusedVertices(struct MeshBuffer_t* meshBuffer, int32_t
01631     firstVertex,
01632     int32_t vertexCount);
01633
01635 AFTERWARP_API LibraryBool MeshBufferJoinDuplicateVertices(struct MeshBuffer_t* meshBuffer,
01636     int32_t firstVertex, int32_t vertexCount, float threshold);
01637
01639 AFTERWARP_API LibraryBool MeshBufferCombine(struct MeshBuffer_t* meshBuffer,
01640     struct MeshBuffer_t* meshBufferSource, int32_t firstVertex, int32_t vertexCount, int32_t firstIndex,
01641     int32_t indexCount);
01642
01644 AFTERWARP_API void MeshBufferClear(struct MeshBuffer_t* meshBuffer);
01645
01647 AFTERWARP_API void MeshBufferSetTransform(struct MeshBuffer_t* meshBuffer, Matrix const* transform);
01648
01650 AFTERWARP_API void MeshBufferGetTransform(struct MeshBuffer_t* meshBuffer, Matrix* transform);
01651
01655 AFTERWARP_API uint32_t MeshBufferTransferVertexElements(struct MeshBuffer_t* meshBuffer, void* buffer,
01656     VertexElement const vertexElements[], int32_t vertexElementCount, int32_t firstVertex, int32_t
01657     vertexCount,
01658     uint32_t channel, uint32_t semanticIndex);
01659
01662 AFTERWARP_API uint32_t MeshBufferTransferIndexElements(struct MeshBuffer_t* meshBuffer, void* buffer,
01663     uint32_t pitch, int32_t firstVertex, int32_t firstIndex, int32_t indexCount);
01664
01667 AFTERWARP_API LibraryBool MeshBufferTransferVertices(struct MeshBuffer_t* meshBuffer,
01668     struct Buffer_t* buffer, VertexElement const vertexElements[], int32_t vertexElementCount,

```

```

01669     int32_t firstVertex, int32_t vertexCount, uint32_t channel, uint32_t offset, uint32_t
        semanticIndex);
01670
01672 AFTERWARD_API LibraryBool MeshBufferTransferIndices(struct MeshBuffer_t* meshBuffer,
01673     struct Buffer_t* buffer, int32_t firstVertex, int32_t firstIndex, int32_t indexCount, uint32_t
        offset);
01674
01678 AFTERWARD_API struct MeshVoxel_t* MeshBufferVoxelize(struct MeshBuffer_t* meshBuffer, uint8_t levels,
01679     LibraryBool colors);
01680
01684 AFTERWARD_API LibraryBool MeshBufferLoadFromFile(struct MeshBuffer_t* meshBuffer,
01685     char const* fileName, struct SceneMeshMaterials_t* materials, struct MeshMetaTags_t* tags,
01686     _Inout_ uint32_t* options, MeshLoadSaveFeedback feedback, void* feedbackUser,
01687     char* debug, _Inout_ int32_t* debugLength);
01688
01691 AFTERWARD_API LibraryBool MeshBufferSaveToFile(struct MeshBuffer_t* meshBuffer,
01692     char const* fileName, struct SceneMeshMaterials_t* materials, struct MeshMetaTags_t* tags,
01693     uint32_t options, MeshLoadSaveFeedback feedback, void* feedbackUser,
01694     char* debug, _Inout_ int32_t* debugLength);
01695
01699 AFTERWARD_API LibraryBool MeshBufferLoadFromFileEx(struct MeshBuffer_t* meshBuffer,
01700     struct MeshMetaTags_t* tags, char const* fileName, Vector* minBounds, Vector* maxBounds,
01701     LibraryBool normals, char* debug, _Inout_ int32_t* debugLength);
01702
01705 AFTERWARD_API LibraryBool MeshBufferSaveToFileEx(struct MeshBuffer_t* meshBuffer, char const*
        fileName,
01706     uint32_t const* options, char* debug, _Inout_ int32_t* debugLength);
01707
01708 // GaussianBlur functions.
01709
01711 AFTERWARD_API struct GaussianBlur_t* GaussianBlurCreate(struct Device_t* device, float sigma,
01712     int32_t samples, LibraryBool hardwareFiltering);
01713
01715 AFTERWARD_API void GaussianBlurDestroy(struct GaussianBlur_t* gaussianBlur);
01716
01718 AFTERWARD_API struct Device_t* GaussianBlurGetDevice(struct GaussianBlur_t* gaussianBlur);
01719
01721 AFTERWARD_API int32_t GaussianBlurGetFixedSamples(struct GaussianBlur_t* gaussianBlur);
01722
01724 AFTERWARD_API int32_t GaussianBlurGetSamples(struct GaussianBlur_t* gaussianBlur);
01725
01730 AFTERWARD_API void GaussianBlurSetSamples(struct GaussianBlur_t* gaussianBlur, int32_t samples);
01731
01733 AFTERWARD_API float GaussianBlurGetSigma(struct GaussianBlur_t* gaussianBlur);
01734
01736 AFTERWARD_API void GaussianBlurSetSigma(struct GaussianBlur_t* gaussianBlur, float sigma);
01737
01739 AFTERWARD_API float GaussianBlurGetChroma(struct GaussianBlur_t* gaussianBlur);
01740
01742 AFTERWARD_API void GaussianBlurSetChroma(struct GaussianBlur_t* gaussianBlur, float chroma);
01743
01745 AFTERWARD_API LibraryBool GaussianBlurGetHardwareFiltering(struct GaussianBlur_t* gaussianBlur);
01746
01748 AFTERWARD_API void GaussianBlurSetHardwareFiltering(struct GaussianBlur_t* gaussianBlur,
01749     LibraryBool hardwareFiltering);
01750
01754 AFTERWARD_API LibraryBool GaussianBlurUpdate(struct GaussianBlur_t* gaussianBlur,
01755     struct Texture_t* destination, struct Texture_t* intermediary, struct Texture_t* source);
01756
01762 AFTERWARD_API LibraryBool GaussianBlurUpdateAt(struct GaussianBlur_t* gaussianBlur,
01763     struct Texture_t* destination, struct Texture_t* intermediary, struct Texture_t* source,
01764     Point const* destPos, Rect const* intermRect);
01765
01766 // KawaseBlur functions.
01767
01769 AFTERWARD_API struct KawaseBlur_t* KawaseBlurCreate(struct Device_t* device);
01770
01772 AFTERWARD_API void KawaseBlurDestroy(struct KawaseBlur_t* kawaseBlur);
01773
01775 AFTERWARD_API struct Device_t* KawaseBlurGetDevice(struct KawaseBlur_t* kawaseBlur);
01776
01778 AFTERWARD_API int32_t KawaseBlurGetPasses(struct KawaseBlur_t* kawaseBlur);
01779
01781 AFTERWARD_API void KawaseBlurSetPasses(struct KawaseBlur_t* kawaseBlur, int32_t passes);
01782
01784 AFTERWARD_API void KawaseBlurGetOffset(struct KawaseBlur_t* kawaseBlur, PointF* offset);
01785
01787 AFTERWARD_API void KawaseBlurSetOffset(struct KawaseBlur_t* kawaseBlur, PointF const* offset);
01788
01792 AFTERWARD_API LibraryBool KawaseBlurUpdate(struct KawaseBlur_t* kawaseBlur,
01793     struct Texture_t* destination, struct Texture_t* intermediary, struct Texture_t* source);
01794
01797 AFTERWARD_API LibraryBool KawaseBlurSinglePass(struct KawaseBlur_t* kawaseBlur,
01798     struct Texture_t* destination, struct Texture_t* source, PointF const* offset);
01799
01800 // ColorDithering functions.
01801

```



```

01803 AFTERWARP_API struct ColorDithering_t* ColorDitheringCreate(struct Device_t* device);
01804
01806 AFTERWARP_API void ColorDitheringDestroy(struct ColorDithering_t* colorDithering);
01807
01809 AFTERWARP_API struct Device_t* ColorDitheringGetDevice(struct ColorDithering_t* colorDithering);
01810
01812 AFTERWARP_API ColorDitheringFormat ColorDitheringGetFormat(struct ColorDithering_t* colorDithering);
01813
01815 AFTERWARP_API void ColorDitheringSetFormat(struct ColorDithering_t* colorDithering,
01816     ColorDitheringFormat format);
01817
01819 AFTERWARP_API LibraryBool ColorDitheringExecuteTo(struct ColorDithering_t* colorDithering,
01820     struct Texture_t* destination, struct Texture_t* source);
01821
01823 AFTERWARP_API LibraryBool ColorDitheringExecute(struct ColorDithering_t* colorDithering,
01824     struct Texture_t* source);
01825
01826 // SelectionHighlight functions.
01827
01829 AFTERWARP_API struct SelectionHighlight_t* SelectionHighlightCreate(struct Device_t* device,
01830     Point const* size, PixelFormat depthStencil, int32_t samples, LibraryBool grayscale);
01831
01833 AFTERWARP_API void SelectionHighlightDestroy(struct SelectionHighlight_t* selectionHighlight);
01834
01836 AFTERWARP_API struct Device_t* SelectionHighlightGetDevice(struct SelectionHighlight_t*
01837     selectionHighlight);
01838
01839 AFTERWARP_API LibraryBool SelectionHighlightSetSize(struct SelectionHighlight_t* selectionHighlight,
01840     Point const* size);
01841
01843 AFTERWARP_API void SelectionHighlightGetSize(struct SelectionHighlight_t* selectionHighlight, Point*
01844     size);
01845
01846 AFTERWARP_API PixelFormat SelectionHighlightGetDepthStencil(struct SelectionHighlight_t*
01847     selectionHighlight);
01848
01849 AFTERWARP_API int32_t SelectionHighlightGetSamples(struct SelectionHighlight_t* selectionHighlight);
01850
01852 AFTERWARP_API LibraryBool SelectionHighlightGetGrayscale(struct SelectionHighlight_t*
01853     selectionHighlight);
01854
01855 AFTERWARP_API void SelectionHighlightGetParameters(struct SelectionHighlight_t* selectionHighlight,
01856     SelectionHighlightParameters* parameters);
01857
01859 AFTERWARP_API LibraryBool SelectionHighlightSetParameters(struct SelectionHighlight_t*
01860     selectionHighlight,
01861     SelectionHighlightParameters const* parameters);
01862
01863 AFTERWARP_API LibraryBool SelectionHighlightClear(struct SelectionHighlight_t* selectionHighlight);
01864
01866 AFTERWARP_API LibraryBool SelectionHighlightBegin(struct SelectionHighlight_t* selectionHighlight);
01867
01869 AFTERWARP_API void SelectionHighlightEnd(struct SelectionHighlight_t* selectionHighlight);
01870
01872 AFTERWARP_API LibraryBool SelectionHighlightRendering(struct SelectionHighlight_t*
01873     selectionHighlight);
01874
01875 AFTERWARP_API LibraryBool SelectionHighlightFilter(struct SelectionHighlight_t* selectionHighlight);
01876
01878 AFTERWARP_API struct Texture_t* SelectionHighlightGetTexture(struct SelectionHighlight_t*
01879     selectionHighlight,
01880     SelectionHighlightTextureType type);
01881
01882 AFTERWARP_API void SelectionHighlightGetTextureCoordinates(struct SelectionHighlight_t*
01883     selectionHighlight,
01884     Quad* coords);
01885
01886 // SpatialFog functions.
01887
01888 AFTERWARP_API struct SpatialFog_t* SpatialFogCreate(struct Device_t* device);
01889
01891 AFTERWARP_API void SpatialFogDestroy(struct SpatialFog_t* spatialFog);
01892
01894 AFTERWARP_API struct Device_t* SpatialFogGetDevice(struct SpatialFog_t* spatialFog);
01895
01897 AFTERWARP_API void SpatialFogGetParameters(struct SpatialFog_t* spatialFog, FogParameters*
01898     parameters);
01899
01900 AFTERWARP_API LibraryBool SpatialFogSetParameters(struct SpatialFog_t* spatialFog,
01901     FogParameters const* parameters);
01902
01904 AFTERWARP_API void SpatialFogGetView(struct SpatialFog_t* spatialFog, Matrix* view);
01905
01907 AFTERWARP_API LibraryBool SpatialFogSetView(struct SpatialFog_t* spatialFog, Matrix const* view);
01908
01910 AFTERWARP_API void SpatialFogGetProjection(struct SpatialFog_t* spatialFog, Matrix* projection);
01911

```

```

01913 AFTERWARP_API LibraryBool SpatialFogSetProjection(struct SpatialFog_t* spatialFog, Matrix const*
    projection);
01914
01916 AFTERWARP_API FogFormula SpatialFogGetFormula(struct SpatialFog_t* spatialFog);
01917
01919 AFTERWARP_API LibraryBool SpatialFogSetFormula(struct SpatialFog_t* spatialFog, FogFormula formula);
01920
01922 AFTERWARP_API DepthFogDistance SpatialFogGetDepthDistance(struct SpatialFog_t* spatialFog);
01923
01925 AFTERWARP_API LibraryBool SpatialFogSetDepthDistance(struct SpatialFog_t* spatialFog,
    DepthFogDistance depthDistance);
01926
01927
01930 AFTERWARP_API LibraryBool SpatialFogExecute(struct SpatialFog_t* spatialFog,
    struct Texture_t* destination, struct Texture_t* linearDepths);
01931
01932
01935 AFTERWARP_API LibraryBool SpatialFogExecuteGlassy(struct SpatialFog_t* spatialFog,
    struct Texture_t* destination, struct Texture_t* linearDepths);
01936
01937
01938 // SceneLights functions.
01939
01941 AFTERWARP_API struct SceneLights_t* SceneLightsCreate(struct Device_t* device);
01942
01944 AFTERWARP_API void SceneLightsDestroy(struct SceneLights_t* sceneLights);
01945
01947 AFTERWARP_API struct Device_t* SceneLightsGetDevice(struct SceneLights_t* sceneLights);
01948
01950 AFTERWARP_API int32_t SceneLightsGetCount(struct SceneLights_t* sceneLights);
01951
01953 AFTERWARP_API SceneLight* SceneLightsGetElement(struct SceneLights_t* sceneLights, int32_t index);
01954
01956 AFTERWARP_API SceneLight* SceneLightsAdd(struct SceneLights_t* sceneLights);
01957
01959 AFTERWARP_API LibraryBool SceneLightsErase(struct SceneLights_t* sceneLights, int32_t index);
01960
01962 AFTERWARP_API void SceneLightsClear(struct SceneLights_t* sceneLights);
01963
01965 AFTERWARP_API void SceneLightsGetViewSize(struct SceneLights_t* sceneLights, Point* viewSize);
01966
01968 AFTERWARP_API void SceneLightsSetViewSize(struct SceneLights_t* sceneLights, Point const* viewSize);
01969
01971 AFTERWARP_API int32_t SceneLightsGetClusterSize(struct SceneLights_t* sceneLights);
01972
01974 AFTERWARP_API void SceneLightsSetClusterSize(struct SceneLights_t* sceneLights, int32_t clusterSize);
01975
01977 AFTERWARP_API int32_t SceneLightsGetDepthSlices(struct SceneLights_t* sceneLights);
01978
01980 AFTERWARP_API void SceneLightsSetDepthSlices(struct SceneLights_t* sceneLights, int32_t depthSlices);
01981
01983 AFTERWARP_API ClustersCullingMode SceneLightsGetCullingMode(struct SceneLights_t* sceneLights);
01984
01986 AFTERWARP_API void SceneLightsSetCullingMode(struct SceneLights_t* sceneLights, ClustersCullingMode
    mode);
01987
01989 AFTERWARP_API void SceneLightsGetClusters(struct SceneLights_t* sceneLights, Point* clusters);
01990
01992 AFTERWARP_API struct Buffer_t* SceneLightsGetIndices(struct SceneLights_t* sceneLights);
01993
01996 AFTERWARP_API LibraryBool SceneLightsExecute(struct SceneLights_t* sceneLights, Matrix const* view,
    Matrix const* projection);
01997
01998
02000 AFTERWARP_API LibraryBool SceneLightsRenderDebug(struct SceneLights_t* sceneLights, LibraryBool
    intensity);
02001
02002 // ObjectMaterials functions.
02003
02005 AFTERWARP_API struct ObjectMaterials_t* ObjectMaterialsCreate(void);
02006
02008 AFTERWARP_API void ObjectMaterialsDestroy(struct ObjectMaterials_t* objectMaterials);
02009
02011 AFTERWARP_API int32_t ObjectMaterialsGetCount(struct ObjectMaterials_t* objectMaterials);
02012
02014 AFTERWARP_API ObjectMaterial* ObjectMaterialsGetElement(struct ObjectMaterials_t* objectMaterials,
    int32_t index);
02015
02016
02018 AFTERWARP_API ObjectMaterial* ObjectMaterialsAdd(struct ObjectMaterials_t* objectMaterials);
02019
02021 AFTERWARP_API LibraryBool ObjectMaterialsErase(struct ObjectMaterials_t* objectMaterials, int32_t
    index);
02022
02024 AFTERWARP_API void ObjectMaterialsClear(struct ObjectMaterials_t* objectMaterials);
02025
02026 // TextureCabinet functions.
02027
02029 AFTERWARP_API struct TextureCabinet_t* TextureCabinetCreate(struct Device_t* device, Point const*
    size,
    TextureFidelity fidelity, int32_t samples, PixelFormat depthStencil);
02030
02031

```



```

02033 AFTERWARP_API void TextureCabinetDestroy(struct TextureCabinet_t* textureCabinet);
02034
02036 AFTERWARP_API struct Device_t* TextureCabinetGetDevice(struct TextureCabinet_t* textureCabinet);
02037
02039 AFTERWARP_API void TextureCabinetGetSize(struct TextureCabinet_t* textureCabinet, Point* size);
02040
02042 AFTERWARP_API LibraryBool TextureCabinetSetSize(struct TextureCabinet_t* textureCabinet, Point const*
size);
02043
02045 AFTERWARP_API int32_t TextureCabinetGetSamples(struct TextureCabinet_t* textureCabinet);
02046
02048 AFTERWARP_API PixelFormat TextureCabinetGetDepthStencil(struct TextureCabinet_t* textureCabinet);
02049
02051 AFTERWARP_API TextureFidelity TextureCabinetGetFidelity(struct TextureCabinet_t* textureCabinet);
02052
02054 AFTERWARP_API TextureCabinetAttributes TextureCabinetGetAttributes(struct TextureCabinet_t*
textureCabinet);
02055
02056 // Changes rendering attributes of the collection.
02057 AFTERWARP_API LibraryBool TextureCabinetSetAttributes(struct TextureCabinet_t* textureCabinet,
TextureCabinetAttributes attributes);
02058
02059
02061 AFTERWARP_API struct Texture_t* TextureCabinetGetTexture(struct TextureCabinet_t* textureCabinet,
TextureCabinetType type);
02062
02063
02065 AFTERWARP_API struct Texture_t* TextureCabinetGetFinalTexture(struct TextureCabinet_t*
textureCabinet);
02066
02069 AFTERWARP_API LibraryBool TextureCabinetClear(struct TextureCabinet_t* textureCabinet,
TextureCabinetPass pass, _In_opt_ FloatColor const* color);
02070
02071
02073 AFTERWARP_API LibraryBool TextureCabinetBegin(struct TextureCabinet_t* textureCabinet,
TextureCabinetPass pass);
02074
02075
02077 AFTERWARP_API void TextureCabinetEnd(struct TextureCabinet_t* textureCabinet);
02078
02080 AFTERWARP_API LibraryBool TextureCabinetRendering(struct TextureCabinet_t* textureCabinet);
02081
02091 AFTERWARP_API LibraryBool TextureCabinetFilter(struct TextureCabinet_t* textureCabinet,
TextureCabinetFilterType filter, Matrix const* projection);
02092
02093
02095 AFTERWARP_API LibraryBool TextureCabinetResolve(struct TextureCabinet_t* textureCabinet);
02096
02098 AFTERWARP_API LibraryBool TextureCabinetPresent(struct TextureCabinet_t* textureCabinet);
02099
02101 AFTERWARP_API void TextureCabinetGetToneMappingBloom(struct TextureCabinet_t* textureCabinet,
ToneMappingBloom* parameters);
02102
02103
02105 AFTERWARP_API void TextureCabinetSetToneMappingBloom(struct TextureCabinet_t* textureCabinet,
ToneMappingBloom const* parameters);
02106
02107
02109 AFTERWARP_API void TextureCabinetGetAmbientOcclusionParameters(struct TextureCabinet_t*
textureCabinet,
AmbientOcclusionParameters* parameters);
02110
02111
02113 AFTERWARP_API void TextureCabinetSetAmbientOcclusionParameters(struct TextureCabinet_t*
textureCabinet,
AmbientOcclusionParameters const* parameters);
02114
02115
02120 AFTERWARP_API LibraryBool TextureCabinetRetrieveGlassyMetrics(struct TextureCabinet_t* textureCabinet,
uint32_t* values);
02121
02122
02123 // ShadowCaster functions.
02124
02126 AFTERWARP_API struct Device_t* ShadowCasterGetDevice(struct ShadowCaster_t* shadowCaster);
02127
02129 AFTERWARP_API void ShadowCasterGetPosition(struct ShadowCaster_t* shadowCaster, Point* position);
02130
02132 AFTERWARP_API void ShadowCasterGetSize(struct ShadowCaster_t* shadowCaster, Point* size);
02133
02135 AFTERWARP_API void ShadowCasterGetViewProjection(struct ShadowCaster_t* shadowCaster,
Matrix* viewProjection);
02136
02137
02139 AFTERWARP_API void ShadowCasterSetViewProjection(struct ShadowCaster_t* shadowCaster,
Matrix const* viewProjection);
02140
02141
02143 AFTERWARP_API struct ShadowCastingAtlas_t* ShadowCasterGetAtlas(struct ShadowCaster_t* shadowCaster);
02144
02146 AFTERWARP_API LibraryBool ShadowCasterClear(struct ShadowCaster_t* shadowCaster);
02147
02149 AFTERWARP_API LibraryBool ShadowCasterBegin(struct ShadowCaster_t* shadowCaster);
02150
02152 AFTERWARP_API void ShadowCasterEnd(struct ShadowCaster_t* shadowCaster);
02153
02155 AFTERWARP_API LibraryBool ShadowCasterFilter(struct ShadowCaster_t* shadowCaster);
02156
02158 AFTERWARP_API LibraryBool ShadowCasterRendering(struct ShadowCaster_t* shadowCaster);

```

```

02159
02161 AFTERWARP_API struct Texture_t* ShadowCasterGetTexture(struct ShadowCaster_t* shadowCaster, int32_t
index);
02162
02163 // ShadowCastingAtlas functions.
02164
02166 AFTERWARP_API struct ShadowCastingAtlas_t* ShadowCastingAtlasCreate(struct Device_t* device,
02167     Point const* size, TechniqueShadows technique, int32_t samples, int32_t padding);
02168
02170 AFTERWARP_API void ShadowCastingAtlasDestroy(struct ShadowCastingAtlas_t* shadowCastingAtlas);
02171
02173 AFTERWARP_API struct Device_t* ShadowCastingAtlasGetDevice(struct ShadowCastingAtlas_t*
shadowCastingAtlas);
02174
02176 AFTERWARP_API TechniqueShadows ShadowCastingAtlasGetTechnique(
02177     struct ShadowCastingAtlas_t* shadowCastingAtlas);
02178
02180 AFTERWARP_API void ShadowCastingAtlasGetParameters(struct ShadowCastingAtlas_t* shadowCastingAtlas,
02181     ShadowParameters* parameters);
02182
02184 AFTERWARP_API LibraryBool ShadowCastingAtlasSetParameters(struct ShadowCastingAtlas_t*
shadowCastingAtlas,
02185     ShadowParameters const* parameters);
02186
02188 AFTERWARP_API void ShadowCastingAtlasGetSize(struct ShadowCastingAtlas_t* shadowCastingAtlas, Point*
size);
02189
02191 AFTERWARP_API int32_t ShadowCastingAtlasGetSamples(struct ShadowCastingAtlas_t* shadowCastingAtlas);
02192
02194 AFTERWARP_API int32_t ShadowCastingAtlasGetPadding(struct ShadowCastingAtlas_t* shadowCastingAtlas);
02195
02197 AFTERWARP_API struct Texture_t* ShadowCastingAtlasGetTexture(
02198     struct ShadowCastingAtlas_t* shadowCastingAtlas);
02199
02204 AFTERWARP_API struct ShadowCaster_t* ShadowCastingAtlasAdd(struct ShadowCastingAtlas_t*
shadowCastingAtlas,
02205     Point const* size, LibraryBool shared);
02206
02208 AFTERWARP_API void ShadowCastingAtlasErase(struct ShadowCastingAtlas_t* shadowCastingAtlas,
02209     struct ShadowCaster_t* caster);
02210
02212 AFTERWARP_API void ShadowCastingAtlasClear(struct ShadowCastingAtlas_t* shadowCastingAtlas);
02213
02215 AFTERWARP_API int32_t ShadowCastingAtlasGetCount(struct ShadowCastingAtlas_t* shadowCastingAtlas);
02216
02218 AFTERWARP_API struct ShadowCaster_t* ShadowCastingAtlasGetCaster(
02219     struct ShadowCastingAtlas_t* shadowCastingAtlas, int32_t index);
02220
02223 AFTERWARP_API LibraryBool ShadowCastingAtlasBorderFill(struct ShadowCastingAtlas_t*
shadowCastingAtlas);
02224
02225 // Scene functions.
02226
02228 AFTERWARP_API struct Scene_t* SceneCreateDepthsNormals(struct Device_t* device);
02229
02231 AFTERWARP_API struct Scene_t* SceneCreateModeling(struct Device_t* device);
02232
02234 AFTERWARP_API void SceneDestroy(struct Scene_t* scene);
02235
02237 AFTERWARP_API struct Device_t* SceneGetDevice(struct Scene_t* scene);
02238
02240 AFTERWARP_API int32_t SceneGetVertexElementCount(struct Scene_t* scene, int32_t index);
02241
02243 AFTERWARP_API VertexElement const* SceneGetVertexElements(struct Scene_t* scene, int32_t index);
02244
02246 AFTERWARP_API LibraryBool SceneSetVertexElements(struct Scene_t* scene, int32_t index,
02247     VertexElement const vertexElements[], int32_t vertexElementCount);
02248
02251 AFTERWARP_API LibraryBool SceneSetVertexElementsFromTextModeller(struct Scene_t* scene, int32_t
index);
02252
02254 AFTERWARP_API int32_t SceneGetActiveVertexElements(struct Scene_t* scene);
02255
02257 AFTERWARP_API LibraryBool SceneSetActiveVertexElements(struct Scene_t* scene, int32_t index);
02258
02260 AFTERWARP_API SceneAttributes SceneGetAttributes(struct Scene_t* scene);
02261
02263 AFTERWARP_API LibraryBool SceneSetAttributes(struct Scene_t* scene, SceneAttributes attributes);
02264
02266 AFTERWARP_API void SceneGetWorld(struct Scene_t* scene, Matrix* world);
02267
02269 AFTERWARP_API void SceneSetWorld(struct Scene_t* scene, Matrix const* world);
02270
02272 AFTERWARP_API void SceneGetView(struct Scene_t* scene, Matrix* view);
02273
02275 AFTERWARP_API void SceneSetView(struct Scene_t* scene, Matrix const* view);
02276

```

```

02278 AFTERWARP_API void SceneGetProjection(struct Scene_t* scene, Matrix* projection);
02279
02281 AFTERWARP_API void SceneSetProjection(struct Scene_t* scene, Matrix const* projection);
02282
02285 AFTERWARP_API LibraryBool SceneInstances(struct Scene_t* scene, Matrix const transforms[],
02286     FloatColor const colors[], int32_t count);
02287
02289 AFTERWARP_API int32_t SceneGetInstancesCount(struct Scene_t* scene);
02290
02292 AFTERWARP_API struct Program_t* SceneGetProgram(struct Scene_t* scene);
02293
02295 AFTERWARP_API LibraryBool SceneBegin(struct Scene_t* scene);
02296
02298 AFTERWARP_API void SceneEnd(struct Scene_t* scene);
02299
02301 AFTERWARP_API LibraryBool SceneRendering(struct Scene_t* scene);
02302
02304 AFTERWARP_API void SceneResetCache(struct Scene_t* scene);
02305
02306 // The following functions only work with "modeling" scene.
02307
02309 AFTERWARP_API void SceneGetMaterial(struct Scene_t* scene, ObjectMaterial* material);
02310
02312 AFTERWARP_API LibraryBool SceneSetMaterial(struct Scene_t* scene, ObjectMaterial const* material);
02313
02316 AFTERWARP_API void SceneGetToneMappingCoefficients(struct Scene_t* scene, Vector4* coefficients);
02317
02320 AFTERWARP_API LibraryBool SceneSetToneMappingCoefficients(struct Scene_t* scene, Vector4 const*
    coefficients);
02321
02323 AFTERWARP_API void SceneGetParallaxMappingParameters(struct Scene_t* scene,
02324     ParallaxMappingParameters* parameters);
02325
02327 AFTERWARP_API LibraryBool SceneSetParallaxMappingParameters(struct Scene_t* scene,
02328     ParallaxMappingParameters const* parameters);
02329
02331 AFTERWARP_API struct Sampler_t* SceneGetSampler(struct Scene_t* scene, SceneSamplerType type);
02332
02334 AFTERWARP_API struct Texture_t* SceneGetTexture(struct Scene_t* scene, SceneTextureType type);
02335
02337 AFTERWARP_API LibraryBool SceneSetTexture(struct Scene_t* scene, struct Texture_t* texture,
02338     SceneTextureType type);
02339
02341 AFTERWARP_API struct SceneLights_t* SceneGetLights(struct Scene_t* scene);
02342
02345 AFTERWARP_API void SceneSetLights(struct Scene_t* scene, struct SceneLights_t* sceneLights);
02346
02348 AFTERWARP_API struct ShadowCastingAtlas_t* SceneGetShadowCastingAtlas(struct Scene_t* scene);
02349
02352 AFTERWARP_API void SceneSetShadowCastingAtlas(struct Scene_t* scene,
02353     struct ShadowCastingAtlas_t* shadowCastingAtlas);
02354
02356 AFTERWARP_API struct TextureCabinet_t* SceneGetTextureCabinet(struct Scene_t* scene);
02357
02360 AFTERWARP_API void SceneSetTextureCabinet(struct Scene_t* scene,
02361     struct TextureCabinet_t* textureCabinet);
02362
02367 AFTERWARP_API LibraryBool ScenePrepare(struct Scene_t* scene);
02368
02369 // OceanSimulation functions.
02370
02372 AFTERWARP_API struct OceanSimulation_t* OceanSimulationCreate(struct Device_t* device);
02373
02375 AFTERWARP_API void OceanSimulationDestroy(struct OceanSimulation_t* oceanSimulation);
02376
02378 AFTERWARP_API struct Device_t* OceanSimulationGetDevice(struct OceanSimulation_t* oceanSimulation);
02379
02381 AFTERWARP_API void OceanSimulationGetWavesParameters(struct OceanSimulation_t* oceanSimulation,
02382     OceanWavesParameters* parameters);
02383
02385 AFTERWARP_API LibraryBool OceanSimulationSetWavesParameters(struct OceanSimulation_t* oceanSimulation,
02386     OceanWavesParameters const* parameters);
02387
02389 AFTERWARP_API struct Texture_t* OceanSimulationGetWavesTexture(struct OceanSimulation_t*
    oceanSimulation,
02390     int32_t index);
02391
02393 AFTERWARP_API void OceanSimulationGetSections(struct OceanSimulation_t* oceanSimulation, Point*
    sections);
02394
02396 AFTERWARP_API LibraryBool OceanSimulationSetSections(struct OceanSimulation_t* oceanSimulation,
02397     Point const* sections);
02398
02400 AFTERWARP_API void OceanSimulationGetView(struct OceanSimulation_t* oceanSimulation, Matrix* view);
02401
02403 AFTERWARP_API LibraryBool OceanSimulationSetView(struct OceanSimulation_t* oceanSimulation,
02404     Matrix const* view);

```

```

02405
02407 AFTERWARP_API void OceanSimulationGetProjection(struct OceanSimulation_t* oceanSimulation,
02408     Matrix* projection);
02409
02411 AFTERWARP_API LibraryBool OceanSimulationSetProjection(struct OceanSimulation_t* oceanSimulation,
02412     Matrix const* projection);
02413
02415 AFTERWARP_API void OceanSimulationGetScale(struct OceanSimulation_t* oceanSimulation, Vector* scale);
02416
02418 AFTERWARP_API LibraryBool OceanSimulationSetScale(struct OceanSimulation_t* oceanSimulation,
02419     Vector const* scale);
02420
02422 AFTERWARP_API void OceanSimulationGetPlane(struct OceanSimulation_t* oceanSimulation, Vector4* plane);
02423
02425 AFTERWARP_API LibraryBool OceanSimulationSetPlane(struct OceanSimulation_t* oceanSimulation,
02426     Vector4 const* plane);
02427
02429 AFTERWARP_API float OceanSimulationGetViewDistance(struct OceanSimulation_t* oceanSimulation);
02430
02432 AFTERWARP_API LibraryBool OceanSimulationSetViewDistance(struct OceanSimulation_t* oceanSimulation,
02433     float viewDistance);
02434
02436 AFTERWARP_API SceneAttributes OceanSimulationGetAttributes(struct OceanSimulation_t* oceanSimulation);
02437
02440 AFTERWARP_API void OceanSimulationSetAttributes(struct OceanSimulation_t* oceanSimulation,
02441     SceneAttributes attributes);
02442
02444 AFTERWARP_API void OceanSimulationGetMaterial(struct OceanSimulation_t* oceanSimulation,
02445     OceanMaterial* material);
02446
02448 AFTERWARP_API LibraryBool OceanSimulationSetMaterial(struct OceanSimulation_t* oceanSimulation,
02449     OceanMaterial const* material);
02450
02452 AFTERWARP_API struct Sampler_t* OceanSimulationGetSamplerShadow(struct OceanSimulation_t*
02453     oceanSimulation);
02455 AFTERWARP_API LibraryBool OceanSimulationUpdate(struct OceanSimulation_t* oceanSimulation, double
02456     latency);
02458 AFTERWARP_API LibraryBool OceanSimulationRender(struct OceanSimulation_t* oceanSimulation,
02459     struct Texture_t* linearDepths, struct SceneLights_t* sceneLights, struct ShadowCastingAtlas_t*
02460     atlas);
02461 // ObjectModel functions.
02462
02464 AFTERWARP_API struct ObjectModels_t* ObjectModelGetOwner(struct ObjectModel_t* objectModel);
02465
02467 AFTERWARP_API ObjectModelID ObjectModelGetID(struct ObjectModel_t* objectModel);
02468
02470 AFTERWARP_API void ObjectModelGetName(struct ObjectModel_t* objectModel, char* modelName,
02471     _Inout_ int32_t* modelNameLength);
02472
02474 AFTERWARP_API ObjectPayload ObjectModelGetPayload(struct ObjectModel_t* objectModel);
02475
02477 AFTERWARP_API LibraryBool ObjectModelSetName(struct ObjectModel_t* objectModel, char const* name);
02478
02480 AFTERWARP_API void ObjectModelGetDescription(struct ObjectModel_t* objectModel, char* description,
02481     _Inout_ int32_t* descriptionLength);
02482
02484 AFTERWARP_API LibraryBool ObjectModelSetDescription(struct ObjectModel_t* objectModel,
02485     char const* description);
02486
02488 AFTERWARP_API struct MeshVoxel_t* ObjectModelGetVoxel(struct ObjectModel_t* objectModel);
02489
02491 AFTERWARP_API void ObjectModelSetVoxel(struct ObjectModel_t* objectModel, struct MeshVoxel_t*
02492     meshVoxel);
02494 AFTERWARP_API struct SceneMesh_t* ObjectModelGetMesh(struct ObjectModel_t* objectModel);
02495
02497 AFTERWARP_API LibraryBool ObjectModelSetMesh(struct ObjectModel_t* objectModel,
02498     struct SceneMesh_t* sceneMesh);
02499
02501 AFTERWARP_API void ObjectModelGetMeshName(struct ObjectModel_t* objectModel, char* meshName,
02502     _Inout_ int32_t* meshNameLength);
02503
02505 AFTERWARP_API LibraryBool ObjectModelSetMeshName(struct ObjectModel_t* objectModel, char const*
02506     meshName);
02508 AFTERWARP_API struct ObjectModel_t* ObjectModelGetParent(struct ObjectModel_t* objectModel);
02509
02511 AFTERWARP_API LibraryBool ObjectModelSetParent(struct ObjectModel_t* objectModel,
02512     struct ObjectModel_t* parent);
02513
02515 AFTERWARP_API int32_t ObjectModelGetChildCount(struct ObjectModel_t* objectModel);
02516
02518 AFTERWARP_API struct ObjectModel_t* ObjectModelGetChild(struct ObjectModel_t* objectModel, int32_t
02519     index);

```

```

02519
02521 AFTERWARP_API void ObjectModelGetTransform(struct ObjectModel_t* objectModel, ModelTransform
transform,
02522     Matrix* matrix);
02523
02525 AFTERWARP_API LibraryBool ObjectModelSetTransform(struct ObjectModel_t* objectModel,
02526     ModelTransform transform, Matrix const* matrix);
02527
02529 AFTERWARP_API void ObjectModelGetPosition(struct ObjectModel_t* objectModel, Vector* position);
02530
02532 AFTERWARP_API float ObjectModelGetDepthBias(struct ObjectModel_t* objectModel);
02533
02535 AFTERWARP_API void ObjectModelSetDepthBias(struct ObjectModel_t* objectModel, float depthBias);
02536
02538 AFTERWARP_API void ObjectModelGetAABB(struct ObjectModel_t* objectModel, Vector* boxMin, Vector*
boxMax);
02539
02541 AFTERWARP_API uint32_t ObjectModelGetAttributes(struct ObjectModel_t* objectModel);
02542
02544 AFTERWARP_API void ObjectModelSetAttributes(struct ObjectModel_t* objectModel, uint32_t attributes);
02545
02547 AFTERWARP_API int32_t ObjectModelGetOrderIndex(struct ObjectModel_t* objectModel);
02548
02550 AFTERWARP_API void ObjectModelSetOrderIndex(struct ObjectModel_t* objectModel, int32_t orderIndex);
02551
02553 AFTERWARP_API uint64_t ObjectModelGetLayers(struct ObjectModel_t* objectModel);
02554
02557 AFTERWARP_API void ObjectModelSetLayers(struct ObjectModel_t* objectModel, uint64_t layers);
02558
02560 AFTERWARP_API void ObjectModelGetSize(struct ObjectModel_t* objectModel, Vector* size);
02561
02563 AFTERWARP_API void ObjectModelSetSize(struct ObjectModel_t* objectModel, Vector const* size);
02564
02566 AFTERWARP_API void ObjectModelGetAlignments(struct ObjectModel_t* objectModel, MeshAligns* aligns);
02567
02569 AFTERWARP_API void ObjectModelSetAlignments(struct ObjectModel_t* objectModel, MeshAligns const*
aligns);
02570
02572 AFTERWARP_API int32_t ObjectModelGetMaterial(struct ObjectModel_t* objectModel);
02573
02575 AFTERWARP_API void ObjectModelSetMaterial(struct ObjectModel_t* objectModel, int32_t material);
02576
02578 AFTERWARP_API void ObjectModelGetColor(struct ObjectModel_t* objectModel, FloatColor* color);
02579
02581 AFTERWARP_API void ObjectModelSetColor(struct ObjectModel_t* objectModel, FloatColor const* color);
02582
02584 AFTERWARP_API void ObjectModelGetHighlight(struct ObjectModel_t* objectModel, FloatColor* highlight);
02585
02587 AFTERWARP_API void ObjectModelSetHighlight(struct ObjectModel_t* objectModel, FloatColor const*
highlight);
02588
02590 AFTERWARP_API void ObjectModelConnectLatches(struct ObjectModel_t* objectModel, int32_t latchParent,
02591     int32_t latchLocal);
02592
02594 AFTERWARP_API void ObjectModelConnectLatchesByName(struct ObjectModel_t* objectModel,
02595     _In_opt_ char const* latchParentName, _In_opt_ char const* latchLocalName);
02596
02598 AFTERWARP_API void ObjectModelGetConnectedLatches(struct ObjectModel_t* objectModel, int32_t*
latchParent,
02599     int32_t* latchLocal);
02600
02602 AFTERWARP_API float ObjectModelGetWaypointDistance(struct ObjectModel_t* objectModel, int32_t group);
02603
02605 AFTERWARP_API void ObjectModelGetLatchWaypointCouple(struct ObjectModel_t* objectModel, int32_t group,
02606     float distance, Vector* position, Quaternion* orientation);
02607
02610 AFTERWARP_API void ObjectModelGetLatchWaypointCoupleMatrix(struct ObjectModel_t* objectModel, int32_t
group,
02611     float distance, Matrix* transform);
02612
02614 AFTERWARP_API void ObjectModelGetLatchTransform(struct ObjectModel_t* objectModel, Matrix* transform,
02615     int32_t latchIndex, LibraryBool local);
02616
02618 AFTERWARP_API void ObjectModelGetLatchTransformByName(struct ObjectModel_t* objectModel, Matrix*
transform,
02619     char const* name, LibraryBool local);
02620
02622 AFTERWARP_API void ObjectModelInvalidate(struct ObjectModel_t* objectModel);
02623
02626 AFTERWARP_API struct ObjectModel_t* ObjectModelGetNext(struct ObjectModel_t* objectModel);
02627
02628 // ObjectModels functions.
02629
02631 AFTERWARP_API struct ObjectModels_t* ObjectModelsCreate(struct SceneMeshes_t* sceneMeshes);
02632
02634 AFTERWARP_API void ObjectModelsDestroy(struct ObjectModels_t* objectModels);
02635

```

```

02637 AFTERWARP_API struct SceneMeshes_t* ObjectModelsGetMeshes(struct ObjectModels_t* objectModels);
02638
02640 AFTERWARP_API int32_t ObjectModelsGetObjectCount(struct ObjectModels_t* objectModels);
02641
02643 AFTERWARP_API struct ObjectModel_t* ObjectModelsGetFirst(struct ObjectModels_t* objectModels);
02644
02650 AFTERWARP_API struct ObjectModel_t* ObjectModelsAddWithID(struct ObjectModels_t* objectModels,
02651     ObjectModelID id, _In_opt_ char const* name, ObjectPayload payload, uint32_t attributes);
02652
02654 AFTERWARP_API struct ObjectModel_t* ObjectModelsAdd(struct ObjectModels_t* objectModels,
02655     _In_opt_ char const* name, ObjectPayload payload, uint32_t attributes);
02656
02658 AFTERWARP_API void ObjectModelsErase(struct ObjectModels_t* objectModels,
02659     struct ObjectModel_t* objectModel);
02660
02662 AFTERWARP_API void ObjectModelsClear(struct ObjectModels_t* objectModels);
02663
02665 AFTERWARP_API struct ObjectModel_t* ObjectModelsGetObjectByID(struct ObjectModels_t* objectModels,
02666     ObjectModelID id);
02667
02669 AFTERWARP_API struct ObjectModel_t* ObjectModelsGetObjectByName(struct ObjectModels_t* objectModels,
02670     char const* name);
02671
02673 AFTERWARP_API struct ObjectModel_t* ObjectModelsPayload(struct ObjectModels_t* objectModels,
02674     ObjectPayload payload);
02675
02677 AFTERWARP_API struct ObjectModelView_t* ObjectModelsCreateView(struct ObjectModels_t* objectModels);
02678
02680 AFTERWARP_API void ObjectModelsEraseView(struct ObjectModels_t* objectModels,
02681     struct ObjectModelView_t* objectModelView);
02682
02684 AFTERWARP_API void ObjectModelsClearViews(struct ObjectModels_t* objectModels);
02685
02686 // ObjectModelView functions.
02687
02689 AFTERWARP_API struct ObjectModels_t* ObjectModelViewGetOwner(struct ObjectModelView_t*
    objectModelView);
02690
02692 AFTERWARP_API void ObjectModelViewGetView(struct ObjectModelView_t* objectModelView, Matrix* view);
02693
02695 AFTERWARP_API void ObjectModelViewSetView(struct ObjectModelView_t* objectModelView, Matrix const*
    view);
02696
02698 AFTERWARP_API void ObjectModelViewGetProjection(struct ObjectModelView_t* objectModelView,
02699     Matrix* projection, LibraryBool* depthClipNegative);
02700
02702 AFTERWARP_API void ObjectModelViewSetProjection(struct ObjectModelView_t* objectModelView,
02703     Matrix const* projection, LibraryBool depthClipNegative);
02704
02706 AFTERWARP_API void ObjectModelViewGetViewProjection(struct ObjectModelView_t* objectModelView,
02707     Matrix* viewProjection);
02708
02710 AFTERWARP_API uint64_t ObjectModelViewGetLayers(struct ObjectModelView_t* objectModelView);
02711
02713 AFTERWARP_API void ObjectModelViewSetLayers(struct ObjectModelView_t* objectModelView, uint64_t
    layers);
02714
02716 AFTERWARP_API void ObjectModelViewUpdateNeeded(struct ObjectModelView_t* objectModelView);
02717
02719 AFTERWARP_API void ObjectModelViewInvalidate(struct ObjectModelView_t* objectModelView);
02720
02722 AFTERWARP_API int32_t ObjectModelViewGetObjectCount(struct ObjectModelView_t* objectModelView);
02723
02725 AFTERWARP_API struct ObjectModel_t* ObjectModelViewGetObject(struct ObjectModelView_t*
    objectModelView,
02726     int32_t index);
02727
02730 AFTERWARP_API LibraryBool ObjectModelViewUpdate(struct ObjectModelView_t* objectModelView);
02731
02733 AFTERWARP_API void ObjectModelViewSort(struct ObjectModelView_t* objectModelView,
02734     ObjectModelViewCompare compare);
02735
02737 AFTERWARP_API void ObjectModelViewSortWith(struct ObjectModelView_t* objectModelView,
02738     ObjectModelCompareFunc compareFunc, void* user);
02739
02743 AFTERWARP_API struct ObjectModel_t* ObjectModelViewSelect(struct ObjectModelView_t* objectModelView,
02744     Vector const* origin, Vector const* direction, float* distance);
02745
02749 AFTERWARP_API struct ObjectModel_t* ObjectModelViewSelectAny(struct ObjectModelView_t*
    objectModelView,
02750     Vector const* origin, Vector const* direction, float* distance);
02751
02754 AFTERWARP_API int32_t ObjectModelViewGetObjectsNotCulled(struct ObjectModelView_t* objectModelView);
02755
02757 AFTERWARP_API int32_t ObjectModelViewGetIntersectedRays(struct ObjectModelView_t* objectModelView);
02758
02760 AFTERWARP_API int32_t ObjectModelViewGetIntersectedObjects(struct ObjectModelView_t* objectModelView);

```



```

02761
02762 AFTERWARP_API LibraryBool ObjectModelViewAutoDraw(struct ObjectModelView_t* objectModelView,
02763     struct Scene_t* scene, struct ObjectMaterials_t* materials, uint32_t options,
02764     _Inout_ int32_t* drawCalls);
02765
02766 // MeshVoxel functions.
02767
02768 AFTERWARP_API struct MeshVoxel_t* MeshVoxelCreate(void);
02769
02770 AFTERWARP_API void MeshVoxelDestroy(struct MeshVoxel_t* meshVoxel);
02771
02772 AFTERWARP_API void MeshVoxelTakeAway(struct MeshVoxel_t* meshVoxel, struct MeshVoxel_t*
02773     meshVoxelAnother);
02774
02775 AFTERWARP_API LibraryBool MeshVoxelLoadFromFile(struct MeshVoxel_t* meshVoxel, char const* fileName,
02776     int32_t dimensions[], uint8_t* levels, LibraryBool* colors);
02777
02778 AFTERWARP_API LibraryBool MeshVoxelLoadFromFileInMemory(struct MeshVoxel_t* meshVoxel, void* buffer,
02779     uint32_t bufferSize, int32_t dimensions[], uint8_t* levels, LibraryBool* colors);
02780
02781 AFTERWARP_API LibraryBool MeshVoxelSaveToFile(struct MeshVoxel_t* meshVoxel, char const* fileName,
02782     int32_t const dimensions[], uint8_t levels, LibraryBool colors);
02783
02784 AFTERWARP_API void MeshVoxelExtents(struct MeshVoxel_t* meshVoxel, Vector* position, Vector* size);
02785
02786 AFTERWARP_API void MeshVoxelComputeParameters(struct MeshVoxel_t* meshVoxel, int32_t dimensions[],
02787     uint8_t* levels, LibraryBool* colors);
02788
02789 AFTERWARP_API LibraryBool MeshVoxelVisualize(struct MeshVoxel_t* meshVoxel, uint8_t levelMax,
02790     MeshVoxelVisualizeFunc visualizeFunc, void* user);
02791
02792 AFTERWARP_API LibraryBool MeshVoxelIntersect(struct MeshVoxel_t* meshVoxel, Vector const* origin,
02793     Vector const* direction, Matrix const* world, Matrix const* view, float* distance, int32_t*
02794     testsCount);
02795
02796 // SceneMeshMaterial functions.
02797
02798 AFTERWARP_API void SceneMeshMaterialGetName(struct SceneMeshMaterial_t* material, char* name,
02799     _Inout_ int32_t* nameLength);
02800
02801 AFTERWARP_API void SceneMeshMaterialGetShading(struct SceneMeshMaterial_t* material,
02802     SceneMeshMaterialShading* shading);
02803
02804 AFTERWARP_API void SceneMeshMaterialSetShading(struct SceneMeshMaterial_t* material,
02805     SceneMeshMaterialShading const* shading);
02806
02807 AFTERWARP_API struct Texture_t* SceneMeshMaterialGetTexture(struct SceneMeshMaterial_t* material,
02808     uint8_t type);
02809
02810 AFTERWARP_API LibraryBool SceneMeshMaterialSetTexture(struct SceneMeshMaterial_t* material,
02811     struct Texture_t* texture, uint8_t type);
02812
02813 AFTERWARP_API void SceneMeshMaterialReleaseTextures(struct SceneMeshMaterial_t* material);
02814
02815 AFTERWARP_API int32_t SceneMeshMaterialGetRangeCount(struct SceneMeshMaterial_t* material);
02816
02817 AFTERWARP_API LibraryBool SceneMeshMaterialGetRange(struct SceneMeshMaterial_t* material,
02818     int32_t index, _Out_ SceneMeshMaterialRange* range);
02819
02820 AFTERWARP_API LibraryBool SceneMeshMaterialSetRange(struct SceneMeshMaterial_t* material,
02821     int32_t index, SceneMeshMaterialRange const* range);
02822
02823 AFTERWARP_API int32_t SceneMeshMaterialAddRange(struct SceneMeshMaterial_t* material,
02824     SceneMeshMaterialRange const* range);
02825
02826 AFTERWARP_API void SceneMeshMaterialClearRanges(struct SceneMeshMaterial_t* material);
02827
02828 AFTERWARP_API LibraryBool SceneMeshMaterialCommit(struct SceneMeshMaterial_t* material,
02829     struct Device_t* device, TextureAttributes attributes);
02830
02831 AFTERWARP_API LibraryBool SceneMeshMaterialCopy(struct SceneMeshMaterial_t* material,
02832     struct SceneMeshMaterial_t* source);
02833
02834 // SceneMeshMaterials functions.
02835
02836 AFTERWARP_API struct SceneMeshMaterials_t* SceneMeshMaterialsCreate(void);
02837
02838 AFTERWARP_API void SceneMeshMaterialsDestroy(struct SceneMeshMaterials_t* materials);
02839
02840 AFTERWARP_API LibraryBool SceneMeshMaterialsSetName(struct SceneMeshMaterials_t* materials,
02841     char const* name);
02842
02843 AFTERWARP_API void SceneMeshMaterialsGetName(struct SceneMeshMaterials_t* materials,
02844     char* name, _Inout_ int32_t* nameLength);
02845
02846 AFTERWARP_API int32_t SceneMeshMaterialsGetCount(struct SceneMeshMaterials_t* materials);
02847

```

```

02886 AFTERWARD_API struct SceneMeshMaterial_t* SceneMeshMaterialsGetMaterial(
02887     struct SceneMeshMaterials_t* materials, int32_t index);
02888
02891 AFTERWARD_API int32_t SceneMeshMaterialsAdd(struct SceneMeshMaterials_t* materials, char const* name);
02892
02894 AFTERWARD_API void SceneMeshMaterialsErase(struct SceneMeshMaterials_t* materials, int32_t index);
02895
02897 AFTERWARD_API void SceneMeshMaterialsClear(struct SceneMeshMaterials_t* materials);
02898
02900 AFTERWARD_API LibraryBool SceneMeshMaterialsCommit(struct SceneMeshMaterials_t* materials,
02901     struct Device_t* device, TextureAttributes attributes);
02902
02904 AFTERWARD_API LibraryBool SceneMeshMaterialsCopy(struct SceneMeshMaterials_t* materials,
02905     struct SceneMeshMaterials_t* materialsAnother);
02906
02908 AFTERWARD_API void SceneMeshMaterialsTakeAway(struct SceneMeshMaterials_t* materials,
02909     struct SceneMeshMaterials_t* materialsAnother);
02910
02912 AFTERWARD_API LibraryBool SceneMeshMaterialsTexturing(struct SceneMeshMaterials_t* materials);
02913
02914 // SceneMeshLatches functions.
02915
02917 AFTERWARD_API struct SceneMeshLatches_t* SceneMeshLatchesCreate(void);
02918
02920 AFTERWARD_API void SceneMeshLatchesDestroy(struct SceneMeshLatches_t* latches);
02921
02923 AFTERWARD_API int32_t SceneMeshLatchesGetCount(struct SceneMeshLatches_t* latches);
02924
02926 AFTERWARD_API LibraryBool SceneMeshLatchesGetLatch(struct SceneMeshLatches_t* latches, int32_t index,
02927     SceneMeshLatch* latch);
02928
02930 AFTERWARD_API LibraryBool SceneMeshLatchesSetLatch(struct SceneMeshLatches_t* latches, int32_t index,
02931     SceneMeshLatch const* latch);
02932
02934 AFTERWARD_API int32_t SceneMeshLatchesGetLatchIndex(struct SceneMeshLatches_t* latches, char const*
    name);
02935
02938 AFTERWARD_API int32_t SceneMeshLatchesAdd(struct SceneMeshLatches_t* latches, _In_opt_ char const*
    name,
02939     int32_t type, int32_t group, _In_opt_ Vector const* position, _In_opt_ Quaternion const*
    orientation);
02940
02942 AFTERWARD_API void SceneMeshLatchesErase(struct SceneMeshLatches_t* latches, int32_t index);
02943
02945 AFTERWARD_API void SceneMeshLatchesClear(struct SceneMeshLatches_t* latches);
02946
02948 AFTERWARD_API LibraryBool SceneMeshLatchesCopy(struct SceneMeshLatches_t* latches,
02949     struct SceneMeshLatches_t* latchesAnother);
02950
02952 AFTERWARD_API void SceneMeshLatchesTakeAway(struct SceneMeshLatches_t* latches,
02953     struct SceneMeshLatches_t* latchesAnother);
02954
02956 AFTERWARD_API LibraryBool SceneMeshLatchesLoadFromFile(struct SceneMeshLatches_t* latches,
02957     char const* fileName);
02958
02960 AFTERWARD_API LibraryBool SceneMeshLatchesLoadFromFileInMemory(struct SceneMeshLatches_t* latches,
02961     void* buffer, uint32_t bufferSize);
02962
02964 AFTERWARD_API LibraryBool SceneMeshLatchesSaveToFile(struct SceneMeshLatches_t* latches,
02965     char const* fileName);
02966
02968 AFTERWARD_API float SceneMeshLatchesGetWaypointDistance(struct SceneMeshLatches_t* latches,
02969     int32_t group);
02970
02972 AFTERWARD_API void SceneMeshLatchesGetWaypointCouple(struct SceneMeshLatches_t* latches,
02973     int32_t group, float distance, Vector* position, Quaternion* orientation);
02974
02977 AFTERWARD_API void SceneMeshLatchesInvalidateWaypoints(struct SceneMeshLatches_t* latches);
02978
02979 // SceneMesh functions.
02980
02982 AFTERWARD_API void SceneMeshGetName(struct SceneMesh_t* sceneMesh, char* meshName,
02983     _Inout_ int32_t* meshNameLength);
02984
02986 AFTERWARD_API ObjectPayload SceneMeshGetPayload(struct SceneMesh_t* sceneMesh);
02987
02989 AFTERWARD_API struct MeshModel_t* SceneMeshGetModel(struct SceneMesh_t* sceneMesh);
02990
02992 AFTERWARD_API struct MeshVoxel_t* SceneMeshGetVoxel(struct SceneMesh_t* sceneMesh);
02993
02995 AFTERWARD_API struct SceneMeshMaterials_t* SceneMeshGetMaterials(struct SceneMesh_t* sceneMesh);
02996
02998 AFTERWARD_API struct MeshMetaTags_t* SceneMeshGetTags(struct SceneMesh_t* sceneMesh);
02999
03001 AFTERWARD_API struct SceneMeshLatches_t* SceneMeshGetLatches(struct SceneMesh_t* sceneMesh);
03002
03004 AFTERWARD_API void SceneMeshGetBounds(struct SceneMesh_t* sceneMesh, Vector* boundsMin, Vector*

```



```

        boundsMax);
03005
03007 AFTERWARP_API float SceneMeshGetScale(struct SceneMesh_t* sceneMesh);
03008
03010 AFTERWARP_API void SceneMeshGetSize(struct SceneMesh_t* sceneMesh, Vector* size);
03011
03014 AFTERWARP_API uint8_t SceneMeshGetVertexElementsIndex(struct SceneMesh_t* sceneMesh);
03015
03018 AFTERWARP_API void SceneMeshSetVertexElementsIndex(struct SceneMesh_t* sceneMesh,
03019     uint8_t vertexElementsIndex);
03020
03022 AFTERWARP_API void SceneMeshSetSlice(struct SceneMesh_t* sceneMesh, struct SceneMesh_t* parent,
03023     struct MeshMetaTag_t* parentTag);
03024
03027 AFTERWARP_API void SceneMeshGetSlice(struct SceneMesh_t* sceneMesh, _Out_ struct SceneMesh_t** parent,
03028     _Out_ struct MeshMetaTag_t** parentTag);
03029
03035 AFTERWARP_API LibraryBool SceneMeshAutoDraw(struct SceneMesh_t* sceneMesh, struct Scene_t* scene,
03036     ObjectMaterial const* material, FloatColor const* color, PrimitiveTopology topology, int32_t
03037     instanceCount,
03037     int32_t elementCount, int32_t firstIndex, int32_t baseVertex, uint32_t options,
03038     _Inout_ int32_t* drawCalls);
03039
03046 AFTERWARP_API LibraryBool SceneMeshAutoDrawSliced(struct SceneMesh_t* sceneMesh, struct Scene_t*
03047     scene,
03047     ObjectMaterial const* material, FloatColor const* color, PrimitiveTopology topology, int32_t
03048     instanceCount,
03048     uint32_t options, _Inout_ int32_t* drawCalls);
03049
03050 // SceneMeshes functions.
03051
03053 AFTERWARP_API struct SceneMeshes_t* SceneMeshesCreate(struct Device_t* device);
03054
03056 AFTERWARP_API void SceneMeshesDestroy(struct SceneMeshes_t* sceneMeshes);
03057
03059 AFTERWARP_API struct SceneMesh_t* SceneMeshesAdd(struct SceneMeshes_t* sceneMeshes, char const* name,
03060     ObjectPayload payload, Vector const* boundsMin, Vector const* boundsMax, float scale);
03061
03063 AFTERWARP_API struct SceneMesh_t* SceneMeshesAddFromBuffer(struct SceneMeshes_t* sceneMeshes,
03064     char const* name, struct MeshBuffer_t* meshBuffer, ObjectPayload payload,
03065     VertexElement const vertexElements[], int32_t vertexElementCount, _In_opt_ Vector const* boundsMin,
03066     _In_opt_ Vector const* boundsMax, uint32_t channel, uint32_t semanticIndex, float scale);
03067
03072 AFTERWARP_API struct SceneMesh_t* SceneMeshesAddFromFile(struct SceneMeshes_t* sceneMeshes,
03073     char const* name, char const* fileName, ObjectPayload payload, VertexElement const vertexElements[],
03074     int32_t vertexElementCount, uint32_t channel, uint32_t semanticIndex, _Inout_ uint32_t* options,
03075     float scale, MeshLoadSaveFeedback feedback, void* feedbackUser, char* debug, _Inout_ int32_t*
03076     debugLength);
03076
03080 AFTERWARP_API LibraryBool SceneMeshesSlice(struct SceneMeshes_t* sceneMeshes, struct SceneMesh_t*
03081     parent,
03081     uint8_t type, char const* prefix);
03082
03084 AFTERWARP_API struct Device_t* SceneMeshesGetDevice(struct SceneMeshes_t* sceneMeshes);
03085
03087 AFTERWARP_API int32_t SceneMeshesGetCount(struct SceneMeshes_t* sceneMeshes);
03088
03091 AFTERWARP_API struct SceneMesh_t* SceneMeshesGetMeshByIndex(struct SceneMeshes_t* sceneMeshes, int32_t
03092     index);
03092
03094 AFTERWARP_API struct SceneMesh_t* SceneMeshesGetMeshByName(struct SceneMeshes_t* sceneMeshes,
03095     char const* name);
03096
03098 AFTERWARP_API struct SceneMesh_t* SceneMeshesPayload(struct SceneMeshes_t* sceneMeshes,
03099     ObjectPayload payload);
03100
03102 AFTERWARP_API void SceneMeshesErase(struct SceneMeshes_t* sceneMeshes, struct SceneMesh_t* sceneMesh);
03103
03105 AFTERWARP_API void SceneMeshesClear(struct SceneMeshes_t* sceneMeshes);
03106
03107 // Application functions.
03108
03110 AFTERWARP_API struct Application_t* ApplicationCreate(_Inout_ ApplicationConfiguration*
03111     configuration);
03111
03113 AFTERWARP_API void ApplicationDestroy(struct Application_t* application);
03114
03117 AFTERWARP_API void ApplicationSetEvents(struct Application_t* application, ApplicationEvents const*
03118     events,
03118     void* user);
03119
03121 AFTERWARP_API UntypedHandle ApplicationGetWindowHandle(struct Application_t* application);
03122
03124 AFTERWARP_API LibraryBool ApplicationSetTitle(struct Application_t* application, char const* title);
03125
03129 AFTERWARP_API void ApplicationGetTitle(struct Application_t* application, char* title,
03130     _Inout_ int32_t* titleLength);

```

```

03131
03133 AFTERWARP_API LibraryBool ApplicationSetIconTitle(struct Application_t* application, char const*
iconTitle);
03134
03138 AFTERWARP_API void ApplicationGetIconTitle(struct Application_t* application, char* iconTitle,
03139 _Inout_ int32_t* iconTitleLength);
03140
03144 AFTERWARP_API void ApplicationGetExecutablePath(struct Application_t* application, char* path,
03145 _Inout_ int32_t* pathLength);
03146
03148 AFTERWARP_API void ApplicationGetWindowRect(struct Application_t* application, Rect* rect);
03149
03151 AFTERWARP_API void ApplicationSetWindowRect(struct Application_t* application, Rect const* rect);
03152
03154 AFTERWARP_API void ApplicationGetClientRect(struct Application_t* application, Rect* rect);
03155
03157 AFTERWARP_API void ApplicationSetClientSize(struct Application_t* application, Point const* size);
03158
03160 AFTERWARP_API void ApplicationGetMinimalSize(struct Application_t* application, Point* size);
03161
03163 AFTERWARP_API void ApplicationSetMinimalSize(struct Application_t* application, Point const* size);
03164
03166 AFTERWARP_API double ApplicationGetWindowScale(struct Application_t* application);
03167
03169 AFTERWARP_API void ApplicationInvalidate(struct Application_t* application);
03170
03172 AFTERWARP_API Key ApplicationTranslateVirtualKey(struct Application_t* application, int32_t
virtualKey);
03173
03175 AFTERWARP_API int32_t ApplicationConvertPortableKey(struct Application_t* application, Key key);
03176
03179 AFTERWARP_API LibraryBool ApplicationExecute(struct Application_t* application);
03180
03182 AFTERWARP_API void ApplicationTerminate(struct Application_t* application);
03183
03185 AFTERWARP_API ApplicationWindowState ApplicationGetWindowState(struct Application_t* application);
03186
03188 AFTERWARP_API LibraryBool ApplicationSetWindowState(struct Application_t* application,
03189 ApplicationWindowState windowState);
03190
03194 AFTERWARP_API LibraryBool ApplicationReadTextFromClipboard(struct Application_t* application,
03195 char* text, _Inout_ int32_t* textSize);
03196
03198 AFTERWARP_API LibraryBool ApplicationWriteTextToClipboard(struct Application_t* application,
03199 char const* text);
03200
03202 AFTERWARP_API AppCursor ApplicationGetCursor(struct Application_t* application);
03203
03205 AFTERWARP_API LibraryBool ApplicationSetCursor(struct Application_t* application, AppCursor cursor);
03206
03208 AFTERWARP_API LibraryBool ApplicationCaptureMouseInput(struct Application_t* application);
03209
03211 AFTERWARP_API LibraryBool ApplicationReleaseMouseInput(struct Application_t* application);
03212
03214 AFTERWARP_API LibraryBool ApplicationMouseInputCaptured(struct Application_t* application);
03215
03218 AFTERWARP_API LibraryBool ApplicationSetIcons(struct Application_t* application,
03219 struct Surface_t* const* surfaces, int32_t count);
03220
03223 AFTERWARP_API LibraryBool ApplicationSetIconsFromFiles(struct Application_t* application,
03224 char const* const* fileNames, int32_t fileNameCount);
03225
03230 AFTERWARP_API LibraryBool ApplicationFileChooserDialog(struct Application_t* application,
03231 _Inout_ char* filePath, _Inout_ int32_t* filePathLength, FileChooserDialog dialog, char const*
filters,
03232 char const* caption, char const* accept, char const* reject);
03233
03234 // ActorCamera functions.
03235
03238 AFTERWARP_API struct ActorCamera_t* ActorCameraCreate(Vector const* position, Vector const* rotation,
03239 float distance);
03240
03242 AFTERWARP_API void ActorCameraDestroy(struct ActorCamera_t* camera);
03243
03245 AFTERWARP_API void ActorCameraGetForward(struct ActorCamera_t* camera, Vector* forward);
03246
03248 AFTERWARP_API void ActorCameraGetRight(struct ActorCamera_t* camera, Vector* right);
03249
03251 AFTERWARP_API void ActorCameraGetCeiling(struct ActorCamera_t* camera, Vector* ceiling);
03252
03254 AFTERWARP_API void ActorCameraGetPosition(struct ActorCamera_t* camera, Vector* position);
03255
03257 AFTERWARP_API LibraryBool ActorCameraSetPosition(struct ActorCamera_t* camera, Vector const*
position);
03258
03260 AFTERWARP_API void ActorCameraGetRotation(struct ActorCamera_t* camera, Vector4* rotation);
03261

```

```

03264 AFTERWARP_API LibraryBool ActorCameraSetRotation(struct ActorCamera_t* camera, Vector4 const*
rotation);
03265
03267 AFTERWARP_API float ActorCameraGetDistance(struct ActorCamera_t* camera);
03268
03270 AFTERWARP_API void ActorCameraSetDistance(struct ActorCamera_t* camera, float distance);
03271
03273 AFTERWARP_API void ActorCameraGetQuaternion(struct ActorCamera_t* camera, Quaternion* quaternion);
03274
03276 AFTERWARP_API LibraryBool ActorCameraSetQuaternion(struct ActorCamera_t* camera,
Quaternion const* quaternion);
03277
03278
03280 AFTERWARP_API void ActorCameraGetView(struct ActorCamera_t* camera, Matrix* view);
03281
03283 AFTERWARP_API void ActorCameraZoom(struct ActorCamera_t* camera, PointF const* position,
PointF const* size, Matrix const* projection, float delta);
03284
03285
03287 AFTERWARP_API void ActorCameraZoomOrtho(struct ActorCamera_t* camera, PointF const* position,
PointF const* size, float distance, float adjustedDistance, Matrix const* projection,
Matrix const* adjustedProjection);
03288
03289
03292 AFTERWARP_API void ActorCameraGetConstraints(struct ActorCamera_t* camera, CameraConstraints*
constraints);
03293
03295 AFTERWARP_API void ActorCameraSetConstraints(struct ActorCamera_t* camera,
CameraConstraints const* constraints);
03296
03297
03302 AFTERWARP_API LibraryBool ActorCameraCommand(struct ActorCamera_t* camera, CameraCommand command,
Vector const* position, PointF const* size, Vector const* sense, Matrix const* projection);
03303
03304
03305 // Timer functions.
03306
03309 AFTERWARP_API struct Timer_t* TimerCreate(void);
03310
03312 AFTERWARP_API void TimerDestroy(struct Timer_t* timer);
03313
03315 AFTERWARP_API double TimerGetSpeed(struct Timer_t* timer);
03316
03319 AFTERWARP_API void TimerSetSpeed(struct Timer_t* timer, double speed);
03320
03323 AFTERWARP_API LibraryBool TimerNextSlice(struct Timer_t* timer);
03324
03328 AFTERWARP_API LibraryBool TimerUpdateNextSlice(struct Timer_t* timer);
03329
03331 AFTERWARP_API void TimerUpdate(struct Timer_t* timer);
03332
03335 AFTERWARP_API void TimerReset(struct Timer_t* timer);
03336
03340 AFTERWARP_API float TimerGetFrameRate(struct Timer_t* timer);
03341
03345 AFTERWARP_API uint64_t TimerGetLatency(struct Timer_t* timer);
03346
03348 AFTERWARP_API int64_t TimerGetTimeSlice(struct Timer_t* timer);
03349
03351 AFTERWARP_API int64_t TimerGetSkippedTimeSlices(struct Timer_t* timer);
03352
03355 AFTERWARP_API void TimerTrimSkippedTimeSlices(struct Timer_t* timer, int64_t slicesToTrim);
03356
03358 AFTERWARP_API int32_t TimerExtractTokens(struct Timer_t* timer);
03359
03360 // Widget functions.
03361
03366 AFTERWARP_API struct Widget_t* WidgetCreate(struct Widget_t* manager, char const* className,
char const* instanceName, ObjectPayload payload);
03367
03368
03371 AFTERWARP_API struct Widget_t* WidgetManagerCreate(struct Application_t* application,
struct TextRenderer_t* textRenderer, char const* instanceName, ObjectPayload payload, PixelFormat
format,
int32_t multisamples, uint32_t attributes);
03372
03373
03374
03376 AFTERWARP_API void WidgetDestroy(struct Widget_t* widget);
03377
03379 AFTERWARP_API struct Widget_t* WidgetGetManager(struct Widget_t* widget);
03380
03382 AFTERWARP_API ObjectPayload WidgetGetPayload(struct Widget_t* widget);
03383
03385 AFTERWARP_API struct Widget_t* WidgetGetParent(struct Widget_t* widget);
03386
03389 AFTERWARP_API LibraryBool WidgetSetParent(struct Widget_t* widget, struct Widget_t* parent);
03390
03392 AFTERWARP_API void WidgetLocalToScreen(struct Widget_t* widget, PointF* screenPos,
PointF const* localPos);
03393
03394
03396 AFTERWARP_API void WidgetScreenToLocal(struct Widget_t* widget, PointF* localPos,
PointF const* screenPos);
03397
03398
03400 AFTERWARP_API void WidgetBringToFront(struct Widget_t* widget);

```

```

03401
03403 AFTERWARP_API void WidgetSendToBack(struct Widget_t* widget);
03404
03406 AFTERWARP_API void WidgetInvalidate(struct Widget_t* widget);
03407
03409 AFTERWARP_API void WidgetAccomodate(struct Widget_t* widget);
03410
03414 AFTERWARP_API LibraryBool WidgetUpdate(struct Widget_t* widget, double latency);
03415
03417 AFTERWARP_API void WidgetAcceptMouse(struct Widget_t* widget, MouseEvent event, MouseButton button,
03418     PointF const* position);
03419
03421 AFTERWARP_API void WidgetAcceptKey(struct Widget_t* widget, KeyEvent event, Key key,
03422     UnicodeChar charCode);
03423
03425 AFTERWARP_API int32_t WidgetGetChildCount(struct Widget_t* widget);
03426
03428 AFTERWARP_API struct Widget_t* WidgetGetChildByIndex(struct Widget_t* widget, int32_t index);
03429
03432 AFTERWARP_API int32_t WidgetGetChildIndex(struct Widget_t* widget, struct Widget_t* child);
03433
03436 AFTERWARP_API struct Widget_t* WidgetFindAt(struct Widget_t* widget, PointF const* position);
03437
03439 AFTERWARP_API LibraryBool WidgetInvokeEvent(struct Widget_t* widget, char const* eventName);
03440
03442 AFTERWARP_API void WidgetSetExternalEvent(struct Widget_t* widget, WidgetExternalEvent event, void*
03443     user);
03444
03445 AFTERWARP_API void WidgetGetExternalEvent(struct Widget_t* widget, WidgetExternalEvent* event, void**
03446     user);
03447
03449 AFTERWARP_API LibraryBool WidgetSetProperty(struct Widget_t* widget, char const* propertyName,
03450     WidgetPropertyType valueType, void const* value, LibraryBool invokeChanged);
03451
03454 AFTERWARP_API LibraryBool WidgetGetProperty(struct Widget_t* widget, char const* propertyName,
03455     WidgetPropertyType valueType, void* value);
03456
03460 AFTERWARP_API LibraryBool WidgetGetPropertyAsString(struct Widget_t* widget, char const* propertyName,
03461     char* value, _Inout_ int32_t* valueLength);
03462
03464 AFTERWARP_API int32_t WidgetGetPropertyCount(struct Widget_t* widget);
03465
03475 AFTERWARP_API LibraryBool WidgetPropertyIdentify(struct Widget_t* widget, _Inout_ WidgetProperty*
03476     property);
03477 // Functions that can only be called on a widget manager.
03478
03480 AFTERWARP_API struct Application_t* WidgetManagerGetApplication(struct Widget_t* widget);
03481
03483 AFTERWARP_API struct TextRenderer_t* WidgetManagerGetTextRenderer(struct Widget_t* widget);
03484
03486 AFTERWARP_API struct Widget_t* WidgetManagerGetWidgetByName(struct Widget_t* widget, char const*
03487     name);
03488
03489 AFTERWARP_API struct Widget_t* WidgetManagerGetWidgetByPayload(struct Widget_t* widget, ObjectPayload
03490     payload);
03491
03492 AFTERWARP_API LibraryBool WidgetManagerPresent(struct Widget_t* widget);
03493
03495 AFTERWARP_API struct Texture_t* WidgetManagerGetTexture(struct Widget_t* widget,
03496     WidgetManagerTextureType textureType);
03497
03499 AFTERWARP_API int32_t WidgetManagerBatchCount(struct Widget_t* widget);
03500
03502 AFTERWARP_API PixelFormat WidgetManagerGetFormat(struct Widget_t* widget);
03503
03505 AFTERWARP_API int32_t WidgetManagerGetMultisamples(struct Widget_t* widget);
03506
03508 AFTERWARP_API uint32_t WidgetManagerGetAttributes(struct Widget_t* widget);
03509
03510 // RandomSequence functions.
03511
03513 AFTERWARP_API void RandomSequenceInit(_Out_ uint64_t* state);
03514
03517 AFTERWARP_API void RandomSequenceInitBySeed(_Out_ uint64_t* state, uint64_t seed);
03518
03520 AFTERWARP_API uint32_t RandomSequenceGenerate(_Inout_ uint64_t* state);
03521
03523 AFTERWARP_API uint64_t RandomSequenceGenerate64(_Inout_ uint64_t* state);
03524
03526 AFTERWARP_API float RandomSequenceGenerateFloat(_Inout_ uint64_t* state);
03527
03529 AFTERWARP_API double RandomSequenceGenerateDouble(_Inout_ uint64_t* state);
03530
03532 AFTERWARP_API int32_t RandomSequenceGenerateRanged(_Inout_ uint64_t* state, int32_t range);
03533
03535 AFTERWARP_API float RandomSequenceGenerateGaussian(_Inout_ uint64_t* state);

```

```

03536
03537 // Ray functions.
03538
03541 AFTERWARP_API void RayCreate(Vector* origin, Vector* direction, PointF const* position,
03542     PointF const* surfaceSize, Matrix const* viewInverse, Matrix const* projection);
03543
03546 AFTERWARP_API LibraryBool RayIntersectTriangle(Vector const* origin, Vector const* direction,
03547     Vector const* vertex1, Vector const* vertex2, Vector const* vertex3, LibraryBool backFacing,
03548     PointF* intersection, float* distance);
03549
03552 AFTERWARP_API LibraryBool RayIntersectCubeVolume(Vector const* origin, Vector const* direction,
03553     Matrix const* world, float* distance);
03554
03557 AFTERWARP_API LibraryBool RayIntersectPlane(Vector const* origin, Vector const* direction,
03558     Vector const* planePoint, Vector const* planeNormal, Vector* intersection, float* distance);
03559
03560 // PixelFormat functions.
03561
03563 AFTERWARP_API LibraryBool PixelFormatValid(PixelFormat format);
03564
03566 AFTERWARP_API uint32_t PixelFormatBits(PixelFormat format);
03567
03569 AFTERWARP_API void PixelFormatConvert(void* dest, void const* source, PixelFormat destFormat,
03570     PixelFormat sourceFormat);
03571
03573 AFTERWARP_API void PixelFormatConvertArray(void* dest, void const* source, PixelFormat destFormat,
03574     PixelFormat sourceFormat, uint32_t destPitch, uint32_t sourcePitch, int32_t width, int32_t height);
03575
03576 // Color functions.
03577
03580 AFTERWARP_API Color MakeColor(Color color, int32_t alpha);
03581
03584 AFTERWARP_API Color MakeColorF(Color color, float alpha);
03585
03589 AFTERWARP_API Color MakeColorWithGray(Color color, int32_t gray, int32_t alpha);
03590
03594 AFTERWARP_API Color MakeColorWithGrayF(Color color, float gray, float alpha);
03595
03598 AFTERWARP_API Color MakeColorRGB(int32_t red, int32_t green, int32_t blue, int32_t alpha);
03599
03602 AFTERWARP_API Color MakeColorRGBF(float red, float green, float blue, float alpha);
03603
03606 AFTERWARP_API Color MakeColorGray(int32_t gray, int32_t alpha);
03607
03610 AFTERWARP_API Color MakeColorGrayF(float gray, float alpha);
03611
03614 AFTERWARP_API Color MakeColorAlpha(int32_t alpha);
03615
03618 AFTERWARP_API Color MakeColorAlphaF(float alpha);
03619
03621 AFTERWARP_API int32_t GetColorAlpha(Color color);
03622
03624 AFTERWARP_API float GetColorAlphaF(Color color);
03625
03628 AFTERWARP_API Color PremultiplyAlpha(Color color);
03629
03632 AFTERWARP_API Color UnpremultiplyAlpha(Color color);
03633
03635 AFTERWARP_API Color DisplaceRB(Color color);
03636
03638 AFTERWARP_API Color InvertColor(Color color);
03639
03641 AFTERWARP_API Color AddColors(Color color1, Color color2);
03642
03644 AFTERWARP_API Color SubtractColors(Color color1, Color color2);
03645
03647 AFTERWARP_API Color MultiplyColors(Color color1, Color color2);
03648
03650 AFTERWARP_API Color AverageColors(Color color1, Color color2);
03651
03653 AFTERWARP_API Color AverageFourColors(Color color1, Color color2, Color color3, Color color4);
03654
03656 AFTERWARP_API Color AverageSixColors(Color color1, Color color2, Color color3, Color color4, Color
03657     color5,
03658     Color color6);
03659
03661 AFTERWARP_API Color BlendColors(Color color1, Color color2, int32_t alpha);
03662
03665 AFTERWARP_API Color BlendFourColors(Color topLeft, Color topRight, Color bottomRight, Color
03666     bottomLeft,
03667     int32_t alphaX, int32_t alphaY, LibraryBool horizontal);
03668
03671 AFTERWARP_API Color ComposeColors(Color source, Color dest);
03672
03675 AFTERWARP_API int32_t ColorToGray(Color color);
03676
03679 AFTERWARP_API int32_t ColorToGray16(Color color);

```

```

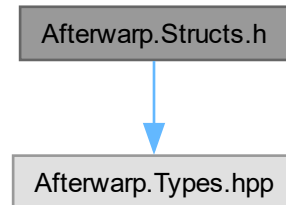
03680
03683 AFTERWARP_API float ColorToGrayF(Color color);
03684
03685 // Interpolation functions.
03686
03688 AFTERWARP_API float Lerp(float value1, float value2, float theta);
03689
03691 AFTERWARP_API float Cubic(float predValue1, float value1, float value2, float succValue2, float
theta);
03692
03695 AFTERWARP_API float CatmullRom(float predValue1, float value1, float value2, float succValue2, float
theta);
03696
03700 AFTERWARP_API float Hermite(float predValue1, float value1, float value2, float succValue2, float
theta,
03701     float tension, float bias);
03702
03705 AFTERWARP_API float SmoothStep(float value1, float value2, float theta);
03706
03710 AFTERWARP_API float SmootherStep(float value1, float value2, float theta);
03711
03713 AFTERWARP_API float SineTransform(float value);
03714
03717 AFTERWARP_API float SineAccelerate(float value);
03718
03721 AFTERWARP_API float SineDecelerate(float value);
03722
03725 AFTERWARP_API float SineCycle(float value);
03726
03728 AFTERWARP_API float SineTwoCycle(float value);
03729
03733 AFTERWARP_API float BiasTransform(float value, float bias);
03734
03738 AFTERWARP_API float GainTransform(float value, float gain);
03739
03740 // Service functions.
03741
03743 AFTERWARP_API void MeshBoundsToMatrixModel(Matrix* model, Vector const* minBounds, Vector const*
maxBounds,
03744     MeshAligns const* aligns, float scale);
03745
03747 AFTERWARP_API void MeshBoundsToMatrixVolume(Matrix* volume, Vector const* minBounds, Vector const*
maxBounds,
03748     MeshAligns const* aligns, float scale);
03749
03751 AFTERWARP_API void MeshBoundsTagOffset(Vector const* meshMinBounds, Vector const* meshMaxBounds,
03752     Vector const* tagMinBounds, Vector const* tagMaxBounds, Vector* offset);
03753
03755 AFTERWARP_API void MeshBoundsToMatrixVolumeTag(Vector const* meshMinBounds, Vector const*
meshMaxBounds,
03756     Vector const* tagMinBounds, Vector const* tagMaxBounds, Matrix* volume, float sizeBias);
03757
03759 AFTERWARP_API void VolumeCalculateNearFarPlanes(Matrix const* worldView, float* nearPlane, float*
farPlane);
03760
03762 AFTERWARP_API void VolumeCalculateVisibleFrame(Matrix const* worldViewProjection, PointF const*
surfaceSize,
03763     RectF* visibleFrame);
03764
03766 AFTERWARP_API uint32_t VertexElementsEstimatePitch(uint32_t channel, VertexElement const
vertexElements[],
03767     int32_t vertexElementCount);
03768
03770 AFTERWARP_API uint64_t GetSystemTicks(void);
03771
03772 #ifdef __cplusplus
03773     } // extern "C"
03774 #endif

```

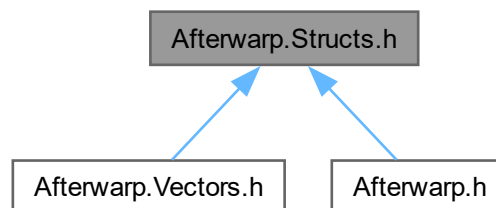
7.3 Afterwarp.Structs.h File Reference

```
#include "Afterwarp.Types.hpp"
```

Include dependency graph for Afterwarp.Structs.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [StencilState](#)
Stencil state that is independent for each front and back facing.
- struct [Blend](#)
Blending parameters that are specific to a certain component, or a portion of color.
- struct [RenderingState](#)
Parameters that define depth, stencil, rasterizer and blending operations.
- struct [VertexElement](#)
Structure that describes the layout of a single buffer element.
- struct [ProgramElement](#)
Structure that describes a single physical element of shader program.
- struct [ProgramVariable](#)
Structure that describes a single (uniform) variable in a shader program.
- struct [ComputeBindTextureFormat](#)
Compute texture parameters and characteristics.

- struct [SamplerState](#)
Sampler parameters that are associated with a particular texture unit.
- struct [TextureParameters](#)
Texture parameters and characteristics.
- struct [SignedDistanceField](#)
Signed Distance Field (SDF) parameters.
- struct [CanvasSamplerState](#)
Sampler parameters that can be easily configured by the canvas.
- struct [ImageRegion](#)
An image region defined by its texture number and bounding rectangle on that texture.
- union [PathElement](#)
A single element in path declaration.
- struct [TextEntryRect](#)
Individual text entry that will appear as rendered.
- struct [TextRenderModifiers](#)
Modifier attributes that can be applied to the rendered text.
- struct [FontEffect](#)
Attributes and characteristics that define how font glyphs are rasterized.
- struct [FontParameters](#)
Parameters that define the appearance and important characteristics of the font.
- struct [MeshBufferEntry](#)
Calculated values for a single mesh vertex that can be modified by callback.
- struct [MeshMetaTagPortion](#)
A portion of the mesh corresponding to a particular tag.
- struct [SelectionHighlightParameters](#)
Parameters that define how selection highlight technique is performed.
- struct [FogParameters](#)
Structure containing spatial fog characteristics.
- struct [ShadowParameters](#)
Parameters that define how shadows are rendered.
- struct [ToneMappingBloom](#)
Parameters that define behavior of tone-mapping and bloom effects.
- struct [AmbientOcclusionParameters](#)
Parameters that define how ambient occlusion is performed.
- struct [ParallaxMappingParameters](#)
Parameters that define how parallax mapping is performed.
- struct [SceneLight](#)
Light source parameters in 3D scene.
- struct [ObjectMaterial](#)
Material that describes a particular object in 3D world.
- struct [OceanWavesParameters](#)
Parameters that define the appearance of waves simulation.
- struct [OceanMaterial](#)
Material that describes ocean water surface in a 3D simulation.
- struct [MeshAligns](#)
Alignment for all three axes that determine the placement of mesh around its model.
- struct [SceneMeshMaterialShading](#)
Shading parameters for scene mesh material.
- struct [SceneMeshMaterialRange](#)
A range of indices in the mesh that a material applies to.
- struct [SceneMeshLatch](#)

- A latch in 3D scene mesh, which describes a bone joint connection or a movement waypoint.*

 - struct [ApplicationConfiguration](#)

Structure that contains all the parameters necessary to create a new application and its window.
- struct [ApplicationEvents](#)
- struct [CameraConstraints](#)
- union [WidgetPropertyData](#)
- A flexible structure for storing property data.*

 - struct [WidgetProperty](#)

Information about a widget property.

Namespaces

- namespace [pxt](#)

Typedefs

- typedef void * [UntypedHandle](#)
- Generic untyped handle.*

 - typedef uint8_t [LibraryBool](#)

Boolean type returned by the library.
- typedef void * [ObjectPayload](#)
- Custom object's payload.*

 - typedef uint32_t [DeviceAttributes](#)

One or more device attribute flags combined (ORed) together.
- typedef uint32_t [DeviceBehavior](#)
- Device behavior flags that affect the rendering code path.*

 - typedef uint32_t [DeviceLegacyBits](#)

Legacy feature bits that define older hardware features.
- typedef struct [pxt::RenderingState](#) [RenderingState](#)
- Parameters that define depth, stencil, rasterizer and blending operations.*

 - typedef struct [Buffer_t](#) [Buffer_t](#)

Buffer that is typically stored on graphics hardware.
- typedef struct [pxt::VertexElement](#) [VertexElement](#)
- Structure that describes the layout of a single buffer element.*

 - typedef struct [pxt::ProgramElement](#) [ProgramElement](#)

Structure that describes a single physical element of shader program.
- typedef struct [pxt::ProgramVariable](#) [ProgramVariable](#)
- Structure that describes a single (uniform) variable in a shader program.*

 - typedef struct [Program_t](#) [Program_t](#)

Shader program that is typically executed on graphics hardware.
- typedef struct [pxt::ComputeBindTextureFormat](#) [ComputeBindTextureFormat](#)
- Compute texture parameters and characteristics.*

 - typedef struct [ComputeProgram_t](#) [ComputeProgram_t](#)

Compute shader program for general processing on GPU (GPGPU).
- typedef struct [pxt::SamplerState](#) [SamplerState](#)
- Sampler parameters that are associated with a particular texture unit.*

 - typedef struct [Sampler_t](#) [Sampler_t](#)

Sampler object that defines how texture is read by the shaders.
- typedef uint32_t [TextureAttributes](#)
- One or more texture attribute flags combined together.*

- typedef struct [pxt::TextureParameters TextureParameters](#)
Texture parameters and characteristics.
- typedef struct [Texture_t Texture_t](#)
Texture that contains image data typically stored in GPU memory.
- typedef struct [Surface_t Surface_t](#)
Raster surface that contains image in memory for pixel manipulation.
- typedef uint32_t [CanvasAttributes](#)
One or more canvas attributes that define rendering behavior.
- typedef struct [pxt::SignedDistanceField SignedDistanceField](#)
Signed Distance Field (SDF) parameters.
- typedef struct [pxt::CanvasSamplerState CanvasSamplerState](#)
Sampler parameters that can be easily configured by the canvas.
- typedef struct [pxt::ImageRegion ImageRegion](#)
An image region defined by its texture number and bounding rectangle on that texture.
- typedef struct [Canvas_t Canvas_t](#)
Canvas that can render geometry either in 2D or on a particular plane in 3D world.
- typedef struct [ImageAtlas_t ImageAtlas_t](#)
Image atlas, containing sub-images that can be rendered with canvas.
- typedef uint32_t [UnicodeChar](#)
32-bit Unicode character type
- typedef struct [pxt::TextEntryRect TextEntryRect](#)
Individual text entry that will appear as rendered.
- typedef struct [pxt::TextRenderModifiers TextRenderModifiers](#)
Modifier attributes that can be applied to the rendered text.
- typedef struct [pxt::FontEffect FontEffect](#)
Attributes and characteristics that define how font glyphs are rasterized.
- typedef struct [pxt::FontParameters FontParameters](#)
Parameters that define the appearance and important characteristics of the font.
- typedef struct [TextRenderer_t TextRenderer_t](#)
Text renderer utility that can draw text on the canvas.
- typedef struct [TextModeller_t TextModeller_t](#)
2D and 3D text rendering module.
- typedef struct [Grapher_t Grapher_t](#)
3D graph plotting module.
- typedef struct [MeshModel_t MeshModel_t](#)
3D mesh model that has vertex and optionally index buffers for rendering.
- typedef int32_t(* [MeshLoadSaveFeedback](#)) (char const *message, int32_t progress, void *user)
- typedef struct [pxt::MeshBufferEntry MeshBufferEntry](#)
Calculated values for a single mesh vertex that can be modified by callback.
- typedef struct [pxt::MeshMetaTagPortion MeshMetaTagPortion](#)
A portion of the mesh corresponding to a particular tag.
- typedef struct [MeshMetaTag_t MeshMetaTag_t](#)
Meta-tag, which describes a certain important axis-aligned bounding area inside a mesh.
- typedef struct [MeshMetaTags_t MeshMetaTags_t](#)
List of meta-tags that describe important axis-aligned bounding areas inside a mesh.
- typedef struct [MeshBuffer_t MeshBuffer_t](#)
Buffer that contains 3D mesh information.
- typedef struct [GaussianBlur_t GaussianBlur_t](#)
Gaussian Blur module.
- typedef struct [KawaseBlur_t KawaseBlur_t](#)
Kawase Blur module.

- typedef struct [ColorDithering_t](#) [ColorDithering_t](#)
Color dithering module.
- typedef struct [pxt::SelectionHighlightParameters](#) [SelectionHighlightParameters](#)
Parameters that define how selection highlight technique is performed.
- typedef struct [SelectionHighlight_t](#) [SelectionHighlight_t](#)
Selection highlight module.
- typedef struct [pxt::FogParameters](#) [FogParameters](#)
Structure containing spatial fog characteristics.
- typedef struct [SpatialFog_t](#) [SpatialFog_t](#)
Module that performs both ground fog and distance-based fog simultaneously.
- typedef struct [pxt::ShadowParameters](#) [ShadowParameters](#)
Parameters that define how shadows are rendered.
- typedef struct [pxt::ToneMappingBloom](#) [ToneMappingBloom](#)
Parameters that define behavior of tone-mapping and bloom effects.
- typedef struct [pxt::AmbientOcclusionParameters](#) [AmbientOcclusionParameters](#)
Parameters that define how ambient occlusion is performed.
- typedef struct [pxt::ParallaxMappingParameters](#) [ParallaxMappingParameters](#)
Parameters that define how parallax mapping is performed.
- typedef struct [pxt::SceneLight](#) [SceneLight](#)
Light source parameters in 3D scene.
- typedef struct [pxt::ObjectMaterial](#) [ObjectMaterial](#)
Material that describes a particular object in 3D world.
- typedef struct [pxt::OceanWavesParameters](#) [OceanWavesParameters](#)
Parameters that define the appearance of waves simulation.
- typedef struct [pxt::OceanMaterial](#) [OceanMaterial](#)
Material that describes ocean water surface in a 3D simulation.
- typedef uint32_t [TextureCabinetAttributes](#)
Cumulative rendering attributes that affect how the scene is filtered.
- typedef uint32_t [SceneAttributes](#)
Rendering attributes that define what type of resources are used and how rendering is performed.
- typedef struct [SceneLights_t](#) [SceneLights_t](#)
Container for storing 3D scene lights.
- typedef struct [ObjectMaterials_t](#) [ObjectMaterials_t](#)
Container for storing 3D object materials.
- typedef struct [TextureCabinet_t](#) [TextureCabinet_t](#)
A collection of drawable textures used to render the scene.
- typedef struct [ShadowCaster_t](#) [ShadowCaster_t](#)
A source that casts shadows that can be shared among multiple light sources.
- typedef struct [ShadowCastingAtlas_t](#) [ShadowCastingAtlas_t](#)
Module that manages shadow maps produced by one or more shadow casters.
- typedef struct [Scene_t](#) [Scene_t](#)
3D rendering scene.
- typedef struct [OceanSimulation_t](#) [OceanSimulation_t](#)
3D ocean semi-infinite wave field rendering module.
- typedef uintptr_t [ObjectModelID](#)
Data type for storing unique object identification numbers.
- typedef struct [pxt::MeshAligns](#) [MeshAligns](#)
Alignment for all three axes that determine the placement of mesh around its model.
- typedef struct [ObjectModel_t](#) [ObjectModel_t](#)
Object model represented in a working 3D environment.
- typedef struct [ObjectModels_t](#) [ObjectModels_t](#)

- Container and manager for working with multiple 3D scene objects.*

 - typedef struct [ObjectModelsIterator_t](#) [ObjectModelsIterator_t](#)
 - Iterator for retrieving individual objects from an object model's container.*
 - typedef struct [ObjectModelView_t](#) [ObjectModelView_t](#)
 - A container that represents a particular view that watches objects in the world.*
 - typedef int32_t(* [ObjectModelCompareFunc](#)) ([ObjectModel_t](#) const *object1, [ObjectModel_t](#) const *object2, void *user)
 - Comparison function for a pair of objects in a 3D scene.*
 - typedef struct [MeshVoxel_t](#) [MeshVoxel_t](#)
 - Voxel representation of a 3D mesh model.*
 - typedef void(* [MeshVoxelVisualizeFunc](#)) ([Vector](#) const *position, [Vector](#) const *size, [FloatColor](#) const *color, void *user)
 - Mesh voxel rendering function.*
 - typedef struct [pxt::SceneMeshMaterialShading](#) [SceneMeshMaterialShading](#)
 - Shading parameters for scene mesh material.*
 - typedef struct [pxt::SceneMeshMaterialRange](#) [SceneMeshMaterialRange](#)
 - A range of indices in the mesh that a material applies to.*
 - typedef struct [SceneMeshMaterial_t](#) [SceneMeshMaterial_t](#)
 - Material that describes how a portion of a scene mesh geometry should look like.*
 - typedef struct [SceneMeshMaterials_t](#) [SceneMeshMaterials_t](#)
 - Container for scene mesh material that describe the appearance of scene mesh geometry.*
 - typedef struct [pxt::SceneMeshLatch](#) [SceneMeshLatch](#)
 - A latch in 3D scene mesh, which describes a bone joint connection or a movement waypoint.*
 - typedef struct [SceneMeshLatches_t](#) [SceneMeshLatches_t](#)
 - A collection of marks in a 3D mesh to denote bone joint connections and movement waypoints.*
 - typedef struct [SceneMesh_t](#) [SceneMesh_t](#)
 - High-level wrapper around a rendereable and pickable mesh in 3D scene.*
 - typedef struct [SceneMeshes_t](#) [SceneMeshes_t](#)
 - Container for high-level wrappers of rendereable and pickable objects in 3D scene.*
 - typedef struct [pxt::ApplicationConfiguration](#) [ApplicationConfiguration](#)
 - Structure that contains all the parameters necessary to create a new application and its window.*
 - typedef struct [Application_t](#) [Application_t](#)
 - Application helper module.*
 - typedef void(* [EventFunc](#)) ([Application_t](#) *application, void *user)
 - Basic application event.*
 - typedef [LibraryBool](#)(* [BoolFunc](#)) ([Application_t](#) *application, void *user)
 - Application creation event.*
 - typedef void(* [MouseFunc](#)) ([Application_t](#) *application, [MouseEvent](#) event, [MouseButton](#) button, [Point](#) const *position, void *user)
 - Application mouse handling event.*
 - typedef void(* [KeyboardFunc](#)) ([Application_t](#) *application, [KeyEvent](#) event, int32_t virtualKey, uint16_t key↵ Code, void *user)
 - Application keyboard handling event.*
 - typedef void(* [StatusFunc](#)) ([Application](#) *application, [ApplicationEvent](#) event, void *user)
 - Event function for application status change events.*
 - typedef [LibraryBool](#)(* [EventHookFunc](#)) ([Application_t](#) *application, void const *event, void *user)
 - Custom application message handling event.*
 - typedef struct [pxt::ApplicationEvents](#) [ApplicationEvents](#)
 - typedef struct [pxt::CameraConstraints](#) [CameraConstraints](#)
 - typedef struct [ActorCamera_t](#) [ActorCamera_t](#)
 - typedef struct [Timer_t](#) [Timer_t](#)

- *Multimedia timer that can accurately calculate latency, frame rate and provide time-based processing.*
- typedef union `pxt::WidgetPropertyData` `WidgetPropertyData`
A flexible structure for storing property data.
- typedef struct `pxt::WidgetProperty` `WidgetProperty`
Information about a widget property.
- typedef struct `Widget_t` `Widget_t`
A widget that represents a basic building block of user interface.
- typedef `LibraryBool(* WidgetExternalEvent)` (struct `Widget_t *widget`, char const *event, void *user)
Widget's external callback event type.

Functions

- uint32_t `deviceAttributeExtensions` (uint8_t const extensionLevel)

7.4 Afterwarp.Structs.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Afterwarp Framework (https://afterwarp.io).
00003  * Copyright (c) 2015 - 2025 Dr. Yuriy Kotsarenko. All rights reserved.
00004  *
00005  * Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in
00006  * compliance with the License. You may obtain a copy of the License at
00007  * http://www.apache.org/licenses/LICENSE-2.0
00008  *
00009  * Unless required by applicable law or agreed to in writing, software distributed under the License
00010  * is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
00011  * implied.
00012  * See the License for the specific language governing permissions and limitations under the License.
00013  */
00014 // Afterwarp.Types.h
00015 #pragma once
00016
00017 #ifdef __cplusplus
00018     #include "Afterwarp.Types.hpp"
00019 #else
00020     #include "Afterwarp.Types.h"
00021 #endif
00022
00023 // Calling conventions for library functions.
00024 #if defined(_WIN32) || defined(__CYGWIN__)
00025     #ifdef PXT_LIBRARY
00026         #ifdef __GNUC__
00027             #define __AFTERWARP_API __attribute__((dllexport))
00028         #else
00029             #define __AFTERWARP_API __declspec(dllexport)
00030         #endif
00031     #else
00032         #ifdef __GNUC__
00033             #define __AFTERWARP_API __attribute__((dllimport))
00034         #else
00035             #define __AFTERWARP_API __declspec(dllimport)
00036         #endif
00037     #endif
00038 #else
00039     #define __AFTERWARP_API __attribute__((visibility("default")))
00040 #endif
00041
00042 #if defined(__cplusplus)
00043     #define AFTERWARP_API extern "C" __AFTERWARP_API
00044 #else
00045     #define AFTERWARP_API __AFTERWARP_API
00046 #endif
00047
00048 #ifdef __cplusplus
00049     inline namespace pxt {
00050 #endif
00051
00061 // General types and enumerations.
00062

```

```

00064 typedef void* UntypedHandle;
00065
00067 typedef uint8_t LibraryBool;
00068
00070 typedef void* ObjectPayload;
00071
00072 // Device types and structures.
00073
00075 struct SwapChain_t;
00076
00077 #ifndef __cplusplus
00079     typedef struct SwapChain_t SwapChain;
00080 #endif
00081
00083 typedef uint32_t DeviceAttributes;
00084
00086 typedef uint32_t DeviceBehavior;
00087
00089 typedef uint32_t DeviceLegacyBits;
00090
00096 #ifdef __cplusplus
00097     #ifndef PXT_AFTERWARP_FRAMEWORK
00098         inline uint32_t deviceAttributeExtensions(uint8_t const extensionLevel)
00099         {
00100             return static_cast<uint32_t>(extensionLevel + 1u) << 29;
00101         }
00102     #endif
00103 #else
00104     #define DEVICE_ATTRIBUTE_EXTENSIONS(X) ((uint32_t)((X) + 1u) << 29)
00105 #endif
00106
00108 struct StencilState
00109 {
00111     ComparisonFunc func;
00112
00114     StencilOp failOp;
00115
00117     StencilOp depthFailOp;
00118
00121     StencilOp depthPassOp;
00122 };
00123
00125 struct Blend
00126 {
00128     BlendFactor source;
00129
00131     BlendFactor dest;
00132
00134     BlendOp op;
00135 };
00136
00138 typedef struct RenderingState
00139 {
00141     uint32_t states;
00142
00144     TriangleFace cullFace;
00145
00147     ComparisonFunc depthFunc;
00148
00150     float depthBias;
00151
00153     float slopeDepthBias;
00154
00156     float clampDepthBias;
00157
00159     uint8_t stencilRefValue;
00160
00162     uint8_t stencilRefMask;
00163
00165     uint8_t stencilWriteMask;
00166
00168     FloatColor blendConstant;
00169
00171     struct StencilState stencilFront;
00172
00174     struct StencilState stencilBack;
00175
00177     struct Blend blendColor;
00178
00180     struct Blend blendAlpha;
00181 } RenderingState;
00182
00184 struct Device_t;
00185
00186 #ifndef __cplusplus
00188     typedef struct Device_t Device;
00189 #endif

```

```

00190
00191 // Buffer types and structures.
00192
00194 typedef struct Buffer_t Buffer_t;
00195
00196 #ifndef __cplusplus
00198     typedef Buffer_t Buffer;
00199 #endif
00200
00201 // Program types and structures.
00202
00204 typedef struct VertexElement
00205 {
00207     char name[64];
00208
00210     ElementFormat format;
00211
00214     int32_t count;
00215
00219     uint32_t channel;
00220
00222     uint32_t offset;
00223 } VertexElement;
00224
00226 typedef struct ProgramElement
00227 {
00229     char name[64];
00230
00232     ShaderElement element;
00233
00235     ShaderType shader;
00236
00238     uint32_t channel;
00239
00242     int32_t index;
00243
00245     uint32_t size;
00246 } ProgramElement;
00247
00249 typedef struct ProgramVariable
00250 {
00252     char name[64];
00253
00255     int32_t index;
00256
00258     ShaderType shader;
00259
00261     ElementFormat format;
00262
00265     int32_t count;
00266
00268     uint32_t size;
00269
00271     uint32_t offset;
00272 } ProgramVariable;
00273
00275 typedef struct Program_t Program_t;
00276
00277 #ifndef __cplusplus
00279     typedef Program_t Program;
00280 #endif
00281
00282 // Compute structures.
00283
00285 typedef struct ComputeBindTextureFormat
00286 {
00288     uint32_t channel;
00289
00291     int32_t mipLevel;
00292
00294     int32_t layer;
00295
00297     ComputeTextureAccess access;
00298
00300     PixelFormat format;
00301 } ComputeBindTextureFormat;
00302
00304 typedef struct ComputeProgram_t ComputeProgram_t;
00305
00306 #ifndef __cplusplus
00308     typedef ComputeProgram_t ComputeProgram;
00309 #endif
00310
00311 // Sampler structures.
00312
00314 typedef struct SamplerState
00315 {

```

```

00317 TextureFilter filterMin;
00318
00320 TextureFilter filterMag;
00321
00323 TextureFilter filterMip;
00324
00326 TextureAddress address[3];
00327
00329 FloatColor borderColor;
00330
00332 int32_t anisotropy;
00333
00335 float minLOD;
00336
00338 float maxLOD;
00339
00341 float biasLOD;
00342
00344 ComparisonFunc compareFunc;
00345
00347 uint8_t compareToRef; // 0 - false, 1 - true
00348 } SamplerState;
00349
00351 typedef struct Sampler_t Sampler_t;
00352
00353 #ifndef __cplusplus
00355     typedef Sampler_t Sampler;
00356 #endif
00357
00358 // Texture structures.
00359
00361 typedef uint32_t TextureAttributes;
00362
00364 typedef struct TextureParameters
00365 {
00367     int32_t width;
00368
00370     int32_t height;
00371
00373     int32_t layers;
00374
00376     PixelFormat format;
00377
00379     TextureType type;
00380
00382     TextureAttributes attributes;
00383
00385     int32_t multisamples;
00386
00388     PixelFormat depthStencil;
00389 } TextureParameters;
00390
00392 typedef struct Texture_t Texture_t;
00393
00394 #ifndef __cplusplus
00396     typedef Texture_t Texture;
00397 #endif
00398
00399 // Surface types.
00400
00402 typedef struct Surface_t Surface_t;
00403
00404 #ifndef __cplusplus
00406     typedef Surface_t Surface;
00407 #endif
00408
00409 // Canvas types and structures.
00410
00412 typedef uint32_t CanvasAttributes;
00413
00415 typedef struct SignedDistanceField
00416 {
00418     SuperSampleSDF superSampleSDF;
00419
00421     float signedFieldDistance;
00422
00424     Vector outlineOffsetSDF;
00425
00427     float outlineDistanceMinSDF;
00428
00430     float outlineDistanceMaxSDF;
00431 } SignedDistanceField;
00432
00434 typedef struct CanvasSamplerState
00435 {
00437     TextureFilter filterMin;
00438

```



```

00440 TextureFilter filterMag;
00441
00443 TextureFilter filterMip;
00444
00446 TextureAddress addressU;
00447
00449 TextureAddress addressV;
00450
00452 Color borderColor;
00453 } CanvasSamplerState;
00454
00456 typedef struct ImageRegion
00457 {
00458 #ifdef __cplusplus
00460 enum {
00462     Flip = 0x01,
00463
00465     Mirror = 0x02,
00466
00468     Rotate = 0x04};
00469 #endif
00470
00472 uint16_t left;
00473
00475 uint16_t top;
00476
00478 uint16_t width;
00479
00481 uint16_t height;
00482
00484 uint16_t index;
00485 } ImageRegion;
00486
00488 union PathElement
00489 {
00491 struct
00492 {
00494     uint8_t command;
00495
00497     uint8_t reserved;
00498
00500     uint16_t length;
00501 } tag;
00502
00504 float value;
00505
00506 #ifdef __cplusplus
00508 PathElement() = default;
00509
00511 PathElement(uint8_t command, uint16_t length = 0u)
00512 {
00513     tag.command = command;
00514     tag.reserved = 0;
00515     tag.length = length;
00516 }
00517
00519 PathElement(float value) { this->value = value; }
00520 #endif
00521 };
00522
00523 #ifndef __cplusplus
00525 typedef union PathElement PathElement;
00526 #endif
00527
00529 typedef struct Canvas_t Canvas_t;
00530
00531 #ifndef __cplusplus
00533 typedef Canvas_t Canvas;
00534 #endif
00535
00537 typedef struct ImageAtlas_t ImageAtlas_t;
00538
00539 #ifndef __cplusplus
00541 typedef ImageAtlas_t ImageAtlas;
00542 #endif
00543
00544 // TextRenderer types and structures.
00545
00547 typedef uint32_t UnicodeChar;
00548
00550 typedef struct TextEntryRect
00551 {
00553     intptr_t position;
00554
00556     RectF rectangle;
00557 } TextEntryRect;
00558

```

```

00560 typedef struct TextRenderModifiers
00561 {
00566     float scale;
00567
00570     float interleave;
00571
00573     float verticalSpace;
00574
00576     BlendingEffect effect;
00577 } TextRenderModifiers;
00578
00580 typedef struct FontEffect
00581 {
00584     float fillBrightness;
00585
00588     float fillOpacity;
00589
00591     FontBorder borderType;
00592
00594     float borderThickness;
00595
00598     float borderBrightness;
00599
00602     float borderOpacity;
00603
00605     float shadowSmoothness;
00606
00608     PointF shadowDistance;
00609
00612     float shadowBrightness;
00613
00616     float shadowOpacity;
00617
00620     float signedFieldDistance;
00621 } FontEffect;
00622
00624 typedef struct FontParameters
00625 {
00627     char family[128];
00628
00630     float size;
00631
00633     FontWeight weight;
00634
00636     FontStretch stretch;
00637
00639     FontSlant slant;
00640
00642     FontAttributes attributes;
00643
00645     FontEffect effect;
00646
00647 #ifdef __cplusplus
00649     FontSettings() = default;
00650
00652     FontParameters(char const* family, float size, FontWeight weight = FontWeight::Normal,
00653         FontStretch stretch = FontStretch::Normal, FontSlant slant = FontSlant::None)
00654         : family{family}, size(size), weight(weight), stretch(stretch), slant(slant), attributes(0),
00655         effect{1.0f, 1.0f, FontBorder::None, 1.0f, 0.25f, 0.75f, 3.0f, {2.0f, 2.0f}, 0.15f, 0.0f, 0.0f}
00656         { strncpy_s(this->family, sizeof(this->family), family, sizeof(this->family) - 1); }
00657 #endif
00658 } FontParameters;
00659
00661 typedef struct TextRenderer_t TextRenderer_t;
00662
00663 #ifndef __cplusplus
00665     typedef TextRenderer_t TextRenderer;
00666 #endif
00667
00669 typedef struct TextModeller_t TextModeller_t;
00670
00671 #ifndef __cplusplus
00673     typedef TextModeller_t TextModeller;
00674 #endif
00675
00676 // Grapher types.
00677
00679 typedef struct Grapher_t Grapher_t;
00680
00681 #ifndef __cplusplus
00683     typedef Grapher_t Grapher;
00684 #endif
00685
00686 // MeshModel types and structures.
00687
00689 typedef struct MeshModel_t MeshModel_t;
00690

```

```

00691 #ifndef __cplusplus
00693     typedef MeshModel_t MeshModel;
00694 #endif
00695
00696 // MeshBuffer types and structures.
00697
00701 typedef int32_t (*MeshLoadSaveFeedback)(char const* message, int32_t progress, void* user);
00702
00704 typedef struct MeshBufferEntry
00705 {
00707     Vector position;
00708
00710     Vector normal;
00711
00713     Vector tangent;
00714
00716     PointF texCoord;
00717
00719     FloatColor color;
00720 } MeshBufferEntry;
00721
00723 typedef struct MeshMetaTagPortion
00724 {
00726     int32_t firstIndex;
00727
00729     int32_t indexCount;
00730
00732     int32_t firstVertex;
00733
00735     int32_t vertexCount;
00736
00738     Vector boundsMin;
00739
00741     Vector boundsMax;
00742 } MeshMetaTagPortion;
00743
00745 typedef struct MeshMetaTag_t MeshMetaTag_t;
00746
00747 #ifndef __cplusplus
00749     typedef MeshMetaTag_t MeshMetaTag;
00750 #endif
00751
00753 typedef struct MeshMetaTags_t MeshMetaTags_t;
00754
00755 #ifndef __cplusplus
00757     typedef MeshMetaTags_t MeshMetaTags;
00758 #endif
00759
00761 typedef struct MeshBuffer_t MeshBuffer_t;
00762
00763 #ifndef __cplusplus
00765     typedef MeshBuffer_t MeshBuffer;
00766 #endif
00767
00768 // GaussianBlur types.
00769
00771 typedef struct GaussianBlur_t GaussianBlur_t;
00772
00773 #ifndef __cplusplus
00775     typedef GaussianBlur_t GaussianBlur;
00776 #endif
00777
00778 // KawaseBlur types.
00779
00781 typedef struct KawaseBlur_t KawaseBlur_t;
00782
00783 #ifndef __cplusplus
00785     typedef KawaseBlur_t KawaseBlur;
00786 #endif
00787
00788 // ColorDithering types.
00789
00791 typedef struct ColorDithering_t ColorDithering_t;
00792
00793 #ifndef __cplusplus
00795     typedef ColorDithering_t ColorDithering;
00796 #endif
00797
00798 // SelectionHighlight types.
00799
00801 typedef struct SelectionHighlightParameters
00802 {
00804     FloatColorRGB outlineColor;
00805
00807     float outlineStart;
00808
00810     float glowIntensity;

```

```
00811
00813     int32_t blurPasses;
00814
00816     PointF blurOffset;
00817 } SelectionHighlightParameters;
00818
00820 typedef struct SelectionHighlight_t SelectionHighlight_t;
00821
00822 #ifndef __cplusplus
00824     typedef SelectionHighlight_t SelectionHighlight;
00825 #endif
00826
00827 // SpatialFog types.
00828
00830 typedef struct FogParameters
00831 {
00833     FloatColor color;
00834
00836     float opacity;
00837
00839     Vector4 groundPlane;
00840
00842     float densityGround;
00843
00845     union {
00846         float distance;
00847         float scattering;
00848     };
00849
00851     union {
00852         float bias;
00853         float extinction;
00854     };
00855 } FogParameters;
00856
00858 typedef struct SpatialFog_t SpatialFog_t;
00859
00860 #ifndef __cplusplus
00862     typedef SpatialFog_t SpatialFog;
00863 #endif
00864
00865 // Scene types and structures.
00866
00868 typedef struct ShadowParameters
00869 {
00871     float exponent1;
00872
00874     float exponent2;
00875
00877     float variance;
00878
00880     float bleedSigma;
00881
00883     float bias;
00884
00886     float blurSigma;
00887
00889     int32_t blurSamples;
00890 } ShadowParameters;
00891
00893 typedef struct ToneMappingBloom
00894 {
00896     float bloomThreshold;
00897
00899     float bloomGamma;
00900
00902     Vector bloomCoefficients;
00903
00905     float bloomColorShift;
00906
00908     float bloomBlurSigma;
00909
00911     int32_t bloomBlurSamples;
00912
00914     float toneWhite;
00915
00917     Vector toneFactors;
00918
00920     float frostedPower;
00921
00923     float glassyBuckets;
00924 } ToneMappingBloom;
00925
00927 typedef struct AmbientOcclusionParameters
00928 {
00930     float radius;
```

```
00933     int32_t samples;
00934
00936     float attenuation;
00937
00939     float contrast;
00940
00942     float depthBias;
00943
00945     float kernelBias;
00946
00948     float blurSigma;
00949
00951     float blurFallOff;
00952
00954     float strength;
00955 } AmbientOcclusionParameters;
00956
00958 typedef struct ParallaxMappingParameters
00959 {
00961     float scale;
00962
00964     int32_t samplesMin;
00965
00967     int32_t samplesMax;
00968
00970     float occlusion;
00971 } ParallaxMappingParameters;
00972
00974 typedef struct SceneLight
00975 {
00977     FloatColorRGB ambientColor;
00978
00980     float attenuationStart;
00981
00983     FloatColorRGB albedoColor;
00984
00986     float attenuationEnd;
00987
00989     FloatColorRGB specularColor;
00990
00992     float specularExponent;
00993
00995     Vector position;
00996
00998     float intensity;
00999
01001     Vector direction;
01002
01004     float angle;
01005
01007     float angleCutoff;
01008
01011     int32_t shadowCaster;
01012
01015     uint32_t bitmask;
01016
01018     uint32_t reserved;
01019
01021     ObjectPayload payload;
01022 } SceneLight;
01023
01025 typedef struct ObjectMaterial
01026 {
01028     TechniqueLighting technique;
01029
01031     FloatColorRGB ambientColor;
01032
01034     float shadows;
01035
01037     FloatColor albedoColor;
01038
01040     FloatColorRGB specularColor;
01041
01043     float specularExponent;
01044
01046     FloatColorRGB emissiveColor;
01047
01049     float occlusion;
01050
01052     Vector roughness;
01053
01056     uint32_t bitmask;
01057
01059     float frostedGlass;
01060
01062     ObjectPayload payload;
01063 } ObjectMaterial;
```

```

01064
01066 typedef struct OceanWavesParameters
01067 {
01069     int32_t resolution;
01070
01072     PointF wind;
01073
01075     float waveSize;
01076
01078     float choppiness;
01079 } OceanWavesParameters;
01080
01082 typedef struct OceanMaterial
01083 {
01085     FloatColorRGB ambientColor;
01086
01088     float fresnel;
01089
01091     FloatColor albedoColor;
01092
01094     FloatColorRGB specularColor;
01095
01097     float specularExponent;
01098
01100     FloatColorRGB emissiveColor;
01101
01103     float extinction;
01104
01107     float shadows;
01108
01111     uint32_t bitmask;
01112 } OceanMaterial;
01113
01115 typedef uint32_t TextureCabinetAttributes;
01116
01118 typedef uint32_t SceneAttributes;
01119
01121 typedef struct SceneLights_t SceneLights_t;
01122
01123 #ifndef __cplusplus
01125     typedef SceneLights_t SceneLights;
01126 #endif
01127
01129 typedef struct ObjectMaterials_t ObjectMaterials_t;
01130
01131 #ifndef __cplusplus
01133     typedef ObjectMaterials_t ObjectMaterials;
01134 #endif
01135
01137 typedef struct TextureCabinet_t TextureCabinet_t;
01138
01139 #ifndef __cplusplus
01141     typedef TextureCabinet_t TextureCabinet;
01142 #endif
01143
01145 typedef struct ShadowCaster_t ShadowCaster_t;
01146
01147 #ifndef __cplusplus
01149     typedef ShadowCaster_t ShadowCaster;
01150 #endif
01151
01153 typedef struct ShadowCastingAtlas_t ShadowCastingAtlas_t;
01154
01155 #ifndef __cplusplus
01157     typedef ShadowCastingAtlas_t ShadowCastingAtlas;
01158 #endif
01159
01161 typedef struct Scene_t Scene_t;
01162
01163 #ifndef __cplusplus
01165     typedef Scene_t Scene;
01166 #endif
01167
01169 typedef struct OceanSimulation_t OceanSimulation_t;
01170
01171 #ifndef __cplusplus
01173     typedef OceanSimulation_t OceanSimulation;
01174 #endif
01175
01176 // ObjectModel types.
01177
01179 typedef uintptr_t ObjectModelID;
01180
01182 typedef struct MeshAligns
01183 {
01185     MeshAlign x;
01186

```

```

01188     MeshAlign y;
01189
01191     MeshAlign z;
01192
01194     float bias;
01195
01196 #ifdef __cplusplus
01198     MeshAligns() = default;
01199
01201     MeshAligns(MeshAlign x = MeshAlign::Origin, MeshAlign y = MeshAlign::Positive,
01202               MeshAlign z = MeshAlign::Origin, float bias = 0.1f) : x(x), y(y), z(z), bias(bias) {}
01203 #endif
01204 } MeshAligns;
01205
01207 typedef struct ObjectModel_t ObjectModel_t;
01208
01209 #ifndef __cplusplus
01211     typedef ObjectModel_t ObjectModel;
01212 #endif
01213
01215 typedef struct ObjectModels_t ObjectModels_t;
01216
01217 #ifndef __cplusplus
01219     typedef ObjectModels_t ObjectModels;
01220 #endif
01221
01223 typedef struct ObjectModelsIterator_t ObjectModelsIterator_t;
01224
01225 #ifndef __cplusplus
01227     typedef ObjectModelsIterator_t ObjectModelsIterator;
01228 #endif
01229
01231 typedef struct ObjectModelView_t ObjectModelView_t;
01232
01233 #ifndef __cplusplus
01235     typedef ObjectModelView_t ObjectModelView;
01236 #endif
01237
01239 typedef int32_t(*ObjectModelCompareFunc)(ObjectModel_t const* object1, ObjectModel_t const* object2,
01240                                         void* user);
01241
01242 // MeshVoxel types.
01243
01245 typedef struct MeshVoxel_t MeshVoxel_t;
01246
01247 #ifndef __cplusplus
01249     typedef MeshVoxel_t MeshVoxel;
01250 #endif
01251
01253 typedef void(*MeshVoxelVisualizeFunc)(Vector const* position, Vector const* size, FloatColor const*
01254                                       color,
01255                                       void* user);
01256
01256 // SceneMeshMaterial types.
01257
01259 typedef struct SceneMeshMaterialShading
01260 {
01262     FloatColorRGB ambient;
01263
01265     float bloom;
01266
01268     FloatColor diffuse;
01269
01271     FloatColorRGB specular;
01272
01274     float specularExponent;
01275
01277     FloatColorRGB emissive;
01278
01280     int32_t lighting;
01281
01283     Vector roughness;
01284 } SceneMeshMaterialShading;
01285
01287 typedef struct SceneMeshMaterialRange
01288 {
01290     int32_t indexStart;
01291
01293     int32_t indexCount;
01294 } SceneMeshMaterialRange;
01295
01297 typedef struct SceneMeshMaterial_t SceneMeshMaterial_t;
01298
01299 #ifndef __cplusplus
01301     typedef SceneMeshMaterial_t SceneMeshMaterial;
01302 #endif
01303

```

```

01305 typedef struct SceneMeshMaterials_t SceneMeshMaterials_t;
01306
01307 #ifndef __cplusplus
01309     typedef SceneMeshMaterials_t SceneMeshMaterials;
01310 #endif
01311
01312 // SceneMeshLatches types.
01313
01315 typedef struct SceneMeshLatch
01316 {
01318     char name[256];
01319
01321     Vector position;
01322
01324     uint16_t type;
01325
01327     uint16_t group;
01328
01330     Quaternion orientation;
01331 } SceneMeshLatch;
01332
01334 typedef struct SceneMeshLatches_t SceneMeshLatches_t;
01335
01336 #ifndef __cplusplus
01338     typedef SceneMeshLatches_t SceneMeshLatches;
01339 #endif
01340
01341 // SceneMesh types.
01342
01344 typedef struct SceneMesh_t SceneMesh_t;
01345
01346 #ifndef __cplusplus
01348     typedef SceneMesh_t SceneMesh;
01349 #endif
01350
01352 typedef struct SceneMeshes_t SceneMeshes_t;
01353
01354 #ifndef __cplusplus
01356     typedef SceneMeshes_t SceneMeshes;
01357 #endif
01358
01359 // Application types and structures.
01360
01362 typedef struct ApplicationConfiguration
01363 {
01365     Point size;
01366
01368     ApplicationWindowState state;
01369
01372     Point position;
01373
01375     union
01376     {
01378         struct
01379         {
01381             UntypedHandle handleInstance;
01382
01384             UntypedHandle handleIconSmall;
01385
01387             UntypedHandle handleIconBig;
01388
01390             char windowClassName[32];
01391         } win;
01392
01394         struct
01395         {
01397             uint8_t renderingType;
01398
01400             int32_t parameterCount;
01401
01403             char** parameterStrings;
01404         } nx;
01405     } startup;
01406 } ApplicationConfiguration;
01407
01409 typedef struct Application_t Application_t;
01410
01411 #ifndef __cplusplus
01413     typedef Application_t Application;
01414 #endif
01415
01417 typedef void(*EventFunc)(Application_t* application, void* user);
01418
01420 typedef LibraryBool(*BoolFunc)(Application_t* application, void* user);
01421
01423 typedef void(*MouseFunc)(Application_t* application, MouseEvent event, MouseButton button,
01424     Point const* position, void* user);

```



```

01425
01427 typedef void(*KeyboardFunc)(Application_t* application, KeyEvent event, int32_t virtualKey, uint16_t
    keyCode,
01428     void* user);
01429
01431 typedef void(*StatusFunc)(Application* application, ApplicationEvent event, void* user);
01432
01434 typedef LibraryBool(*EventHookFunc)(Application_t* application, void const* event, void* user);
01435
01438 typedef struct ApplicationEvents
01439 {
01442     BoolFunc eventCreate;
01443
01445     EventFunc eventDestroy;
01446
01448     EventFunc eventRender;
01449
01451     MouseFunc eventMouse;
01452
01454     KeyboardFunc eventKey;
01455
01457     StatusFunc eventStatus;
01458
01460     BoolFunc eventIdle;
01461
01463     EventFunc eventResize;
01464
01466     EventHookFunc eventHook;
01467 } ApplicationEvents;
01468
01469 // ActorCamera types and structures.
01470
01473 typedef struct CameraConstraints
01474 {
01476     Vector4 positionMin;
01477
01479     Vector4 positionMax;
01480
01482     Vector4 rotationMin;
01483
01485     Vector4 rotationMax;
01486 } CameraConstraints;
01487
01490 typedef struct ActorCamera_t ActorCamera_t;
01491
01492 #ifndef __cplusplus
01495     typedef ActorCamera_t ActorCamera;
01496 #endif
01497
01498 // Timer types.
01499
01501 typedef struct Timer_t Timer_t;
01502
01503 #ifndef __cplusplus
01505     typedef Timer_t Timer;
01506 #endif
01507
01508 // Widget types.
01509
01511 typedef union WidgetPropertyData
01512 {
01514     int32_t valueInteger;
01515
01517     int64_t valueInt64;
01518
01520     float valueFloat;
01521
01523     double valueReal;
01524
01526     bool valueBool;
01527
01529     PointF valuePoint;
01530
01532     Vector valueVector;
01533
01535     RectF valueRect;
01536
01538     Margins valueMargins;
01539
01541     Color valueColor;
01542
01544     ColorPair valueColorPair;
01545
01547     ColorRect valueColorRect;
01548
01550     uint8_t valueEnum;
01551

```

```

01554   char valueString[184];
01555
01557   FontParameters valueFont;
01558 } WidgetPropertyData;
01559
01561 typedef struct WidgetProperty
01562 {
01564   char name[256];
01565
01567   WidgetPropertyType type;
01568
01570   WidgetPropertyBehavior behavior;
01571
01573   uint16_t attributes;
01574
01579   uint16_t location;
01580
01582   uint16_t reserved;
01583
01585   WidgetPropertyData data;
01586 } WidgetProperty;
01587
01589 typedef struct Widget_t Widget_t;
01590
01591 #ifndef __cplusplus
01593   typedef Widget_t Widget;
01594 #endif
01595
01597 typedef LibraryBool(*WidgetExternalEvent)(struct Widget_t* widget, char const* event, void* user);
01598
01599 #ifdef __cplusplus
01600   } // namespace pxt
01601 #endif

```

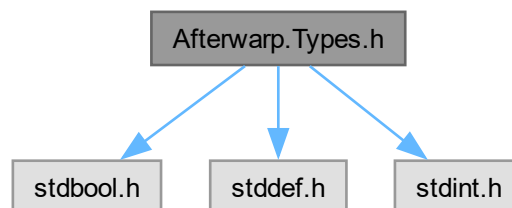
7.5 Afterwarp.Types.h File Reference

```

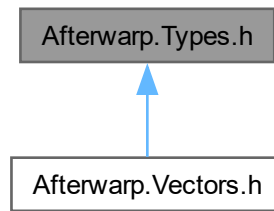
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>

```

Include dependency graph for Afterwarp.Types.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [ColorPair](#)
- struct [ColorRect](#)
- struct [FloatColorRGB](#)
- struct [FloatColor](#)
- struct [Point](#)
2D integer point (or vector).
- struct [PointF](#)
2D floating-point vector.
- struct [Vector](#)
3D floating-point vector.
- struct [Vector4](#)
4-component (XYZ + W) floating-point vector.
- struct [Matrix3x2](#)
3x2 matrix representing rotation and translation in context of 2D graphics.
- struct [Matrix](#)
4x4 matrix representing transformation information in context of 3D graphics.
- struct [Quaternion](#)
3D quaternion.
- struct [Rect](#)
Rectangle defined by integer top and left margins, width and height.
- struct [RectF](#)
Rectangle defined by floating-point top and left margins, width and height.
- struct [Quad](#)
Floating-point quadrilateral defined by four vertices in clockwise order starting from top/left.
- struct [Margins](#)
[Margins](#) defined by top, left, right and bottom edge paddings.

Typedefs

- typedef uint8_t [PixelFormat](#)
Defines how individual pixels and their colors are encoded in images and textures.
- typedef uint8_t [AlphaFormatRequest](#)
Defines how alpha-channel should be handled in the loaded image.
- typedef uint8_t [DeviceTechnology](#)
Type of graphics technology used in application.
- typedef uint8_t [DevicePlatform](#)
Type of OS platform the application is running on.
- typedef uint8_t [TriangleFace](#)
Type of triangle face that should have processing applied to.
- typedef uint8_t [ComparisonFunc](#)
Function that defines how depth/stencil operands should be compared.
- typedef uint8_t [StencilOp](#)
Operation that should be performed on stencil value.
- typedef uint8_t [BlendFactor](#)
Factor that determines how alpha-blending operand is considered.
- typedef uint8_t [BlendOp](#)
Operation that should be performed between two blending operands.
- typedef uint8_t [BufferDataTypes](#)
Type of data that a generic buffer contains.
- typedef uint8_t [BufferAccessTypes](#)
Type of access that is performed with mesh buffer.
- typedef uint8_t [PrimitiveTopology](#)
Rendering primitive topology.
- typedef uint16_t [ElementFormat](#)
Format in which a single value of the given element is represented.
- typedef uint8_t [ShaderType](#)
Type of shader in graphics rendering pipeline.
- typedef uint8_t [ShaderElement](#)
Type that characterizes shader element.
- typedef uint8_t [ComputeTextureAccess](#)
Compute program texture access type.
- typedef uint8_t [TextureFilter](#)
Filtering mode that defines how colors are sampled from the texture.
- typedef uint8_t [TextureAddress](#)
Addressing mode that defines how texture coordinates outside of [0, 1] range are treated.
- typedef uint8_t [TextureType](#)
Type of texture that defines how it is composed.
- typedef uint8_t [BlendingEffect](#)
Blending effect that defines how primitives are rendered on drawing surface.
- typedef uint8_t [CanvasContextState](#)
One of possible defined context states that can be pre-configured by the canvas.
- typedef uint8_t [SuperSampleSDF](#)
Type of super-sampling used for rendering Signed Distance Field (SDF).
- typedef uint8_t [PathJoint](#)
Type of joint for connecting path lines.
- typedef uint8_t [LineCaps](#)
Type of caps that covers line end points.
- typedef uint8_t [PointShape](#)

- Shape of point primitive.*

 - typedef uint8_t [FontWeight](#)

Weight of the font (apparent thickness of glyphs).
- typedef uint8_t [FontStretch](#)

Relative width of the font.
- typedef uint8_t [FontSlant](#)

Font slant styles.
- typedef uint8_t [FontBorder](#)

Border that outlines text glyph's shapes.
- typedef uint8_t [FontAttributes](#)

Font attribute bits that define some visual characteristics such as underline and strikeout.
- typedef uint8_t [TextAlignment](#)

Text alignment when drawing with certain functions.
- typedef uint8_t [ColorDitheringFormat](#)

Format to be used as target for color dithering.
- typedef uint8_t [SelectionHighlightTextureType](#)

Type of texture provided by selection highlight module.
- typedef uint8_t [FogFormula](#)

Defines type of formula used in fog calculation.
- typedef uint8_t [DepthFogDistance](#)

Defines how distance is calculated for depth fog.
- typedef uint8_t [TechniqueLighting](#)

Lighting technique that defines what type of lighting formula and/or BDRF is being used.
- typedef uint8_t [TechniqueShadows](#)

Shadow rendering technique.
- typedef uint8_t [ClustersCullingMode](#)

Light culling mode for clustered shading.
- typedef uint8_t [TextureFidelity](#)

Determines the choice of pixel formats for the container.
- typedef uint8_t [TextureCabinetType](#)

Texture type used in the deferred shading scene.
- typedef uint8_t [TextureCabinetPass](#)

Defines type of rendering pass that is currently being performed.
- typedef uint8_t [TextureCabinetFilterType](#)

Filter type that defines how textures are processed.
- typedef uint8_t [SceneTextureType](#)

Type of texture used by the scene.
- typedef uint8_t [SceneSamplerType](#)

Type of sampler used by the scene.
- typedef uint8_t [ModelTransform](#)

Type of transform as used by object models.
- typedef uint8_t [MeshAlign](#)

Axis alignment that determines the placement of mesh around its model.
- typedef uint8_t [ObjectModelViewCompare](#)

Type of comparison applied to an ordered list of objects.
- typedef uint8_t [CameraCommand](#)

Camera movement/rotation command.
- typedef uint8_t [ApplicationWindowState](#)

Application's window state enumeration.
- typedef uint8_t [MouseEvent](#)

Type of mouse event corresponding to the notification.

- typedef uint8_t [MouseButton](#)
Type of mouse button that corresponds to the notification.
- typedef uint8_t [KeyEvent](#)
Type of keyboard event corresponding to the notification.
- typedef uint8_t [ApplicationEvent](#)
Type of application event corresponding to the notification.
- typedef uint8_t [Key](#)
Storage type for portable key constants.
- typedef uint8_t [AppCursor](#)
Application cursor enumeration.
- typedef uint8_t [FileChooserDialog](#)
Type of file chooser dialog type.
- typedef uint8_t [WidgetAlignment](#)
Alignment of the widget respective its parent.
- typedef uint8_t [WidgetPropertyType](#)
Data type of a widget property.
- typedef uint8_t [WidgetPropertyBehavior](#)
Behavior of the widget property, which can indicate its lifetime or whether the property is an event.
- typedef uint8_t [WidgetManagerTextureType](#)
Texture type used by widget manager for the compositioning of the rendered UI.
- typedef uint32_t [Color](#)
- typedef struct [ColorPair](#) [ColorPair](#)
- typedef struct [ColorRect](#) [ColorRect](#)
- typedef struct [FloatColorRGB](#) [FloatColorRGB](#)
- typedef struct [FloatColor](#) [FloatColor](#)
- typedef struct [Point](#) [Point](#)
2D integer point (or vector).
- typedef struct [PointF](#) [PointF](#)
2D floating-point vector.
- typedef struct [Vector](#) [Vector](#)
3D floating-point vector.
- typedef struct [Vector4](#) [Vector4](#)
4-component (XYZ + W) floating-point vector.
- typedef struct [Matrix3x2](#) [Matrix3x2](#)
3x2 matrix representing rotation and translation in context of 2D graphics.
- typedef struct [Matrix](#) [Matrix](#)
4x4 matrix representing transformation information in context of 3D graphics.
- typedef struct [Quaternion](#) [Quaternion](#)
3D quaternion.
- typedef struct [Rect](#) [Rect](#)
Rectangle defined by integer top and left margins, width and height.
- typedef struct [RectF](#) [RectF](#)
Rectangle defined by floating-point top and left margins, width and height.
- typedef struct [Quad](#) [Quad](#)
Floating-point quadrilateral defined by four vertices in clockwise order starting from top/left.
- typedef struct [Margins](#) [Margins](#)
[Margins](#) defined by top, left, right and bottom edge paddings.

Enumerations

- enum {
[PIXEL_FORMAT_UNKNOWN](#) , [PIXEL_FORMAT_R8](#) , [PIXEL_FORMAT_RG8](#) , [PIXEL_FORMAT_RGBA8](#) ,
[PIXEL_FORMAT_R16](#) , [PIXEL_FORMAT_RG16](#) , [PIXEL_FORMAT_RGBA16](#) , [PIXEL_FORMAT_R8S](#) ,
[PIXEL_FORMAT_RG8S](#) , [PIXEL_FORMAT_RGBA8S](#) , [PIXEL_FORMAT_R16S](#) , [PIXEL_FORMAT_RG16S](#) ,
[PIXEL_FORMAT_RGBA16S](#) , [PIXEL_FORMAT_R8U](#) , [PIXEL_FORMAT_RG8U](#) , [PIXEL_FORMAT_RGBA8U](#)
,
[PIXEL_FORMAT_R16U](#) , [PIXEL_FORMAT_RG16U](#) , [PIXEL_FORMAT_RGBA16U](#) , [PIXEL_FORMAT_R32U](#)
,
[PIXEL_FORMAT_RG32U](#) , [PIXEL_FORMAT_RGBA32U](#) , [PIXEL_FORMAT_R8I](#) , [PIXEL_FORMAT_RG8I](#) ,
[PIXEL_FORMAT_RGBA8I](#) , [PIXEL_FORMAT_R16I](#) , [PIXEL_FORMAT_RG16I](#) , [PIXEL_FORMAT_RGBA16I](#)
,
[PIXEL_FORMAT_R32I](#) , [PIXEL_FORMAT_RG32I](#) , [PIXEL_FORMAT_RGBA32I](#) , [PIXEL_FORMAT_R16F](#) ,
[PIXEL_FORMAT_RG16F](#) , [PIXEL_FORMAT_RGBA16F](#) , [PIXEL_FORMAT_R32F](#) , [PIXEL_FORMAT_RG32F](#)
,
[PIXEL_FORMAT_RGBA32F](#) , [PIXEL_FORMAT_RG11B10F](#) , [PIXEL_FORMAT_RGB9E5F](#) , [PIXEL_FORMAT_RGB10A2](#)
,
[PIXEL_FORMAT_RGB10A2U](#) , [PIXEL_FORMAT_RGBX8](#) , [PIXEL_FORMAT_RGBA8_SRGB](#) , [PIXEL_FORMAT_BGRA8](#)
,
[PIXEL_FORMAT_BGRX8](#) , [PIXEL_FORMAT_BGRA8_SRGB](#) , [PIXEL_FORMAT_BGR10A2](#) , [PIXEL_FORMAT_BGR10X2](#)
,
[PIXEL_FORMAT_BGRA4](#) , [PIXEL_FORMAT_BGRX4](#) , [PIXEL_FORMAT_B5G6R5](#) , [PIXEL_FORMAT_BGR5A1](#)
,
[PIXEL_FORMAT_BGR5X1](#) , [PIXEL_FORMAT_RGB8](#) , [PIXEL_FORMAT_BGR8](#) , [PIXEL_FORMAT_A8](#) ,
[PIXEL_FORMAT_L8](#) , [PIXEL_FORMAT_L16](#) , [PIXEL_FORMAT_LA4](#) , [PIXEL_FORMAT_LA8](#) ,
[PIXEL_FORMAT_I8](#) , [PIXEL_FORMAT_R_BC](#) , [PIXEL_FORMAT_RG_BC](#) , [PIXEL_FORMAT_RGB_BC](#) ,
[PIXEL_FORMAT_RGBA_BC](#) , [PIXEL_FORMAT_RGB_BC_SRGB](#) , [PIXEL_FORMAT_RGBA_BC_SRGB](#) ,
[PIXEL_FORMAT_D16](#) ,
[PIXEL_FORMAT_D24S8](#) , [PIXEL_FORMAT_D32F](#) , [PIXEL_FORMAT_D32S8F](#) }
Pixel format enumeration that can be passed as one of [PixelFormat](#) values.
- enum { [ALPHA_FORMAT_REQUEST_DONT_CARE](#) , [ALPHA_FORMAT_REQUEST_NON_PREMULTIPLIED](#)
, [ALPHA_FORMAT_REQUEST_PREMULTIPLIED](#) }
Alpha-channel format request enumeration that can be passed as one of [AlphaFormatRequest](#) values.
- enum { [LEGACY_BIT_MISSING_DEPTH_TEXTURE](#) = 0x00000001 , [LEGACY_BIT_DEPTH_FORMAT_RAWZ](#)
= 0x00000002 , [LEGACY_BIT_DIFFERENT_BITDEPTH_MRT](#) = 0x00000004 }
- enum { [DEVICE_CLEAR_LAYER_COLOR](#) = 0x01 , [DEVICE_CLEAR_LAYER_DEPTH](#) = 0x02 ,
[DEVICE_CLEAR_LAYER_STENCIL](#) = 0x04 }
Type of surface layer that should be cleared.
- enum { [DEVICE_ATTRIBUTE_DEBUG](#) = 0x01 , [DEVICE_ATTRIBUTE_SOFTWARE](#) = 0x02 , [DEVICE_ATTRIBUTE_LEGACY](#)
= 0x04 , [DEVICE_ATTRIBUTE_LIMITED_EXTENSIONS](#) = 0x80 }
Device attributes that define its behavior.
- enum {
[DEVICE_BEHAVIOR_PER_SAMPLE_SHADING](#) = 0x00000001 , [DEVICE_BEHAVIOR_DEPTH_CLIP_NEGATIVE](#)
= 0x00000002 , [DEVICE_BEHAVIOR_COMPUTE](#) = 0x00000004 , [DEVICE_BEHAVIOR_TESSELLATION](#) =
0x00000008 ,
[DEVICE_BEHAVIOR_VARIABLE_RATE_REFRESH](#) = 0x00000100 , [DEVICE_BEHAVIOR_POST_DEPTH_COVERAGE](#)
= 0x00000200 , [DEVICE_BEHAVIOR_FORCE_BUFFER_UNBIND](#) = 0x00001000 , [DEVICE_BEHAVIOR_DEPTH_CLEAR_BA](#)
= 0x00002000 ,
[DEVICE_BEHAVIOR_SIGNED_NORM_INT_FORMAT_BUGGED](#) = 0x00004000 }
Device behavior attributes that define the rendering code path.
- enum {
[DEVICE_BARRIER_SHADER_BUFFER](#) = 0x00000001 , [DEVICE_BARRIER_TEXTURE_FETCH](#) =
0x00000002 , [DEVICE_BARRIER_SHADER_IMAGE_ACCESS](#) = 0x00000004 , [DEVICE_BARRIER_TEXTURE_UPDATE](#)
= 0x00000008 ,
[DEVICE_BARRIER_BUFFER_UPDATE](#) = 0x00000010 , [DEVICE_BARRIER_DRAWABLES](#) = 0x00000020
, [DEVICE_BARRIER_ATOMIC_COUNTER](#) = 0x00000040 , [DEVICE_BARRIER_STRUCTURED](#) =
0x00000080 }

Device memory barrier cumulative bits.

- enum {
[DEVICE_TECHNOLOGY_UNKNOWN](#) , [DEVICE_TECHNOLOGY_DIRECT3D](#) , [DEVICE_TECHNOLOGY_OPENGL](#)
[, DEVICE_TECHNOLOGY_OPENGL_ES](#) ,
[DEVICE_TECHNOLOGY_VULKAN](#) , [DEVICE_TECHNOLOGY_METAL](#) , [DEVICE_TECHNOLOGY_WEBGL](#)
[, DEVICE_TECHNOLOGY_SOFTWARE](#) ,
[DEVICE_TECHNOLOGY_PROPRIETARY](#) }

Graphics device technology enumeration that can be passed as one of [DeviceTechnology](#) values.

- enum {
[DEVICE_PLATFORM_UNKNOWN](#) , [DEVICE_PLATFORM_WINDOWS](#) , [DEVICE_PLATFORM_LINUX](#) ,
[DEVICE_PLATFORM_UNIX](#) ,
[DEVICE_PLATFORM_OSX](#) , [DEVICE_PLATFORM_IOS](#) , [DEVICE_PLATFORM_ANDROID](#) }

Graphics device platform enumeration that can be passed as one of [DevicePlatform](#) values.

- enum {
[DEVICE_STATE_DEPTH_TEST](#) = 0x0001 , [DEVICE_STATE_DEPTH_WRITE](#) = 0x0002 , [DEVICE_STATE_STENCIL_TEST](#)
= 0x0004 , [DEVICE_STATE_DEPTH_CLIP](#) = 0x0008 ,
[DEVICE_STATE_SCISSOR_CLIP](#) = 0x0010 , [DEVICE_STATE_WIREFRAME](#) = 0x0020 , [DEVICE_STATE_CULL_CLOCKWISE](#)
= 0x0040 , [DEVICE_STATE_MULTISAMPLING](#) = 0x0080 ,
[DEVICE_STATE_LINE_ANTIALIAS](#) = 0x0100 , [DEVICE_STATE_COLOR_WRITE](#) = 0x0200 , [DEVICE_STATE_BLEND_ENABLED](#)
= 0x0400 , [DEVICE_STATE_ALPHA_TO_COVERAGE](#) = 0x0800 ,
[DEVICE_STATE_CUBE_MAP_SEAMLESS](#) = 0x1000 , [DEVICE_STATE_PER_SAMPLE_SHADING](#) =
0x2000 }

State flags that can be combined together to form a rendering operation state.

- enum { [TRIANGLE_FACE_NONE](#) , [TRIANGLE_FACE_BACK](#) , [TRIANGLE_FACE_FRONT](#) , [TRIANGLE_FACE_BOTH](#)
}

Face processing type enumeration that can be passed as one of [TriangleFace](#) values.

- enum {
[COMPARISON_FUNC_ALWAYS](#) , [COMPARISON_FUNC_LESS](#) , [COMPARISON_FUNC_LESS_EQUAL](#) ,
[COMPARISON_FUNC_GREATER](#) ,
[COMPARISON_FUNC_GREATER_EQUAL](#) , [COMPARISON_FUNC_EQUAL](#) , [COMPARISON_FUNC_NOT_EQUAL](#)
, [COMPARISON_FUNC_NEVER](#) }

Comparison function enumeration that can be passed as one of [ComparisonFunc](#) values.

- enum {
[STENCIL_OP_KEEP](#) , [STENCIL_OP_ZERO](#) , [STENCIL_OP_REPLACE](#) , [STENCIL_OP_INVERT](#) ,
[STENCIL_OP_INCREMENT](#) , [STENCIL_OP_DECREMENT](#) , [STENCIL_OP_INCREMENT_WARP](#) ,
[STENCIL_OP_DECREMENT_WARP](#) }

Stencil operation enumeration that can be passed as one of [StencilOp](#) values.

- enum {
[BLEND_FACTOR_ONE](#) , [BLEND_FACTOR_ZERO](#) , [BLEND_FACTOR_SOURCE_COLOR](#) , [BLEND_FACTOR_INV_SOURCE_COLOR](#)
, [BLEND_FACTOR_SOURCE_ALPHA](#) , [BLEND_FACTOR_INV_SOURCE_ALPHA](#) , [BLEND_FACTOR_DEST_COLOR](#)
, [BLEND_FACTOR_INV_DEST_COLOR](#) ,
[BLEND_FACTOR_DEST_ALPHA](#) , [BLEND_FACTOR_INV_DEST_ALPHA](#) , [BLEND_FACTOR_SOURCE_ALPHA_SAT](#)
, [BLEND_FACTOR_CONSTANT_COLOR](#) ,
[BLEND_FACTOR_INV_CONSTANT_COLOR](#) , [BLEND_FACTOR_CONSTANT_ALPHA](#) , [BLEND_FACTOR_INV_CONSTANT_ALPHA](#)
, [BLEND_FACTOR_SOURCE_COLOR_1](#) ,
[BLEND_FACTOR_INV_SOURCE_COLOR_1](#) , [BLEND_FACTOR_SOURCE_ALPHA_1](#) , [BLEND_FACTOR_INV_SOURCE_ALPHA_1](#)
}

Blending factor enumeration that can be passed as one of [BlendFactor](#) values.

- enum {
[BLEND_OP_ADD](#) , [BLEND_OP_SUBTRACT](#) , [BLEND_OP_INV_SUBTRACT](#) , [BLEND_OP_MINIMUM](#) ,
[BLEND_OP_MAXIMUM](#) }

Blending operation enumeration that can be passed as one of [BlendOp](#) values.

- enum {
[BUFFER_DATA_TYPE_VERTEX](#) , [BUFFER_DATA_TYPE_INDEX](#) , [BUFFER_DATA_TYPE_CONSTANT](#) ,
[BUFFER_DATA_TYPE_TYPED](#) ,


```

BUFFER_DATA_TYPE_RW_TYPED , BUFFER_DATA_TYPE_STRUCTURED , BUFFER_DATA_TYPE_RW_STRUCTURED
}

```

Buffer data type enumeration that can be passed as one of [BufferData](#) values.

- enum {
[BUFFER_ACCESS_TYPE_DEFAULT](#) , [BUFFER_ACCESS_TYPE_STATIC](#) , [BUFFER_ACCESS_TYPE_DYNAMIC](#)
, [BUFFER_ACCESS_TYPE_COMPUTE](#) ,
[BUFFER_ACCESS_TYPE_STAGING](#) }

Buffer access type enumeration that can be passed as one of [BufferAccess](#) values.

- enum {
[PRIMITIVE_TOPOLOGY_UNKNOWN](#) , [PRIMITIVE_TOPOLOGY_POINTS](#) , [PRIMITIVE_TOPOLOGY_LINES](#)
, [PRIMITIVE_TOPOLOGY_LINE_STRIP](#) ,
[PRIMITIVE_TOPOLOGY_TRIANGLES](#) , [PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP](#) , [PRIMITIVE_TOPOLOGY_LINES_ADJACENCY](#)
, [PRIMITIVE_TOPOLOGY_LINE_STRIP_ADJACENCY](#) ,
[PRIMITIVE_TOPOLOGY_TRIANGLES_ADJACENCY](#) , [PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_ADJACENCY](#)
}

Primitive topology enumeration that can be passed as one of [PrimitiveTopology](#) values.

- enum {
[ELEMENT_FORMAT_UNDEFINED](#) , [ELEMENT_FORMAT_FLOAT](#) , [ELEMENT_FORMAT_HALF_FLOAT](#) ,
[ELEMENT_FORMAT_DOUBLE](#) ,
[ELEMENT_FORMAT_INT](#) , [ELEMENT_FORMAT_UNSIGNED_INT](#) , [ELEMENT_FORMAT_SHORT](#) ,
[ELEMENT_FORMAT_UNSIGNED_SHORT](#) ,
[ELEMENT_FORMAT_BYTE](#) , [ELEMENT_FORMAT_UNSIGNED_BYTE](#) , [ELEMENT_FORMAT_FLOAT_11_11_10](#)
}

Element format enumeration that can be passed as one of [ElementFormat](#) values.

- enum {
[SHADER_TYPE_UNDEFINED](#) , [SHADER_TYPE_FRAGMENT](#) , [SHADER_TYPE_VERTEX](#) , [SHADER_TYPE_GEOMETRY](#)
, [SHADER_TYPE_COMPUTE](#) , [SHADER_TYPE_HULL](#) , [SHADER_TYPE_DOMAIN](#) }

Shader type enumeration that can be passed as one of [ShaderType](#) values.

- enum {
[SHADER_ELEMENT_UNDEFINED](#) , [SHADER_ELEMENT_TEXTURE](#) , [SHADER_ELEMENT_CONSTANT_BUFFER](#)
, [SHADER_ELEMENT_TYPED_BUFFER](#) ,
[SHADER_ELEMENT_RW_TYPED_BUFFER](#) , [SHADER_ELEMENT_STRUCTURED_BUFFER](#) , [SHADER_ELEMENT_RW_STRUCTURED_BUFFER](#)
}

Shader element enumeration that can be passed as one of [ShaderElement](#) values.

- enum { [VERTEX_CHANNEL_INDEX_BUFFER](#) = 0x7FFFFFFF , [VERTEX_CHANNEL_NO_BUFFERS](#) =
0x7FFFFFFE , [SHADER_INDEX_UNDEFINED](#) = 0x7FFFFFFF , [VERTEX_CHANNEL_NORMALIZED](#) =
0x80000000 }

- enum { [COMPUTE_TEXTURE_ACCESS_READ](#) , [COMPUTE_TEXTURE_ACCESS_WRITE](#) , [COMPUTE_TEXTURE_ACCESS_READ_WRITE](#) }

Compute program texture access type that can be passed as one of [ComputeTextureAccess](#) values.

- enum { [TEXTURE_FILTER_NONE](#) , [TEXTURE_FILTER_NEAREST](#) , [TEXTURE_FILTER_LINEAR](#) ,
[TEXTURE_FILTER_ANISOTROPIC](#) }

Texture filtering enumeration that can be passed as one of [TextureFilter](#) values.

- enum {
[TEXTURE_ADDRESS_WRAP](#) , [TEXTURE_ADDRESS_CLAMP](#) , [TEXTURE_ADDRESS_MIRROR](#) ,
[TEXTURE_ADDRESS_MIRROR_ONCE](#) ,
[TEXTURE_ADDRESS_BORDER](#) }

Texture addressing enumeration that can be passed as one of [TextureAddress](#) values.

- enum {
[TEXTURE_ATTRIBUTE_MIPMAPPING](#) = 0x01 , [TEXTURE_ATTRIBUTE_DRAWABLE](#) = 0x02 ,
[TEXTURE_ATTRIBUTE_DYNAMIC](#) = 0x04 , [TEXTURE_ATTRIBUTE_PREMULTIPLIED_ALPHA](#) = 0x08
, [TEXTURE_ATTRIBUTE_SHADER_RESOURCE](#) = 0x10 ,
[TEXTURE_COMPUTE](#) = 0x40 , [TEXTURE_SCRATCH](#) = 0x8000 }

Attribute that defines special behavior and/or unique characteristics of the texture.

- enum { `TEXTURE_TYPE_SURFACE` , `TEXTURE_TYPE_CUBE_MAP` , `TEXTURE_TYPE_VOLUME` }
Texture type enumeration that can be passed as one of [TextureType](#) values.
- enum {
`BLENDING_EFFECT_UNDEFINED` = 0 , `BLENDING_EFFECT_NORMAL` , `BLENDING_EFFECT_SHADOW`
, `BLENDING_EFFECT_ADD` ,
`BLENDING_EFFECT_SUBTRACT` , `BLENDING_EFFECT_MULTIPLY` , `BLENDING_EFFECT_INVERSE_MULTIPLY`
, `BLENDING_EFFECT_SOURCE_COLOR` ,
`BLENDING_EFFECT_SOURCE_COLOR_ADD` , `BLENDING_EFFECT_COPY` = 254 , `BLENDING_EFFECT_CUSTOM`
= 255 }
Blending effect enumeration that can be passed as one of [BlendingEffect](#) values.
- enum { `CANVAS_CONTEXT_STATE_FLAT_SCENE` , `CANVAS_CONTEXT_STATE_FLAT_TEXT` ,
`CANVAS_CONTEXT_STATE_PROJECTED` , `CANVAS_CONTEXT_STATE_UNDEFINED` = `UINT8_MAX`
} }
Canvas context state enumeration that can be passed as one of [CanvasContextState](#) values.
- enum {
`CANVAS_ATTRIBUTE_PROJECTED` = 0x01 , `CANVAS_ATTRIBUTE_SDF` = 0x02 , `CANVAS_ATTRIBUTE_OUTLINE_SDF`
= 0x04 , `CANVAS_ATTRIBUTE_CUBIC` = 0x08 ,
`CANVAS_ATTRIBUTE_COLOR_ADJUST` = 0x10 , `CANVAS_ATTRIBUTE_TRANSFORM` = 0x20 }
Canvas attribute flags that can be combined together.
- enum { `SUPER_SAMPLE_SDF_NO_SUPER_SAMPLE` , `SUPER_SAMPLE_SDF_SUPER_SAMPLE_4X` ,
`SUPER_SAMPLE_SDF_SUPER_SAMPLE_16X` }
Super-sample SDF type enumeration that can be passed as one of [SuperSampleSDF](#) values.
- enum { `IMAGE_REGION_FLIP` = 0x01 , `IMAGE_REGION_MIRROR` = 0x02 , `IMAGE_REGION_ROTATE` =
0x04 }
Image Region modifier flags that can be combined together.
- enum {
`PATH_JOINT_NONE` , `PATH_JOINT_SIMPLE` , `PATH_JOINT_MITER` , `PATH_JOINT_BEVEL` ,
`PATH_JOINT_ROUND` , `PATH_JOINT_MITER_BEVEL` }
Path joint enumeration that can be passed as one of [PathJoint](#) values.
- enum { `LINE_CAPS_BUTT` , `LINE_CAPS_SQUARE` , `LINE_CAPS_ROUND` }
Line caps enumeration that can be passed as one of [LineCaps](#) values.
- enum {
`POINT_SHAPE_SQUARE` , `POINT_SHAPE_ROUND` , `POINT_SHAPE_TRIANGLE` , `POINT_SHAPE_STAR`
, `POINT_SHAPE_CROSS` }
Point shape enumeration that can be passed as one of [PointShape](#) values.
- enum {
`FONT_WEIGHT_THIN` , `FONT_WEIGHT_EXTRA_LIGHT` , `FONT_WEIGHT_LIGHT` , `FONT_WEIGHT_SEMI_LIGHT`
, `FONT_WEIGHT_NORMAL` , `FONT_WEIGHT_MEDIUM` , `FONT_WEIGHT_SEMI_BOLD` , `FONT_WEIGHT_BOLD`
, `FONT_WEIGHT_EXTRA_BOLD` , `FONT_WEIGHT_HEAVY` , `FONT_WEIGHT_EXTRA_HEAVY` }
Font weight enumeration that can be passed as one of [FontWeight](#) values.
- enum {
`FONT_STRETCH_ULTRA_CONDENSED` , `FONT_STRETCH_EXTRA_CONDENSED` , `FONT_STRETCH_CONDENSED`
, `FONT_STRETCH_SEMI_CONDENSED` ,
`FONT_STRETCH_NORMAL` , `FONT_STRETCH_SEMI_EXPANDED` , `FONT_STRETCH_EXPANDED` ,
`FONT_STRETCH_EXTRA_EXPANDED` ,
`FONT_STRETCH_ULTRA_EXPANDED` }
Font stretch enumeration that can be passed as one of [FontStretch](#) values.
- enum { `FONT_SLANT_NONE` , `FONT_SLANT_OBLIQUE` , `FONT_SLANT_ITALIC` }
Font slant styles enumeration that can be passed as one of [FontSlant](#) values.
- enum { `FONT_ATTRIBUTE_UNDERLINE` = 0x01 , `FONT_ATTRIBUTE_STRIKE_OUT` = 0x02 }
Font attribute flags that define some visual characteristics.

- enum { [FONT_BORDER_NONE](#) , [FONT_BORDER_NORMAL](#) , [FONT_BORDER_SEMI_HEAVY](#) , [FONT_BORDER_HEAVY](#) }

Font border enumeration that can be passed as one of [FontBorder](#) values.

- enum { [TEXT_ALIGNMENT_START](#) , [TEXT_ALIGNMENT_MIDDLE](#) , [TEXT_ALIGNMENT_END](#) }

Text alignment enumeration that can be passed as one of [TextAlignment](#) values.

- enum { [COLOR_DITHERING_FORMAT_RGB8](#) , [COLOR_DITHERING_FORMAT_RGB6](#) , [COLOR_DITHERING_FORMAT_R5G6B5](#) , [COLOR_DITHERING_FORMAT_RGBA](#) , [COLOR_DITHERING_FORMAT_RGBA4](#) , [COLOR_DITHERING_FORMAT_RGBA2](#) }

Format enumeration that can be passed as one of [ColorDitheringFormat](#) values.

- enum { [SELECTION_HIGHLIGHT_TEXTURE_HIGHLIGHT](#) , [SELECTION_HIGHLIGHT_TEXTURE_HIGHLIGHT_MASK](#) }

Texture type enumeration that can be passed as one of [SelectionHighlightTextureType](#) values.

- enum { [FOG_FORMULA_LINEAR](#) , [FOG_FORMULA_EXPONENTIAL](#) , [FOG_FORMULA_GROUND](#) }

Fog formula enumeration that can be passed as one of [FogFormula](#) values.

- enum { [DEPTH_FOG_DISTANCE_Z_BASED](#) , [DEPTH_FOG_DISTANCE_EYE_RELATIVE](#) }

Depth fog distance enumeration that can be passed as one of [DepthFogDistance](#) values.

- enum { [TECHNIQUE_LIGHTING_PHONG](#) , [TECHNIQUE_LIGHTING_MINNAERT](#) , [TECHNIQUE_LIGHTING_COOK_TORRANCE](#) , [TECHNIQUE_LIGHTING_OREN_NAYER](#) }

Lighting technique enumeration that can be passed as one of [TechniqueLighting](#) values.

- enum { [TECHNIQUE_SHADOWS_NONE](#) , [TECHNIQUE_SHADOWS_ESM](#) , [TECHNIQUE_SHADOWS_ESM_WARP](#) , [TECHNIQUE_SHADOWS_EVSM](#) }

Shadow technique enumeration that can be passed as one of [TechniqueShadows](#) values.

- enum { [CLUSTERS_CULLING_MODE_QUALITY](#) , [CLUSTERS_CULLING_MODE_PERFORMANCE](#) }

Light culling mode enumeration that can be passed as one of [ClustersCullingMode](#) values.

- enum { [TEXTURE_FIDELITY_LOW](#) , [TEXTURE_FIDELITY_MEDIUM](#) , [TEXTURE_FIDELITY_HIGH](#) , [TEXTURE_FIDELITY_EXTREME](#) }

Texture fidelity enumeration that can be passed as one of [TextureFidelity](#) values.

- enum { [TEXTURE_CABINET_TYPE_DEPTH](#) = 0 , [TEXTURE_CABINET_TYPE_NORMALS](#) = 1 , [TEXTURE_CABINET_TYPE_COLORS](#) = 4 , [TEXTURE_CABINET_TYPE_COLOR](#) = 5 , [TEXTURE_CABINET_TYPE_LINEAR_DEPTHS](#) = 11 }

Texture cabinet type enumeration that can be passed as one of [TextureCabinetType](#) values.

- enum { [TEXTURE_CABINET_PASS_COLOR](#) , [TEXTURE_CABINET_PASS_GLASSY](#) , [TEXTURE_CABINET_PASS_DEPTH](#) }

Texture cabinet pass enumeration that can be passed as one of [TextureCabinetPass](#) values.

- enum { [TEXTURE_CABINET_FILTER_TYPE_OCCLUSION](#) , [TEXTURE_CABINET_FILTER_TYPE_BLOOM](#) , [TEXTURE_CABINET_FILTER_TYPE_GLASSY](#) , [TEXTURE_CABINET_FILTER_TYPE_GLASSY_FOG](#) }

Texture cabinet filter type enumeration that can be passed as one of [TextureCabinetFilterType](#) values.

- enum { [TEXTURE_CABINET_ATTRIBUTE_AMBIENT_OCCLUSION](#) = 0x0001 , [TEXTURE_CABINET_ATTRIBUTE_AMBIENT_OCCLUSION](#) = 0x0002 , [TEXTURE_CABINET_ATTRIBUTE_BLOOM](#) = 0x0004 , [TEXTURE_CABINET_ATTRIBUTE_BLOOM_DOWNSCALE](#) = 0x0008 , [TEXTURE_CABINET_ATTRIBUTE_BLOOM_CUBIC](#) = 0x0010 , [TEXTURE_CABINET_ATTRIBUTE_LINEAR_DEPTHS](#) = 0x0020 , [TEXTURE_CABINET_ATTRIBUTE_GLASSY](#) = 0x0040 , [TEXTURE_CABINET_ATTRIBUTE_GLASSY_FAST](#) = 0x0080 , [TEXTURE_CABINET_ATTRIBUTE_GLASSY_FOG](#) = 0x0100 , [TEXTURE_CABINET_ATTRIBUTE_GLASSY_FROSTED](#) = 0x0200 }

Cumulative rendering attributes that can be passed to [TextureCabinetAttributes](#).

- enum { [SCENE_ATTRIBUTE_INSTANCING](#) = 0x01 , [SCENE_ATTRIBUTE_NORMALS](#) = 0x02 , [SCENE_ATTRIBUTE_COLORING](#) = 0x04 , [SCENE_ATTRIBUTE_TEXTUREING_CUBIC](#) = 0x08 , [SCENE_ATTRIBUTE_SHADOWS_CUBIC](#) = 0x10 , [SCENE_ATTRIBUTE_GLASSY](#) = 0x20 , [SCENE_ATTRIBUTE_SINGLE_PASS](#) = 0x40 , [SCENE_ATTRIBUTE_DEPTH_PREPASS](#) = 0x80 , [SCENE_ATTRIBUTE_LINEAR_DEPTHS](#) = 0x100 , [SCENE_ATTRIBUTE_MODELING](#) = 0x1000 }

Cumulative rendering attributes that can be passed to [SceneAttributes](#).

- enum { [SCENE_TEXTURE_TYPE_ALBEDO](#) , [SCENE_TEXTURE_TYPE_NORMAL_MAP](#) , [SCENE_TEXTURE_TYPE_PARALLAX_MAP](#) }

Scene texture type enumeration that can be passed as one of [SceneTextureType](#) values.

- enum { [SCENE_SAMPLER_TYPE_ALBEDO](#) , [SCENE_SAMPLER_TYPE_NORMAL_MAP](#) , [SCENE_SAMPLER_TYPE_PARALLAX_MAP](#) , [SCENE_SAMPLER_TYPE_SHADOW_MAP](#) }

Scene sampler type enumeration that can be passed as one of [SceneSamplerType](#) values.

- enum { [MESH_META_TAG_TYPE_INDETERMINATE](#) = 0 , [MESH_META_TAG_TYPE_GEOMETRY](#) = 1 , [MESH_META_TAG_TYPE_OBJECT](#) = 2 }

Type of the mesh meta-tag.

- enum { [SCENE_MESH_TEXTURE_DIFFUSE](#) = 0 , [SCENE_MESH_TEXTURE_NORMALS](#) = 1 , [SCENE_MESH_TEXTURE_PARALLAX](#) = 2 }

Type of texture used in scene mesh material.

- enum { [SCENE_MESH_LATCH_TYPE_JOINT](#) , [SCENE_MESH_LATCH_TYPE_WAYPOINT](#) }

Type of latch used in the 3D scene mesh.

- enum { [SCENE_MESH_VERTEX_ELEMENTS_INDEX_UNDEFINED](#) = 0xFF }

Enumeration values that can be passed to scene mesh creation functions.

- enum { [AUTO_DRAW_OPTION_POST_UNBIND](#) = 0x01 , [AUTO_DRAW_OPTION_RESET_TEXTURES](#) = 0x02 , [AUTO_DRAW_OPTION_MATERIAL_IGNORE](#) = 0x04 , [AUTO_DRAW_OPTION_MATERIAL_TRANSPARENCY](#) = 0x08 , [AUTO_DRAW_OPTION_MATERIAL_SKIP_TRANSPARENT](#) = 0x10 , [AUTO_DRAW_OPTION_MATERIAL_SKIP_OPAQUE](#) = 0x20 , [AUTO_DRAW_OPTION_OBJECT_SKIP_TRANSPARENT](#) = 0x40 , [AUTO_DRAW_OPTION_OBJECT_SKIP_NON_TRANSPARENT](#) = 0x80 , [AUTO_DRAW_OPTION_OBJECT_SKIP_HAVING_MATERIALS](#) = 0x100 , [AUTO_DRAW_OPTION_OBJECT_SKIP_NOT_HAVING_MATERIALS](#) = 0x200 , [AUTO_DRAW_OPTION_OBJECT_HIGHLIGHTED](#) = 0x400 , [AUTO_DRAW_OPTION_OBJECT_DISABLE_INSTANCING](#) = 0x800 }

Cumulative options that can be passed to one of [xxAutoDraw](#) functions.

- enum { [MODEL_TRANSFORM_LOCAL](#) , [MODEL_TRANSFORM_LOCAL_MODEL](#) , [MODEL_TRANSFORM_LOCAL_VOLUME](#) , [MODEL_TRANSFORM_GLOBAL](#) , [MODEL_TRANSFORM_GLOBAL_MODEL](#) , [MODEL_TRANSFORM_GLOBAL_VOLUME](#) }

Object model transform type enumeration that can be passed as one of [ModelTransform](#) values.

- enum { [MESH_ALIGN_POSITIVE](#) , [MESH_ALIGN_ORIGIN](#) , [MESH_ALIGN_NEGATIVE](#) , [MESH_ALIGN_UNALIGNED](#) }

Axis alignment enumeration that can be passed as one of [MeshAlign](#) values.

- enum { [OBJECT_MODEL_COMPARE_DEPTH](#) , [OBJECT_MODEL_COMPARE_DEPTH_REVERSE](#) , [OBJECT_MODEL_COMPARE_MESHES](#) , [OBJECT_MODEL_COMPARE_OBJECTS](#) }

Object model comparison enumeration that can be passed as one of [ObjectModelViewCompare](#) values.

- enum { [OBJECT_MODEL_ATTRIBUTE_VISIBLE](#) = 0x01 , [OBJECT_MODEL_ATTRIBUTE_SELECTABLE](#) = 0x02 , [OBJECT_MODEL_ATTRIBUTE_TRANSPARENT](#) = 0x04 , [OBJECT_MODEL_ATTRIBUTE_HIERARCHYLESS](#) = 0x08 , [OBJECT_MODEL_ATTRIBUTE_DISABLE_REALIGN](#) = 0x10 , [OBJECT_MODEL_ATTRIBUTE_NON_VIEWABLE](#) = 0x20 }

Cumulative flags for a 3D object model that define its status.

- enum { [MESH_LOADING_OPTION_MATERIAL_VERTEX_COLOR](#) = 0x01 , [MESH_LOADING_OPTION_MATERIAL_VERTEX_COLORS](#) = 0x02 , [MESH_LOADING_OPTION_MATERIAL_TEXTURE_MISSING](#) = 0x80 , [MESH_LOADING_OPTION_EXPORT_NORMALS](#) = 0x100 , [MESH_LOADING_OPTION_EXPORT_TANGENTS](#) = 0x200 , [MESH_LOADING_OPTION_EXPORT_TEXTURE_COORDINATES](#) = 0x400 , [MESH_LOADING_OPTION_EXPORT_COLORS](#) = 0x800 , [MESH_LOADING_OPTION_EXPORT_WEIGHTS](#) = 0x1000 , [MESH_LOADING_OPTION_STRIP_GEOMETRY_NAMES](#) = 0x2000 }

Non-exclusive options passed and/or returned from mesh loading and saving functions.

- enum {
CAMERA_COMMAND_MOVEMENT , CAMERA_COMMAND_DRAGGING , CAMERA_COMMAND_ROTATION
, CAMERA_COMMAND_UPDATE ,
CAMERA_COMMAND_STOP }

Camera movement/rotation command enumeration that can be passed as one of [CameraCommand](#) values.

- enum { APPLICATION_WINDOW_STATE_WINDOWED , APPLICATION_WINDOW_STATE_MINIMIZED ,
APPLICATION_WINDOW_STATE_MAXIMIZED , APPLICATION_WINDOW_STATE_FULLSCREEN }

Application window state enumeration that can be passed as one of [ApplicationWindowState](#) values.

- enum {
MOUSE_EVENT_MOUSE_DOWN , MOUSE_EVENT_MOUSE_MOVE , MOUSE_EVENT_MOUSE_UP ,
MOUSE_EVENT_WHEEL_UP ,
MOUSE_EVENT_WHEEL_DOWN , MOUSE_EVENT_MOUSE_ENTER , MOUSE_EVENT_MOUSE_LEAVE
}

Mouse event type enumeration that can be passed as one of [MouseEvent](#) values.

- enum { MOUSE_BUTTON_NONE , MOUSE_BUTTON_LEFT , MOUSE_BUTTON_RIGHT , MOUSE_BUTTON_MIDDLE
}

Mouse button type enumeration that can be passed as one of [MouseButton](#) values.

- enum { KEY_EVENT_PRESSED , KEY_EVENT_RELEASED , KEY_EVENT_TYPING }

Keyboard event type enumeration that can be passed as one of [KeyEvent](#) values.

- enum {
APPLICATION_EVENT_ACTIVATE , APPLICATION_EVENT_DEACTIVATE , APPLICATION_EVENT_MINIMIZE
, APPLICATION_EVENT_RESTORE ,
APPLICATION_EVENT_APPEARANCE }

Application event type enumeration that can be passed as one of [ApplicationEvent](#) values.

- enum [Key](#) {
KEY_NULL = 0x00 , KEY_HOME = 0x02 , KEY_END = 0x03 , KEY_PRINT_SCREEN = 0x05 ,
KEY_BACKSPACE = 0x08 , KEY_TAB = 0x09 , KEY_LINEFEED = 0x0A , KEY_CLEAR = 0x0B ,
KEY_RETURN = 0x0D , KEY_PAGE_UP = 0x0E , KEY_PAGE_DOWN = 0x0F , KEY_CAPSLOCK = 0x11 ,
KEY_NUM_LOCK = 0x12 , KEY_PAUSE = 0x13 , KEY_SCROLL_LOCK = 0x14 , KEY_SYS_REQ = 0x15 ,
KEY_CANCEL = 0x18 , KEY_INSERT = 0x1A , KEY_ESCAPE = 0x1B , KEY_LEFT = 0x1C ,
KEY_UP = 0x1D , KEY_RIGHT = 0x1E , KEY_DOWN = 0x1F , KEY_SPACE = 0x20 ,
KEY_DELETE = 0x7F , KEY_F1 = 0x80 , KEY_F2 = 0x81 , KEY_F3 = 0x82 ,
KEY_F4 = 0x83 , KEY_F5 = 0x84 , KEY_F6 = 0x85 , KEY_F7 = 0x86 ,
KEY_F8 = 0x87 , KEY_F9 = 0x88 , KEY_F10 = 0x89 , KEY_F11 = 0x8A ,
KEY_F12 = 0x8B , KEY_SHIFT_LEFT = 0x98 , KEY_SHIFT_RIGHT = 0x99 , KEY_CTRL_LEFT = 0x9A ,
KEY_CTRL_RIGHT = 0x9B , KEY_ALT_LEFT = 0x9C , KEY_ALT_RIGHT = 0x9D , KEY_SUPER_LEFT =
0x9E ,
KEY_SUPER_RIGHT = 0x9F , KEY_NUMPAD_0 = 0xA0 , KEY_NUMPAD_1 = 0xA1 , KEY_NUMPAD_2 =
0xA2 ,
KEY_NUMPAD_3 = 0xA3 , KEY_NUMPAD_4 = 0xA4 , KEY_NUMPAD_5 = 0xA5 , KEY_NUMPAD_6 = 0xA6
,
KEY_NUMPAD_7 = 0xA7 , KEY_NUMPAD_8 = 0xA8 , KEY_NUMPAD_9 = 0xA9 , KEY_MULTIPLY = 0xAA ,
KEY_DIVIDE = 0xAB , KEY_ADD = 0xAC , KEY_SUBTRACT = 0xAD , KEY_SEPARATOR = 0xAE ,
KEY_DECIMAL = 0xAF , KEY_SLEEP = 0xC0 , KEY_VOLUME_UP = 0xCC , KEY_VOLUME_DOWN = 0xCD ,
KEY_VOLUME_MUTE = 0xCF , KEY_MEDIA_NEXT_TRACK = 0xD0 , KEY_MEDIA_PREV_TRACK = 0xD1
, KEY_MEDIA_STOP = 0xD2 ,
KEY_MEDIA_PLAY_PAUSE = 0xD3 , KEY_SELECT = 0xE0 , KEY_PRINT = 0xE1 , KEY_EXECUTE = 0xE2
,
KEY_HELP = 0xE3 , KEY_APPS = 0xE4 }

Portable key type enumeration that can be passed as one of [Key](#) values.

- enum {
APP_CURSOR_BLANK = 0x00 , APP_CURSOR_ARROW = 0x01 , APP_CURSOR_TEXT_SELECT ,
APP_CURSOR_TEXT_SELECT_VERTICAL ,
APP_CURSOR_WAIT , APP_CURSOR_ARROW_WAIT , APP_CURSOR_CROSSHAIR , APP_CURSOR_CELL

```
,
APP_CURSOR_LINK_SELECT , APP_CURSOR_NOT_ALLOWED , APP_CURSOR_HELP , APP_CURSOR_MOVE
,
APP_CURSOR_DRAG , APP_CURSOR_DRAGGING , APP_CURSOR_RESIZE_VERT , APP_CURSOR_RESIZE_HORIZ
,
APP_CURSOR_RESIZE1 , APP_CURSOR_RESIZE2 , APP_CURSOR_RESIZE_TOP , APP_CURSOR_RESIZE_BOTTOM
,
APP_CURSOR_RESIZE_LEFT , APP_CURSOR_RESIZE_RIGHT , APP_CURSOR_RESIZE_TOP_LEFT ,
APP_CURSOR_RESIZE_TOP_RIGHT ,
APP_CURSOR_RESIZE_BOTTOM_RIGHT , APP_CURSOR_RESIZE_BOTTOM_LEFT , APP_CURSOR_UNDEFINED
= 255 }
```

Application cursor enumeration that can be passed as one of [AppCursor](#) values.

- enum { [FILE_CHOOSER_DIALOG_OPEN_FILE](#) , [FILE_CHOOSER_DIALOG_SAVE_FILE](#) , [FILE_CHOOSER_DIALOG_DIRECTORY](#) }

File chooser dialog type enumeration that can be passed as one of [FileChooserDialog](#) values.

- enum { [WIDGET_ALIGNMENT_NONE](#) , [WIDGET_ALIGNMENT_CLIENT](#) , [WIDGET_ALIGNMENT_LEFT](#) , [WIDGET_ALIGNMENT_TOP](#) , [WIDGET_ALIGNMENT_RIGHT](#) , [WIDGET_ALIGNMENT_BOTTOM](#) }

Widget alignment enumeration that can be passed as one of [WidgetAlignment](#) values.

- enum { [WIDGET_PROPERTY_TYPE_NONE](#) , [WIDGET_PROPERTY_TYPE_INTEGER](#) , [WIDGET_PROPERTY_TYPE_INT64](#) , [WIDGET_PROPERTY_TYPE_FLOAT](#) , [WIDGET_PROPERTY_TYPE_REAL](#) , [WIDGET_PROPERTY_TYPE_BOOLEAN](#) , [WIDGET_PROPERTY_TYPE_POINT](#) , [WIDGET_PROPERTY_TYPE_VECTOR](#) , [WIDGET_PROPERTY_TYPE_RECT](#) , [WIDGET_PROPERTY_TYPE_MARGINS](#) , [WIDGET_PROPERTY_TYPE_COLOR](#) , [WIDGET_PROPERTY_TYPE_COLOR_PAIR](#) , [WIDGET_PROPERTY_TYPE_COLOR_RECT](#) , [WIDGET_PROPERTY_TYPE_ENUMERATION](#) , [WIDGET_PROPERTY_TYPE_FONT](#) }

Widget property type enumeration that can be passed as one of [WidgetPropertyType](#) values.

- enum { [WIDGET_PROPERTY_BEHAVIOR_NORMAL](#) , [WIDGET_PROPERTY_BEHAVIOR_STATIC](#) , [WIDGET_PROPERTY_BEHAVIOR_VOLATILE](#) , [WIDGET_PROPERTY_BEHAVIOR_EVENT](#) }

Widget property behavior enumeration that can be passed as one of [WidgetPropertyBehavior](#) values.

- enum { [WIDGET_PROPERTY_ATTRIBUTE_ASSIGNED](#) = 0x1000 }

Widget property attribute bitmasks.

- enum { [WIDGET_MANAGER_TEXTURE_TYPE_CANVAS](#) , [WIDGET_MANAGER_TEXTURE_TYPE_SCREEN](#) , [WIDGET_MANAGER_TEXTURE_TYPE_AUXILIARY](#) }

Widget manager texture type enumeration that can be passed as one of [WidgetManagerTextureType](#) values.

- enum [WidgetManagerAttribute](#) { [WIDGET_MANAGER_ATTRIBUTE_COMPOSITION](#) = 0x01 , [WIDGET_MANAGER_ATTRIBUTE_ALPHA](#) = 0x80 }

Widget manager cumulative attribute flags that define its behavior.

7.5.1 Typedef Documentation

7.5.1.1 AlphaFormatRequest

```
typedef uint8_t AlphaFormatRequest
```

Defines how alpha-channel should be handled in the loaded image.

7.5.1.2 AppCursor

```
typedef uint8_t AppCursor
```

Application cursor enumeration.

7.5.1.3 ApplicationEvent

```
typedef uint8_t ApplicationEvent
```

Type of application event corresponding to the notification.

7.5.1.4 ApplicationWindowState

```
typedef uint8_t ApplicationWindowState
```

Application's window state enumeration.

7.5.1.5 BlendFactor

```
typedef uint8_t BlendFactor
```

Factor that determines how alpha-blending operand is considered.

7.5.1.6 BlendingEffect

```
typedef uint8_t BlendingEffect
```

Blending effect that defines how primitives are rendered on drawing surface.

7.5.1.7 BlendOp

```
typedef uint8_t BlendOp
```

Operation that should be performed between two blending operands.

7.5.1.8 BufferAccessType

```
typedef uint8_t BufferAccessType
```

Type of access that is performed with mesh buffer.

7.5.1.9 BufferDataType

```
typedef uint8_t BufferDataType
```

Type of data that a generic buffer contains.

7.5.1.10 CameraCommand

```
typedef uint8_t CameraCommand
```

Camera movement/rotation command.

7.5.1.11 CanvasContextState

```
typedef uint8_t CanvasContextState
```

One of possible defined context states that can be pre-configured by the canvas.

7.5.1.12 ClustersCullingMode

```
typedef uint8_t ClustersCullingMode
```

Light culling mode for clustered shading.

7.5.1.13 Color

```
typedef uint32_t Color
```

Raw (untyped) color value is represented as a raw 32-bit unsigned integer, with components allocated according to ARGB8 format.

7.5.1.14 ColorDitheringFormat

```
typedef uint8_t ColorDitheringFormat
```

Format to be used as target for color dithering.

7.5.1.15 ColorPair

```
typedef struct ColorPair ColorPair
```

A combination of two colors, primarily used for displaying text with the first color being on top and the second being on bottom. The format for specifying colors is defined as ARGB8.

7.5.1.16 ColorRect

```
typedef struct ColorRect ColorRect
```

A rectangle of four colors, primarily used for displaying colored quads, where each color corresponds to top/left, top/right, bottom/right and bottom/left accordingly (clockwise). The format for specifying colors is defined as ARGB8.

7.5.1.17 ComparisonFunc

```
typedef uint8_t ComparisonFunc
```

Function that defines how depth/stencil operands should be compared.

7.5.1.18 ComputeTextureAccess

```
typedef uint8_t ComputeTextureAccess
```

Compute program texture access type.

7.5.1.19 DepthFogDistance

```
typedef uint8_t DepthFogDistance
```

Defines how distance is calculated for depth fog.

7.5.1.20 DevicePlatform

```
typedef uint8_t DevicePlatform
```

Type of OS platform the application is running on.

7.5.1.21 DeviceTechnology

```
typedef uint8_t DeviceTechnology
```

Type of graphics technology used in application.

7.5.1.22 ElementFormat

```
typedef uint16_t ElementFormat
```

Format in which a single value of the given element is represented.

7.5.1.23 FileChooserDialog

```
typedef uint8_t FileChooserDialog
```

Type of file chooser dialog type.

7.5.1.24 FloatColor

```
typedef struct FloatColor FloatColor
```

A special high-precision color value that has each individual component represented as 32-bit floating-point value in range of [0, 1]. Although components may have values outside of aforementioned range, such colors cannot be reliably displayed on the screen.

7.5.1.25 FloatColorRGB

```
typedef struct FloatColorRGB FloatColorRGB
```

A three-component special high-precision color value that has each individual component represented as 32-bit floating-point value in range of [0, 1]. Although components may have values outside of aforementioned range, such colors cannot be reliably displayed on the screen.

7.5.1.26 FogFormula

```
typedef uint8_t FogFormula
```

Defines type of formula used in fog calculation.

7.5.1.27 FontAttributes

```
typedef uint8_t FontAttributes
```

Font attribute bits that define some visual characteristics such as underline and strikeout.

7.5.1.28 FontBorder

```
typedef uint8_t FontBorder
```

Border that outlines text glyph's shapes.

7.5.1.29 FontSlant

```
typedef uint8_t FontSlant
```

Font slant styles.

7.5.1.30 FontStretch

```
typedef uint8_t FontStretch
```

Relative width of the font.

7.5.1.31 FontWeight

```
typedef uint8_t FontWeight
```

Weight of the font (apparent thickness of glyphs).

7.5.1.32 Key

```
typedef uint8_t Key
```

Storage type for portable key constants.

7.5.1.33 KeyEvent

```
typedef uint8_t KeyEvent
```

Type of keyboard event corresponding to the notification.

7.5.1.34 LineCaps

```
typedef uint8_t LineCaps
```

Type of caps that covers line end points.

7.5.1.35 Margins

```
typedef struct Margins Margins
```

[Margins](#) defined by top, left, right and bottom edge paddings.

7.5.1.36 Matrix

```
typedef struct Matrix Matrix
```

4x4 matrix representing transformation information in context of 3D graphics.

7.5.1.37 Matrix3x2

```
typedef struct Matrix3x2 Matrix3x2
```

3x2 matrix representing rotation and translation in context of 2D graphics.

7.5.1.38 MeshAlign

```
typedef uint8_t MeshAlign
```

Axis alignment that determines the placement of mesh around its model.

7.5.1.39 ModelTransform

```
typedef uint8_t ModelTransform
```

Type of transform as used by object models.

7.5.1.40 MouseButton

```
typedef uint8_t MouseButton
```

Type of mouse button that corresponds to the notification.

7.5.1.41 MouseEvent

```
typedef uint8_t MouseEvent
```

Type of mouse event corresponding to the notification.

7.5.1.42 ObjectModelViewCompare

```
typedef uint8_t ObjectModelViewCompare
```

Type of comparison applied to an ordered list of objects.

7.5.1.43 PathJoint

```
typedef uint8_t PathJoint
```

Type of joint for connecting path lines.

7.5.1.44 PixelFormat

```
typedef uint8_t PixelFormat
```

Defines how individual pixels and their colors are encoded in images and textures.

7.5.1.45 Point

```
typedef struct Point Point
```

2D integer point (or vector).

7.5.1.46 PointF

```
typedef struct PointF PointF
```

2D floating-point vector.

7.5.1.47 PointShape

```
typedef uint8_t PointShape
```

Shape of point primitive.

7.5.1.48 PrimitiveTopology

```
typedef uint8_t PrimitiveTopology
```

Rendering primitive topology.

7.5.1.49 Quad

```
typedef struct Quad Quad
```

Floating-point quadrilateral defined by four vertices in clockwise order starting from top/left.

7.5.1.50 Quaternion

```
typedef struct Quaternion Quaternion
```

3D quaternion.

7.5.1.51 Rect

```
typedef struct Rect Rect
```

Rectangle defined by integer top and left margins, width and height.

7.5.1.52 RectF

```
typedef struct RectF RectF
```

Rectangle defined by floating-point top and left margins, width and height.

7.5.1.53 SceneSamplerType

```
typedef uint8_t SceneSamplerType
```

Type of sampler used by the scene.

7.5.1.54 SceneTextureType

```
typedef uint8_t SceneTextureType
```

Type of texture used by the scene.

7.5.1.55 SelectionHighlightTextureType

```
typedef uint8_t SelectionHighlightTextureType
```

Type of texture provided by selection highlight module.

7.5.1.56 ShaderElement

```
typedef uint8_t ShaderElement
```

Type that characterizes shader element.

7.5.1.57 ShaderType

```
typedef uint8_t ShaderType
```

Type of shader in graphics rendering pipeline.

7.5.1.58 StencilOp

```
typedef uint8_t StencilOp
```

Operation that should be performed on stencil value.

7.5.1.59 SuperSampleSDF

```
typedef uint8_t SuperSampleSDF
```

Type of super-sampling used for rendering Signed Distance Field (SDF).

7.5.1.60 TechniqueLighting

```
typedef uint8_t TechniqueLighting
```

Lighting technique that defines what type of lighting formula and/or BDRF is being used.

7.5.1.61 TechniqueShadows

```
typedef uint8_t TechniqueShadows
```

Shadow rendering technique.

7.5.1.62 TextAlignment

```
typedef uint8_t TextAlignment
```

Text alignment when drawing with certain functions.

7.5.1.63 TextureAddress

```
typedef uint8_t TextureAddress
```

Addressing mode that defines how texture coordinates outside of [0, 1] range are treated.

7.5.1.64 TextureCabinetFilterType

```
typedef uint8_t TextureCabinetFilterType
```

Filter type that defines how textures are processed.

7.5.1.65 TextureCabinetPass

```
typedef uint8_t TextureCabinetPass
```

Defines type of rendering pass that is currently being performed.

7.5.1.66 TextureCabinetType

```
typedef uint8_t TextureCabinetType
```

Texture type used in the deferred shading scene.

7.5.1.67 TextureFidelity

```
typedef uint8_t TextureFidelity
```

Determines the choice of pixel formats for the container.

7.5.1.68 TextureFilter

```
typedef uint8_t TextureFilter
```

Filtering mode that defines how colors are sampled from the texture.

7.5.1.69 TextureType

```
typedef uint8_t TextureType
```

Type of texture that defines how it is composed.

7.5.1.70 TriangleFace

```
typedef uint8_t TriangleFace
```

Type of triangle face that should have processing applied to.

7.5.1.71 Vector

```
typedef struct Vector Vector
```

3D floating-point vector.

7.5.1.72 Vector4

```
typedef struct Vector4 Vector4
```

4-component (XYZ + W) floating-point vector.

7.5.1.73 WidgetAlignment

```
typedef uint8_t WidgetAlignment
```

Alignment of the widget respective its parent.

7.5.1.74 WidgetManagerTextureType

```
typedef uint8_t WidgetManagerTextureType
```

Texture type used by widget manager for the compositioning of the rendered UI.

7.5.1.75 WidgetPropertyBehavior

```
typedef uint8_t WidgetPropertyBehavior
```

Behavior of the widget property, which can indicate its lifetime or whether the property is an event.

7.5.1.76 WidgetPropertyType

```
typedef uint8_t WidgetPropertyType
```

Data type of a widget property.

7.5.2 Enumeration Type Documentation

7.5.2.1 anonymous enum

```
anonymous enum
```

Alpha-channel format request enumeration that can be passed as one of [AlphaFormatRequest](#) values.

Enumerator

ALPHA_FORMAT_REQUEST_DONT_CARE	Alpha-channel can be handled either way.
ALPHA_FORMAT_REQUEST_NON_PREMULTIPLIED	Alpha-channel in the image should not be premultiplied. Under normal circumstances, this is the recommended approach as it preserves RGB color information in its original form. However, when using mipmapping for images that have alpha-channel, Premultiplied gives more accurate results.
ALPHA_FORMAT_REQUEST_PREMULTIPLIED	Alpha-channel in the image should be premultiplied. Under normal circumstances, this is not recommended as the image would lose information after RGB components are premultiplied by alpha (and for smaller alpha values, less information is preserved). However, when using mipmapping for images that have alpha-channel, this gives more accurate results.

7.5.2.2 anonymous enum

anonymous enum

Enumerator

LEGACY_BIT_MISSING_DEPTH_TEXTURE	Direct3D 9 without depth texture support.
LEGACY_BIT_DEPTH_FORMAT_RAWZ	Direct3D 9 with Nvidia Geforce 6/7 series use RAWZ depth buffer format.
LEGACY_BIT_DIFFERENT_BITDEPTH_MRT	Direct3D 9 with capability bits set to support different bit-depths in MRT stack.

7.5.2.3 anonymous enum

anonymous enum

Type of surface layer that should be cleared.

Enumerator

DEVICE_CLEAR_LAYER_COLOR	Color buffer.
DEVICE_CLEAR_LAYER_DEPTH	Depth buffer.
DEVICE_CLEAR_LAYER_STENCIL	Stencil buffer.

7.5.2.4 anonymous enum

anonymous enum

Device attributes that define its behavior.

Enumerator

DEVICE_ATTRIBUTE_DEBUG	Debug-type context that helps to resolve issues in rendering code.
DEVICE_ATTRIBUTE_SOFTWARE	Software-based solution is used for rendering instead of GPU acceleration.
DEVICE_ATTRIBUTE_LEGACY	Compatibility device that is meant for older technologies.
DEVICE_ATTRIBUTE_LIMITED_EXTENSIONS	Device should only enable extensions determined by their appropriate level (deviceAttributeExtensions), even if a higher core version is available that inherently supports higher level extensions.

7.5.2.5 anonymous enum

anonymous enum

Device behavior attributes that define the rendering code path.

Enumerator

DEVICE_BEHAVIOR_PER_SAMPLE_SHADING	Graphics hardware supports executing fragment shader per each MSAA sample.
DEVICE_BEHAVIOR_DEPTH_CLIP_NEGATIVE	3D clip space uses negative values for depth in range of [-1, 1] instead of [0, 1]. This typically occurs on OpenGL (ES) backend that do not support clip control functions.
DEVICE_BEHAVIOR_COMPUTE	Compute shader support is available.
DEVICE_BEHAVIOR_TESSELLATION	Tessellation shader support is available.
DEVICE_BEHAVIOR_VARIABLE_RATE_REFRESH	Direct3D 11: Variable Rate Refresh (VRR) display support is available.
DEVICE_BEHAVIOR_POST_DEPTH_COVERAGE	OpenGL: ARB_post_depth_coverage, Direct3D 11: PostZCoverageEnable through NvAPI.
DEVICE_BEHAVIOR_FORCE_BUFFER_UNBIND	OpenGL / Intel HD Graphics bug workaround: always unbind buffers before updating their contents.
DEVICE_BEHAVIOR_DEPTH_CLEAR_BAD_PRECISION ↔	Direct3D 11 / Parallels bug workaround: clear depth buffer with value lower than one.
DEVICE_BEHAVIOR_SIGNED_NORM_INT_FORMAT_BUGGED ↔	Direct3D 11 / VMware bug workaround: avoid signed normalized integer formats.

7.5.2.6 anonymous enum

anonymous enum

Device memory barrier cumulative bits.

Enumerator

DEVICE_BARRIER_SHADER_BUFFER	Shader data sourced from buffers after the barrier will reflect data written by shaders prior to the barrier.
------------------------------	---

Enumerator

DEVICE_BARRIER_TEXTURE_FETCH	Texture fetches from shaders, including fetches from typed buffers, after the barrier will reflect data written by shaders prior to the barrier.
DEVICE_BARRIER_SHADER_IMAGE_ACCESS	Memory accesses using shader image load, store, and atomic built-in functions issued after the barrier will reflect data written by shaders prior to the barrier. Additionally, image stores and atomics issued after the barrier will not execute until all memory accesses (e.g., loads, stores, texture fetches, vertex fetches) initiated prior to the barrier complete.
DEVICE_BARRIER_TEXTURE_UPDATE	Writes to a texture by calling Update and Copy functions, and reads via Retrieve after the barrier will reflect data written by shaders prior to the barrier. Additionally, texture writes from these commands issued after the barrier will not execute until all shader writes initiated prior to the barrier complete.
DEVICE_BARRIER_BUFFER_UPDATE	Buffer reads or writes by calling Update and Copy functions after the barrier will reflect data written by shaders prior to the barrier. Additionally, writes via these commands issued after the barrier will wait on the completion of any shader writes to the same memory initiated prior to the barrier.
DEVICE_BARRIER_DRAWABLES	Reads and writes to drawable textures after the barrier will reflect data written by shaders prior to the barrier. Additionally, framebuffer writes issued after the barrier will wait on the completion of all shader writes issued prior to the barrier.
DEVICE_BARRIER_ATOMIC_COUNTER	Accesses to atomic counters after the barrier will reflect writes prior to the barrier.
DEVICE_BARRIER_STRUCTURED	Accesses to structured buffers after the barrier will reflect writes prior to the barrier.

7.5.2.7 anonymous enum

anonymous enum

Graphics device technology enumeration that can be passed as one of [DeviceTechnology](#) values.

Enumerator

DEVICE_TECHNOLOGY_UNKNOWN	The technology has not yet been established.
DEVICE_TECHNOLOGY_DIRECT3D	Microsoft Direct3D technology.
DEVICE_TECHNOLOGY_OPENGL	OpenGL by Khronos Group.
DEVICE_TECHNOLOGY_OPENGL_ES	OpenGL ES by Khronos Group.
DEVICE_TECHNOLOGY_VULKAN	Vulkan API by Khronos Group.
DEVICE_TECHNOLOGY_METAL	Metal API by Apple.
DEVICE_TECHNOLOGY_WEBGL	WebGL by Khronos Group.
DEVICE_TECHNOLOGY_SOFTWARE	Software rasterizer.
DEVICE_TECHNOLOGY_PROPRIETARY	Private proprietary technology.

7.5.2.8 anonymous enum

anonymous enum

Graphics device platform enumeration that can be passed as one of [DevicePlatform](#) values.

Enumerator

DEVICE_PLATFORM_UNKNOWN	The platform could not be properly identified.
DEVICE_PLATFORM_WINDOWS	Microsoft Windows platform.
DEVICE_PLATFORM_LINUX	Linux platform.
DEVICE_PLATFORM_UNIX	Generic Unix platform.
DEVICE_PLATFORM_OSX	Apple OS X platform.
DEVICE_PLATFORM_IOS	Apple iOS platform.
DEVICE_PLATFORM_ANDROID	Google Android platform.

7.5.2.9 anonymous enum

anonymous enum

State flags that can be combined together to form a rendering operation state.

Enumerator

DEVICE_STATE_DEPTH_TEST	Depth testing should be performed. If this state is not present, depth writing is also disabled.
DEVICE_STATE_DEPTH_WRITE	Depth values should be written to Depth Buffer.
DEVICE_STATE_STENCIL_TEST	Stencil testing should be performed. If this state is not present, no other stencil operation will occur.
DEVICE_STATE_DEPTH_CLIP	Geometry should be clipped by the corresponding depth limits.
DEVICE_STATE_SCISSOR_CLIP	Geometry should be clipped by the corresponding scissor rectangles.
DEVICE_STATE_WIREFRAME	Primitives should be rendered in a wireframe mode.
DEVICE_STATE_CULL_CLOCKWISE	Triangles having vertices in clockwise order would be culled. If this flag is not set, then triangles having vertices in counter-clockwise order would be culled.
DEVICE_STATE_MULTISAMPLING	Geometry should be rendered with multisampling.
DEVICE_STATE_LINE_ANTIALIAS	Standard 3D lines should be rendered with antialiasing.
DEVICE_STATE_COLOR_WRITE	Writing to color buffer should be performed.
DEVICE_STATE_BLEND_ENABLE	Blending operation should be performed.
DEVICE_STATE_ALPHA_TO_COVERAGE	Alpha channel should be converted to coverage samples.
DEVICE_STATE_CUBE_MAP_SEAMLESS	Cube Maps should be sampled from neighbor faces at seams.
DEVICE_STATE_PER_SAMPLE_SHADING	Enables fragment shader execution for each individual MSAA sample. This has only effect on OpenGL (ES) based implementations.

7.5.2.10 anonymous enum

```
anonymous enum
```

Face processing type enumeration that can be passed as one of [TriangleFace](#) values.

Enumerator

TRIANGLE_FACE_NONE	Neither back nor front faces should be processed.
TRIANGLE_FACE_BACK	Back faces should be processed.
TRIANGLE_FACE_FRONT	Front faces should be processed.
TRIANGLE_FACE_BOTH	Both back and front faces should be processed.

7.5.2.11 anonymous enum

```
anonymous enum
```

Comparison function enumeration that can be passed as one of [ComparisonFunc](#) values.

Enumerator

COMPARISON_FUNC_ALWAYS	Comparison is always true.
COMPARISON_FUNC_LESS	Comparison results true when source is less than destination.
COMPARISON_FUNC_LESS_EQUAL	Comparison results true when source is less than or equal to destination.
COMPARISON_FUNC_GREATER	Comparison results true when source is greater than destination.
COMPARISON_FUNC_GREATER_EQUAL	Comparison results true when source is greater than or equal to destination.
COMPARISON_FUNC_EQUAL	Comparison results true when source equals to destination.
COMPARISON_FUNC_NOT_EQUAL	Comparison results true when source is not equal to destination.
COMPARISON_FUNC_NEVER	Comparison is always false.

7.5.2.12 anonymous enum

```
anonymous enum
```

Stencil operation enumeration that can be passed as one of [StencilOp](#) values.

Enumerator

STENCIL_OP_KEEP	Preserve destination value.
STENCIL_OP_ZERO	Set destination value to zero.
STENCIL_OP_REPLACE	Replace destination value with the source one.
STENCIL_OP_INVERT	Invert the destination value.
STENCIL_OP_INCREMENT	Increment the destination value by the source one and clamp, if necessary.
STENCIL_OP_DECREMENT	Decrement the destination value by the source one and clamp, if necessary.
STENCIL_OP_INCREMENT_WARP	Increment the destination value by the source one with wrap-around.
STENCIL_OP_DECREMENT_WARP	Decrement the destination value by the source one with wrap-around.

7.5.2.13 anonymous enum

anonymous enum

Blending factor enumeration that can be passed as one of [BlendFactor](#) values.

Enumerator

BLEND_FACTOR_ONE	Blending factor is a constant that equals to one.
BLEND_FACTOR_ZERO	Blending factor is a constant that equals to zero.
BLEND_FACTOR_SOURCE_COLOR	Source color is used as a blending factor.
BLEND_FACTOR_INV_SOURCE_COLOR	One minus source color is used as a blending factor.
BLEND_FACTOR_SOURCE_ALPHA	Source alpha value is used as a blending factor.
BLEND_FACTOR_INV_SOURCE_ALPHA	One minus source alpha value is used as a blending factor.
BLEND_FACTOR_DEST_COLOR	Destination color is used as a blending factor.
BLEND_FACTOR_INV_DEST_COLOR	One minus destination color is used as a blending factor.
BLEND_FACTOR_DEST_ALPHA	Destination alpha value is used as a blending factor.
BLEND_FACTOR_INV_DEST_ALPHA	One minus destination alpha value is used as a blending factor.
BLEND_FACTOR_SOURCE_ALPHA_SAT	Blending factor is calculated as $\min(\text{SrcAlpha}, \text{InvDestAlpha})$.
BLEND_FACTOR_CONSTANT_COLOR	Blending factor is a user-defined color constant.
BLEND_FACTOR_INV_CONSTANT_COLOR	Blending factor is one minus user-defined color constant.
BLEND_FACTOR_CONSTANT_ALPHA	Blending factor is an alpha value from user-defined color constant.
BLEND_FACTOR_INV_CONSTANT_ALPHA	Blending factor is one minus alpha value from user-defined color constant.
BLEND_FACTOR_SOURCE_COLOR_1	Source color (from 2nd pixel shader output) is used as a blending factor.
BLEND_FACTOR_INV_SOURCE_COLOR_1	One minus source color (from 2nd pixel shader output) is used as a blending factor.
BLEND_FACTOR_SOURCE_ALPHA_1	Source alpha value (from 2nd pixel shader output) is used as a blending factor.
BLEND_FACTOR_INV_SOURCE_ALPHA_1	One minus source alpha (from 2nd pixel shader output) value is used as a blending factor.

7.5.2.14 anonymous enum

anonymous enum

Blending operation enumeration that can be passed as one of [BlendOp](#) values.

Enumerator

BLEND_OP_ADD	Add source and destination values.
BLEND_OP_SUBTRACT	Subtract source value from destination value.
BLEND_OP_INV_SUBTRACT	Subtract destination value from source value.
BLEND_OP_MINIMUM	Calculate minimum between source and destination values.
BLEND_OP_MAXIMUM	Calculate maximum between source and destination values.

7.5.2.15 anonymous enum

```
anonymous enum
```

Buffer data type enumeration that can be passed as one of [BufferData Type](#) values.

Enumerator

BUFFER_DATA_TYPE_VERTEX	Vertex information is stored in the buffer (Vertex Buffer).
BUFFER_DATA_TYPE_INDEX	Index information is stored in the buffer (Index Buffer).
BUFFER_DATA_TYPE_CONSTANT	Shader constant information is stored in the buffer (Constant Buffer).
BUFFER_DATA_TYPE_TYPED	Read-only shader buffer contained typed information (TBO / Buffer).
BUFFER_DATA_TYPE_RW_TYPED	Read-only shader buffer contained typed information (TBO / RWBuffer).
BUFFER_DATA_TYPE_STRUCTURED	Read-only shader buffer containing structured information (SSBO / StructuredBuffer).
BUFFER_DATA_TYPE_RW_STRUCTURED	Read/write shader buffer containing structured information (SSBO / RWStructuredBuffer).

7.5.2.16 anonymous enum

```
anonymous enum
```

Buffer access type enumeration that can be passed as one of [BufferAccess Type](#) values.

Enumerator

BUFFER_ACCESS_TYPE_DEFAULT	The contents of buffer are updated occasionally.
BUFFER_ACCESS_TYPE_STATIC	The contents of buffer are defined during construction and typically remain unchanged.
BUFFER_ACCESS_TYPE_DYNAMIC	The contents of buffer are changed very frequently.
BUFFER_ACCESS_TYPE_COMPUTE	The contents of the buffer are used by GPU exclusively.
BUFFER_ACCESS_TYPE_STAGING	The buffer is used for asynchronous transfers from GPU back to CPU.

7.5.2.17 anonymous enum

```
anonymous enum
```

Primitive topology enumeration that can be passed as one of [PrimitiveTopology](#) values.

Enumerator

PRIMITIVE_TOPOLOGY_UNKNOWN	Unknown (undefined) topology.
PRIMITIVE_TOPOLOGY_POINTS	Points or point lists.
PRIMITIVE_TOPOLOGY_LINES	Lines or line lists.
PRIMITIVE_TOPOLOGY_LINE_STRIP	Line strips.

Enumerator

PRIMITIVE_TOPOLOGY_TRIANGLES	Triangles or triangle lists.
PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP	Triangle strips.
PRIMITIVE_TOPOLOGY_LINES_ADJACENCY	Lines or line lists with adjacency information.
PRIMITIVE_TOPOLOGY_LINE_STRIP_↔ ADJACENCY	Line strips with adjacency information.
PRIMITIVE_TOPOLOGY_TRIANGLES_↔ ADJACENCY	Triangles or triangle lists with adjacency information.
PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_↔ ADJACENCY	Triangle strips with adjacency information.

7.5.2.18 anonymous enum

anonymous enum

Element format enumeration that can be passed as one of [ElementFormat](#) values.

Enumerator

ELEMENT_FORMAT_UNDEFINED	Undefined element format (VertexElement::count will define number of actual bytes).
ELEMENT_FORMAT_FLOAT	32-bit floating-point format.
ELEMENT_FORMAT_HALF_FLOAT	16-bit floating-point format.
ELEMENT_FORMAT_DOUBLE	64-bit floating-point format.
ELEMENT_FORMAT_INT	32-bit signed integer format.
ELEMENT_FORMAT_UNSIGNED_INT	32-bit unsigned integer format.
ELEMENT_FORMAT_SHORT	16-bit signed integer format.
ELEMENT_FORMAT_UNSIGNED_SHORT	16-bit unsigned integer format.
ELEMENT_FORMAT_BYTE	8-bit signed integer format.
ELEMENT_FORMAT_UNSIGNED_BYTE	8-bit unsigned integer format.
ELEMENT_FORMAT_FLOAT_11_11_10	32-bit storage for two 11-bit and one 10-bit floating-point numbers. The number of elements for this format must always be set to three (3).

7.5.2.19 anonymous enum

anonymous enum

Shader type enumeration that can be passed as one of [ShaderType](#) values.

Enumerator

SHADER_TYPE_UNDEFINED	Undefined or unknown shader.
SHADER_TYPE_FRAGMENT	Fragment (pixel) shader.
SHADER_TYPE_VERTEX	Vertex shader.
SHADER_TYPE_GEOMETRY	Geometry shader.
SHADER_TYPE_COMPUTE	Compute shader.
SHADER_TYPE_HULL	Hull shader.
SHADER_TYPE_DOMAIN	Domain shader.

7.5.2.20 anonymous enum

anonymous enum

Shader element enumeration that can be passed as one of [ShaderElement](#) values.

Enumerator

SHADER_ELEMENT_UNDEFINED	Undefined element type.
SHADER_ELEMENT_TEXTURE	Element is a texture.
SHADER_ELEMENT_CONSTANT_BUFFER	Element is a constant buffer (uniform buffer object).
SHADER_ELEMENT_TYPED_BUFFER	Element is a typed buffer (TBO / Buffer). Shares channels with textures.
SHADER_ELEMENT_RW_TYPED_BUFFER	Element is a typed buffer (TBO / RWBuffer). Shares channels with textures. For writing under OpenGL, this is bound as ImageBuffer (not TBO). For writing under Direct3D, this is bound as an Unordered Access View (UAV).
SHADER_ELEMENT_STRUCTURED_BUFFER	Element is a compute shader buffer (SSBO / StructuredBuffer).
SHADER_ELEMENT_RW_STRUCTURED_BUFFER	Element is a compute shader buffer (SSBO / RWStructuredBuffer).

7.5.2.21 anonymous enum

anonymous enum

Enumerator

VERTEX_CHANNEL_INDEX_BUFFER	Special channel number that corresponds to index buffer.
VERTEX_CHANNEL_NO_BUFFERS	Special channel number that corresponds to an empty line.
SHADER_INDEX_UNDEFINED	Special shader's element value (channel or index) that corresponds to an undefined (unset) location.
VERTEX_CHANNEL_NORMALIZED	Special bit added to vertex element channel indicating that the integer values are normalized.

7.5.2.22 anonymous enum

anonymous enum

Compute program texture access type that can be passed as one of [ComputeTextureAccess](#) values.

Enumerator

COMPUTE_TEXTURE_ACCESS_READ	Texture will be read from, no writes allowed.
COMPUTE_TEXTURE_ACCESS_WRITE	Texture will only be written to, no reads allowed.
COMPUTE_TEXTURE_ACCESS_READ_WRITE	Texture will be read from and written to.

7.5.2.23 anonymous enum

anonymous enum

Texture filtering enumeration that can be passed as one of [TextureFilter](#) values.

Enumerator

TEXTURE_FILTER_NONE	No filtering is performed (applies only to mipmap filter, in which case it gets disabled).
TEXTURE_FILTER_NEAREST	Nearest integer sample filtering.
TEXTURE_FILTER_LINEAR	Linear interpolation filtering.
TEXTURE_FILTER_ANISOTROPIC	Linear anisotropic filtering.

7.5.2.24 anonymous enum

anonymous enum

Texture addressing enumeration that can be passed as one of [TextureAddress](#) values.

Enumerator

TEXTURE_ADDRESS_WRAP	Texture coordinates are wrapped; that is, only fractional part is considered.
TEXTURE_ADDRESS_CLAMP	Texture coordinates are clamped to stay within [0, 1] range.
TEXTURE_ADDRESS_MIRROR	Texture coordinates are mirrored in each odd cycle (e.g. from 1 to 2, from 3 to 4 and so on).
TEXTURE_ADDRESS_MIRROR_ONCE	Texture coordinates are mirrored for one cycle and then clamped.
TEXTURE_ADDRESS_BORDER	When texture coordinates are outside of [0, 1] range, a separate border color will be used instead.

7.5.2.25 anonymous enum

anonymous enum

Attribute that defines special behavior and/or unique characteristics of the texture.

Enumerator

TEXTURE_ATTRIBUTE_MIPMAPPING	Texture has a full mipmap surface chain attached. MipMapping technique can improve visual quality when the texture is drawn in different sizes, especially in smaller ones.
TEXTURE_ATTRIBUTE_DRAWABLE	Texture contains a special type of surface that can be rendered to. In other words, the texture can be considered a render target (Direct3D) or render buffer object (OpenGL).
TEXTURE_ATTRIBUTE_DYNAMIC	Texture is meant for frequent write-only access. Dynamic textures cannot contain mipmap surface chain attached and cannot be drawable either.

Enumerator

TEXTURE_ATTRIBUTE_PREMULTIPLIED_ALPHA	Texture has its color components premultiplied by alpha-channel. This implies permanent loss of information as the components are multiplied by alpha value and stored (so, for example, pixels with alpha value of zero permanently lose all color information), however this can improve visual quality on mipmaps with translucent pixels. The parameter is merely a hint for rendering system, it does not change the actual pixels - an action that should be performed separately.
TEXTURE_COMPUTE	Texture will be used for Compute shader access (Direct3D-only, makes texture UAV-capable).
TEXTURE_SCRATCH	Texture is of "scratch" type. That is, it uses system memory for storage and, when used in context of Direct3D and/or OpenGL, it cannot be used directly.

7.5.2.26 anonymous enum

anonymous enum

Texture type enumeration that can be passed as one of [TextureType](#) values.

Enumerator

TEXTURE_TYPE_SURFACE	Standard 2D texture surface.
TEXTURE_TYPE_CUBE_MAP	Cube Map consisting of 6 individual 2D faces.
TEXTURE_TYPE_VOLUME	Layered 3D volume texture.

7.5.2.27 anonymous enum

anonymous enum

Blending effect enumeration that can be passed as one of [BlendingEffect](#) values.

Enumerator

BLENDING_EFFECT_UNDEFINED	Undefined blending effect. Canvas does not change blending parameters configured in the device context and uses whatever configuration is currently active.
BLENDING_EFFECT_NORMAL	"Normal" blending effect: $Cd = (Cs * As) + (Cd * (1 - As))$.
BLENDING_EFFECT_SHADOW	"Shadow drawing" effect: $Cd = Cd * (1 - As)$.
BLENDING_EFFECT_ADD	Additive blending effect: $Cd = (Cs * As) + Cd$.
BLENDING_EFFECT_SUBTRACT	Subtractive blending effect: $Cd = (Cs * As) - Cd$.
BLENDING_EFFECT_MULTIPLY	Multiplication blending effect: $Cd = Cs * Cd$.
BLENDING_EFFECT_INVERSE_MULTIPLY	Inverse multiplication effect: $Cd = (1 - Cs) * Cd$.
BLENDING_EFFECT_SOURCE_COLOR	Source color blending effect: $Cd = (Cs * Cs) + (Cd * (1 - Cs))$.
BLENDING_EFFECT_SOURCE_COLOR_ADD	Source color additive blending effect: $Cd = (Cs * Cs) + Cd$.
BLENDING_EFFECT_COPY	Alpha-blending disabled, drawing is just a copy operation.
BLENDING_EFFECT_CUSTOM	Custom alpha-blending as set up through device directly.

7.5.2.28 anonymous enum

anonymous enum

Canvas context state enumeration that can be passed as one of [CanvasContextState](#) values.

Enumerator

CANVAS_CONTEXT_STATE_FLAT_SCENE	Geometric shapes and images rendered in a flat 2D scene.
CANVAS_CONTEXT_STATE_FLAT_TEXT	Text rendered in a flat 2D scene.
CANVAS_CONTEXT_STATE_PROJECTED	Geometric shapes, images and text rendered in a 3D surface projection.
CANVAS_CONTEXT_STATE_UNDEFINED	Undefined context state.

7.5.2.29 anonymous enum

anonymous enum

Canvas attribute flags that can be combined together.

Enumerator

CANVAS_ATTRIBUTE_PROJECTED	Transform 2D coordinates as 3D vectors (with Z = 0) by the appropriate 4x4 matrix as if drawing to an arbitrary plane in 3D.
CANVAS_ATTRIBUTE_SDF	Preprocess the texture by using its alpha-channel as a signed distance field.
CANVAS_ATTRIBUTE_OUTLINE_SDF	Include outline when rendering signed distance field (implies <code>attributeSDF</code>).
CANVAS_ATTRIBUTE_CUBIC	Sample texture using bicubic interpolation for higher-quality results.
CANVAS_ATTRIBUTE_COLOR_ADJUST	Adjusts texture color using YCH instead of RGB (only works for textures).
CANVAS_ATTRIBUTE_TRANSFORM	Applies 3x3 world transformation matrix to non-holographic primitives.

7.5.2.30 anonymous enum

anonymous enum

Super-sample SDF type enumeration that can be passed as one of [SuperSampleSDF](#) values.

Enumerator

SUPER_SAMPLE_SDF_NO_SUPER_SAMPLE	Non-supersampled rendering (for low-performance systems). Note that if outline is enabled, its color will be calculated using simplified scheme, where luma is calculated as usual, chroma is calculated in a simple fashion and hue is not affected.
SUPER_SAMPLE_SDF_SUPER_SAMPLE_4X	4x super-sampled rendering.
SUPER_SAMPLE_SDF_SUPER_SAMPLE_16X	16x super-sampled rendering.

7.5.2.31 anonymous enum

anonymous enum

Image Region modifier flags that can be combined together.

Enumerator

IMAGE_REGION_FLIP	Flip region vertically.
IMAGE_REGION_MIRROR	Mirror region horizontally.
IMAGE_REGION_ROTATE	Rotate region by 90 degrees clockwise.

7.5.2.32 anonymous enum

anonymous enum

Path joint enumeration that can be passed as one of [PathJoint](#) values.

Enumerator

PATH_JOINT_NONE	No joint is produced, lines on each joint end with a butt cap.
PATH_JOINT_SIMPLE	Simple interpolated joint that does not guarantee correct line width on sharp corners.
PATH_JOINT_MITER	Miter joint with angular corners.
PATH_JOINT_BEVEL	Beveled joint.
PATH_JOINT_ROUND	Round joint.
PATH_JOINT_MITER_BEVEL	Miter joint unless the miter would extend beyond its limit multiplied by half thickness.

7.5.2.33 anonymous enum

anonymous enum

Line caps enumeration that can be passed as one of [LineCaps](#) values.

Enumerator

LINE_CAPS_BUTT	Cap that starts and ends exactly at the center of end points.
LINE_CAPS_SQUARE	Cap that extends by half thickness after the end points.
LINE_CAPS_ROUND	Round cap with a radius equal to half thickness with center being each line end point.

7.5.2.34 anonymous enum

anonymous enum

[Point](#) shape enumeration that can be passed as one of [PointShape](#) values.

Enumerator

POINT_SHAPE_SQUARE	Square point.
POINT_SHAPE_ROUND	Round point.
POINT_SHAPE_TRIANGLE	Equilateral triangle point.
POINT_SHAPE_STAR	5-sided star point.
POINT_SHAPE_CROSS	Cross point.

7.5.2.35 anonymous enum

anonymous enum

Font weight enumeration that can be passed as one of [FontWeight](#) values.

Enumerator

FONT_WEIGHT_THIN	Letters should appear as thin as technically possible.
FONT_WEIGHT_EXTRA_LIGHT	Letters should appear very thin.
FONT_WEIGHT_LIGHT	Letters should appear even thinner.
FONT_WEIGHT_SEMI_LIGHT	Letters should have slightly lighter thickness.
FONT_WEIGHT_NORMAL	Letters should appear normal (default thickness).
FONT_WEIGHT_MEDIUM	Letters should have medium thickness, which is slightly above than normal.
FONT_WEIGHT_SEMI_BOLD	Letters should appear semi-bold.
FONT_WEIGHT_BOLD	Letters should appear as bold.
FONT_WEIGHT_EXTRA_BOLD	Letters should appear as extra thick.
FONT_WEIGHT_HEAVY	Letters should appear very thick.
FONT_WEIGHT_EXTRA_HEAVY	Letters should appear as thick as technically possible.

7.5.2.36 anonymous enum

anonymous enum

Font stretch enumeration that can be passed as one of [FontStretch](#) values.

Enumerator

FONT_STRETCH_ULTRA_CONDENSED	Ultra-condensed width.
FONT_STRETCH_EXTRA_CONDENSED	Extra-condensed width.
FONT_STRETCH_CONDENSED	Condensed width.
FONT_STRETCH_SEMI_CONDENSED	Semi-condensed width.
FONT_STRETCH_NORMAL	Normal width.
FONT_STRETCH_SEMI_EXPANDED	Semi-expanded width.
FONT_STRETCH_EXPANDED	Expanded width.
FONT_STRETCH_EXTRA_EXPANDED	Extra-expanded width.
FONT_STRETCH_ULTRA_EXPANDED	Ultra-expanded width.

7.5.2.37 anonymous enum

`anonymous enum`

Font slant styles enumeration that can be passed as one of [FontSlant](#) values.

Enumerator

FONT_SLANT_NONE	Upright font.
FONT_SLANT_OBLIQUE	Slanted font in a roman style.
FONT_SLANT_ITALIC	Slanted in an italic style.

7.5.2.38 anonymous enum

`anonymous enum`

Font attribute flags that define some visual characteristics.

Enumerator

FONT_ATTRIBUTE_UNDERLINE	Font with underlined text.
FONT_ATTRIBUTE_STRIKE_OUT	Font with strike-out text.

7.5.2.39 anonymous enum

`anonymous enum`

Font border enumeration that can be passed as one of [FontBorder](#) values.

Enumerator

FONT_BORDER_NONE	No border will appear around letters.
FONT_BORDER_NORMAL	A light border will appear around letters.
FONT_BORDER_SEMI_HEAVY	A semi-heavy border will appear around letters.
FONT_BORDER_HEAVY	A heavy border will appear around letters.

7.5.2.40 anonymous enum

`anonymous enum`

Text alignment enumeration that can be passed as one of [TextAlignment](#) values.

Enumerator

TEXT_ALIGNMENT_START	Text should be aligned to the beginning (either top or left depending on context).
TEXT_ALIGNMENT_MIDDLE	Text should be centered in the middle.
TEXT_ALIGNMENT_END	Text should be aligned to the end (either bottom or right depending on context).

7.5.2.41 anonymous enum

anonymous enum

Format enumeration that can be passed as one of [ColorDitheringFormat](#) values.

Enumerator

COLOR_DITHERING_FORMAT_RGB8	Red, green and blue channels have 8 bits each (16M colors).
COLOR_DITHERING_FORMAT_RGB6	Red, green and blue channels have 6 bits each (256K colors).
COLOR_DITHERING_FORMAT_R5G6B5	Red and blue channels have 5 bits each, while green channel has 6 bits (64K colors).
COLOR_DITHERING_FORMAT_RGB4	Red, green and blue channels have 4 bits each (4K colors).
COLOR_DITHERING_FORMAT_RG3B2	Red and green channels have 3 bits each, while blue channel has 2 bits (256 colors).

7.5.2.42 anonymous enum

anonymous enum

Texture type enumeration that can be passed as one of [SelectionHighlightTextureType](#) values.

Enumerator

SELECTION_HIGHLIGHT_TEXTURE_HIGHLIGHT	Texture that contains the selection highlight, to be rendered on top of the scene.
SELECTION_HIGHLIGHT_TEXTURE_↔ HIGHLIGHT_MASK	Texture that contains the highlight mask, that can be optionally rendered on top of the scene.

7.5.2.43 anonymous enum

anonymous enum

Fog formula enumeration that can be passed as one of [FogFormula](#) values.

Enumerator

FOG_FORMULA_LINEAR	Fog calculated as $(far - d) / (far - near)$.
FOG_FORMULA_EXPONENTIAL	Fog calculated as $c \times e^{-(d \times b)^2}$.
FOG_FORMULA_GROUND	Ground fog with linear density and exponential depth.

7.5.2.44 anonymous enum

anonymous enum

Depth fog distance enumeration that can be passed as one of [DepthFogDistance](#) values.

Enumerator

DEPTH_FOG_DISTANCE_Z_BASED	Distance is based on depth.
DEPTH_FOG_DISTANCE_EYE_RELATIVE	Distance is relative to camera position.

7.5.2.45 anonymous enum

anonymous enum

Lighting technique enumeration that can be passed as one of [TechniqueLighting](#) values.

Enumerator

TECHNIQUE_LIGHTING_PHONG	Traditional Phong lighting.
TECHNIQUE_LIGHTING_MINNAERT	Minnaert lighting.
TECHNIQUE_LIGHTING_COOK_TORRANCE	Cook-Torrance lighting.
TECHNIQUE_LIGHTING_OREN_NAYER	Oren-Nayer lighting.

7.5.2.46 anonymous enum

anonymous enum

Shadow technique enumeration that can be passed as one of [TechniqueShadows](#) values.

Enumerator

TECHNIQUE_SHADOWS_NONE	No shadows are rendered or used.
TECHNIQUE_SHADOWS_ESM	Exponential Shadow Maps (ESM) using a single depth component.
TECHNIQUE_SHADOWS_ESM_WARP	Exponential Shadow Maps (ESM) using two depth components for positive and negative parts.
TECHNIQUE_SHADOWS_EVSM	Exponential Variance Shadow Maps (EVSM) using four depth components.

7.5.2.47 anonymous enum

anonymous enum

Light culling mode enumeration that can be passed as one of [ClustersCullingMode](#) values.

Enumerator

CLUSTERS_CULLING_MODE_QUALITY	High-quality mode supporting up to 1024 simultaneous lights per cluster.
CLUSTERS_CULLING_MODE_PERFORMANCE	Performance mode supporting up to 4 simultaneous lights per cluster.

7.5.2.48 anonymous enum

anonymous enum

Texture fidelity enumeration that can be passed as one of [TextureFidelity](#) values.

Enumerator

TEXTURE_FIDELITY_LOW	Use lowest quality texture formats, which may reduce GPU memory consumption and potentially improve performance slightly at expense of visual quality.
TEXTURE_FIDELITY_MEDIUM	Use standard quality texture formats suitable for most common usage scenarios.
TEXTURE_FIDELITY_HIGH	High quality texture formats, producing accurate and exceptionally looking images.
TEXTURE_FIDELITY_EXTREME	Excessively high quality texture formats, typically useful for generating images to be printed, screenshots and presentations. This may increase GPU memory consumption and impact performance.

7.5.2.49 anonymous enum

anonymous enum

Texture cabinet type enumeration that can be passed as one of [TextureCabinetType](#) values.

Enumerator

TEXTURE_CABINET_TYPE_DEPTH	Depth buffer of the scene.
TEXTURE_CABINET_TYPE_NORMALS	Texture containing information about screen-space normals.
TEXTURE_CABINET_TYPE_COLOR_HDR	High dynamic range color accumulation buffer.
TEXTURE_CABINET_TYPE_COLOR	Color buffer of the scene (non-MSAA).
TEXTURE_CABINET_TYPE_LINEAR_DEPTHS	Linear depths for spatial fog and water (MSAA).

7.5.2.50 anonymous enum

anonymous enum

Texture cabinet pass enumeration that can be passed as one of [TextureCabinetPass](#) values.

Enumerator

TEXTURE_CABINET_PASS_COLOR	Main color scene pass.
TEXTURE_CABINET_PASS_GLASSY	Order-independent transparency (OIT) post-pass.
TEXTURE_CABINET_PASS_DEPTH	Depth (and normals) pre-pass.

7.5.2.51 anonymous enum

anonymous enum

Texture cabinet filter type enumeration that can be passed as one of [TextureCabinetFilterType](#) values.

Enumerator

TEXTURE_CABINET_FILTER_TYPE_OCCLUSION	Ambient occlusion computation.
TEXTURE_CABINET_FILTER_TYPE_BLOOM	Bloom (glare) effect processing.
TEXTURE_CABINET_FILTER_TYPE_GLASSY	Order-independent transparency (OIT) composition.
TEXTURE_CABINET_FILTER_TYPE_GLASSY ↔ FOG	Spatial fog effects for order-independent transparency (OIT) composition.

7.5.2.52 anonymous enum

anonymous enum

Cumulative rendering attributes that can be passed to [TextureCabinetAttributes](#).

Enumerator

TEXTURE_CABINET_ATTRIBUTE_AMBIENT ↔ OCCLUSION	Screen-Space Ambient Occlusion technique.
TEXTURE_CABINET_ATTRIBUTE_AMBIENT ↔ OCCLUSION_DOWNSCALE	Downscales SSAO texture to enable 4x faster performance on low and mid-end GPUs.
TEXTURE_CABINET_ATTRIBUTE_BLOOM	Bloom (glare) technique.
TEXTURE_CABINET_ATTRIBUTE_BLOOM ↔ DOWNSCALE	Bloom is rendered at half of resolution to accomodate for higher display scales.
TEXTURE_CABINET_ATTRIBUTE_BLOOM_CUBIC	Bloom is applied using cubic interpolation for higher quality look.
TEXTURE_CABINET_ATTRIBUTE_LINEAR ↔ DEPTHS	Linear depths are provided for techniques such as fog or water.
TEXTURE_CABINET_ATTRIBUTE_GLASSY	Accurate order-independent transparency (OIT) single-pass technique.
TEXTURE_CABINET_ATTRIBUTE_GLASSY_FAST	Approximate order-independent transparency (OIT) single-pass technique.
TEXTURE_CABINET_ATTRIBUTE_GLASSY_FOG	Add to order-independent transparency (OIT) technique support for spatial fog effects.
TEXTURE_CABINET_ATTRIBUTE_GLASSY ↔ FROSTED	Frosted Glass effect added to order-independent transparency (OIT) techniques.

7.5.2.53 anonymous enum

anonymous enum

Cumulative rendering attributes that can be passed to [SceneAttributes](#).

Enumerator

SCENE_ATTRIBUTE_INSTANCING	Instancing technique will be used during rendering.
SCENE_ATTRIBUTE_NORMALS	Vertex normals will be processed and taken into account.
SCENE_ATTRIBUTE_COLORING	Vertex colors will be processed and taken into account.
SCENE_ATTRIBUTE_TEXTUREING_CUBIC	Cubic interpolation will be used for texture mapping.
SCENE_ATTRIBUTE_SHADOWS_CUBIC	Cubic interpolation will be used for shadow mapping.
SCENE_ATTRIBUTE_GLASSY	Perform adjustments for Order-Independent Transparency (OIT) rendering.
SCENE_ATTRIBUTE_SINGLE_PASS	The whole scene is rendered in a single pass with ambient occlusion.
SCENE_ATTRIBUTE_DEPTH_PREPASS	Generate output for depth pre-pass, which at minimum includes normals. Note: it is possible to do depth pre-pass without normals by excluding this attribute.
SCENE_ATTRIBUTE_LINEAR_DEPTHS	Produce linear depths as part of depth pre-pass for spatial fog or water effects. Note: this requires SCENE_ATTRIBUTE_DEPTH_PREPASS flag to be enabled and thus always includes normals.
SCENE_ATTRIBUTE_MODELING	Scene is a modeling scene, which means it supports textures and materials.

7.5.2.54 anonymous enum

anonymous enum

Scene texture type enumeration that can be passed as one of [SceneTextureType](#) values.

Enumerator

SCENE_TEXTURE_TYPE_ALBEDO	Albedo texture.
SCENE_TEXTURE_TYPE_NORMAL_MAP	Normal Map texture.
SCENE_TEXTURE_TYPE_PARALLAX_MAP	Displacement Map texture for Parallax Mapping technique.

7.5.2.55 anonymous enum

anonymous enum

Scene sampler type enumeration that can be passed as one of [SceneSamplerType](#) values.

Enumerator

SCENE_SAMPLER_TYPE_ALBEDO	Albedo sampler.
SCENE_SAMPLER_TYPE_NORMAL_MAP	Normal Map sampler.
SCENE_SAMPLER_TYPE_PARALLAX_MAP	Displacement Map sampler for Parallax Mapping technique.
SCENE_SAMPLER_TYPE_SHADOW_MAP	Shadow Map sampler.

7.5.2.56 anonymous enum

anonymous enum

Type of the mesh meta-tag.

Enumerator

MESH_META_TAG_TYPE_INDETERMINATE	Tag type is not determined.
MESH_META_TAG_TYPE_GEOMETRY	Tag represents a geometry group.
MESH_META_TAG_TYPE_OBJECT	Tag represents an object group.

7.5.2.57 anonymous enum

anonymous enum

Type of texture used in scene mesh material.

Enumerator

SCENE_MESH_TEXTURE_DIFFUSE	Diffuse color texture.
SCENE_MESH_TEXTURE_NORMALS	Normal-mapping texture.
SCENE_MESH_TEXTURE_PARALLAX	Parallax-mapping texture.

7.5.2.58 anonymous enum

anonymous enum

Type of latch used in the 3D scene mesh.

Enumerator

SCENE_MESH_LATCH_TYPE_JOINT	Bone joint connection.
SCENE_MESH_LATCH_TYPE_WAYPOINT	Movement waypoint.

7.5.2.59 anonymous enum

anonymous enum

Enumeration values that can be passed to scene mesh creation functions.

Enumerator

SCENE_MESH_VERTEX_ELEMENTS_INDEX_↔ UNDEFINED	A special "undefined" value for an vertex elements index.
---	---

7.5.2.60 anonymous enum

anonymous enum

Cumulative options that can be passed to one of xxAutoDraw functions.

Enumerator

AUTO_DRAW_OPTION_POST_UNBIND	Unbind the rendering buffers after issuing draw call. This can help resolve racing conditions under buggy drivers (mostly OpenGL) at the expense of reduced performance.
AUTO_DRAW_OPTION_RESET_TEXTURES	Reset scene textures after issuing draw call. For performance reasons, the textures should be reset after rendering all objects.
AUTO_DRAW_OPTION_MATERIAL_IGNORE	Ignore materials integrated into the mesh if such are present, using object material and/or vertex colors instead. This has no impact on "OBJECT_SKIP_X_HAVING_MATERIALS" options functionality.
AUTO_DRAW_OPTION_MATERIAL_↔ TRANSPARENCY	Alpha-channel from the scene mesh materials should be assigned to alpha-channel of the scene material (which normally corresponds to bloom factor). This is typically used when rendering semi-transparent objects.
AUTO_DRAW_OPTION_MATERIAL_SKIP_↔ TRANSPARENT	Portions of the mesh that have semi-transparent materials should be skipped.
AUTO_DRAW_OPTION_MATERIAL_SKIP_OPAQUE	Portions of the mesh that have opaque materials should be skipped.
AUTO_DRAW_OPTION_OBJECT_SKIP_↔ TRANSPARENT	Skips objects that have transparent attribute set.
AUTO_DRAW_OPTION_OBJECT_SKIP_NON_↔ TRANSPARENT	Skips objects that do not have transparent attribute set.
AUTO_DRAW_OPTION_OBJECT_SKIP_HAVING_↔ _MATERIALS	Skips objects that have mesh materials present.
AUTO_DRAW_OPTION_OBJECT_SKIP_NOT_↔ HAVING_MATERIALS	Skips objects that do not have mesh materials present.
AUTO_DRAW_OPTION_OBJECT_HIGHLIGHTED	Draws objects that have a highlight color assigned.
AUTO_DRAW_OPTION_OBJECT_DISABLE_↔ INSTANCING	Do not use instancing to render multiple objects simultaneously.

7.5.2.61 anonymous enum

anonymous enum

Object model transform type enumeration that can be passed as one of [ModelTransform](#) values.

Enumerator

MODEL_TRANSFORM_LOCAL	Local object transform.
MODEL_TRANSFORM_LOCAL_MODEL	Local Model object transform (for rendering).
MODEL_TRANSFORM_LOCAL_VOLUME	Local Volume object transform (OOBB Unity Cube).

Enumerator

MODEL_TRANSFORM_GLOBAL	Global object transform.
MODEL_TRANSFORM_GLOBAL_MODEL	Global Model object transform (for rendering).
MODEL_TRANSFORM_GLOBAL_VOLUME	Global Volume object transform (OOBB Unity Cube).

7.5.2.62 anonymous enum

anonymous enum

Axis alignment enumeration that can be passed as one of [MeshAlign](#) values.

Enumerator

MESH_ALIGN_POSITIVE	Mesh is aligned from the edge of positive axis.
MESH_ALIGN_ORIGIN	Mesh is aligned in the middle of axis origin.
MESH_ALIGN_NEGATIVE	Mesh is aligned from the edge of negative axis.
MESH_ALIGN_UNALIGNED	Mesh is unaligned on the axis.

7.5.2.63 anonymous enum

anonymous enum

Object model comparison enumeration that can be passed as one of [ObjectModelViewCompare](#) values.

Enumerator

OBJECT_MODEL_COMPARE_DEPTH	Objects are compared by their depth.
OBJECT_MODEL_COMPARE_DEPTH_REVERSE	Objects are compared by their depth in reverse (for rendering transparency without OIT).
OBJECT_MODEL_COMPARE_ORDERED	Objects are compared first by their order index, then by depth.
OBJECT_MODEL_COMPARE_MESHES	Objects are compared first by their mesh and then by depth.
OBJECT_MODEL_COMPARE_OBJECTS	Objects are sorted by their instance type id, mesh and then material.

7.5.2.64 anonymous enum

anonymous enum

Cumulative flags for a 3D object model that define its status.

Enumerator

OBJECT_MODEL_ATTRIBUTE_VISIBLE	Object is visible so can be rendered and selected (if selected flag is set).
--------------------------------	--

Enumerator

OBJECT_MODEL_ATTRIBUTE_SELECTABLE	Object can be selected.
OBJECT_MODEL_ATTRIBUTE_TRANSPARENT	Object is semi-transparent (might be rendered through a different mechanism).
OBJECT_MODEL_ATTRIBUTE_HIERARCHYLESS	Object does not participate in bounding volume hierarchy (BVH) calculation. This is particularly useful for constantly moving objects or those objects that have unusually large dimensions overlapping with other objects.
OBJECT_MODEL_ATTRIBUTE_DISABLE_REALIGN	Object should not automatically realign itself.
OBJECT_MODEL_ATTRIBUTE_NON_VIEWABLE	Object does not appear among visible objects in the view. This is useful for abstract objects that have no associated mesh, so aren't rendered in any way, but can still be selected.

7.5.2.65 anonymous enum

anonymous enum

Non-exclusive options passed and/or returned from mesh loading and saving functions.

Enumerator

MESH_LOADING_OPTION_MATERIAL_VERTEX↔ _COLOR	Material diffuse colors should be mapped to vertex colors.
MESH_LOADING_OPTION_MATERIAL_VERTEX↔ _COLOR_PREFERRED	Assume "MESH_LOADING_OPTION_MATERIAL_↔ VERTEX_COLOR" option only when the model would reasonably look the same as if it would have been rendered with its materials. This requires that all model materials do not use texturing, have no emissive color, no lighting technique, no bloom factor specified or have these at their default values.
MESH_LOADING_OPTION_MATERIAL_↔ TEXTURE_MISSING	One or more material texture could not be loaded.
MESH_LOADING_OPTION_EXPORT_NORMALS	Include vertex normals in the exported file.
MESH_LOADING_OPTION_EXPORT_TANGENTS	Include vertex tangents in the exported file.
MESH_LOADING_OPTION_EXPORT_TEXTURE↔ _COORDINATES	Include vertex texture coordinates in the exported file.
MESH_LOADING_OPTION_EXPORT_COLORS	Include vertex colors in the exported file.
MESH_LOADING_OPTION_EXPORT_WEIGHTS	Include vertex weights in the exported file.
MESH_LOADING_OPTION_STRIP_GEOMETRY_↔ NAMES	Strip names for geometry and object nodes when saving in OBJ format.

7.5.2.66 anonymous enum

anonymous enum

Camera movement/rotation command enumeration that can be passed as one of [CameraCommand](#) values.

Enumerator

CAMERA_COMMAND_MOVEMENT	Start camera movement on XZ plane.
CAMERA_COMMAND_DRAGGING	Start camera dragging.
CAMERA_COMMAND_ROTATION	Start camera rotation.
CAMERA_COMMAND_UPDATE	Update camera movement/rotation.
CAMERA_COMMAND_STOP	Stop camera movement/rotation.

7.5.2.67 anonymous enum

anonymous enum

Application window state enumeration that can be passed as one of [ApplicationWindowState](#) values.

Enumerator

APPLICATION_WINDOW_STATE_WINDOWED	Windows appears normally with its title and border.
APPLICATION_WINDOW_STATE_MINIMIZED	Window appears minimized, iconic or hidden.
APPLICATION_WINDOW_STATE_MAXIMIZED	Window appears maximized to fill the screen's client area.
APPLICATION_WINDOW_STATE_FULLSCREEN	Window appears occupying the entire screen.

7.5.2.68 anonymous enum

anonymous enum

Mouse event type enumeration that can be passed as one of [MouseEvent](#) values.

Enumerator

MOUSE_EVENT_MOUSE_DOWN	Mouse button is pressed.
MOUSE_EVENT_MOUSE_MOVE	Mouse cursor is being moved.
MOUSE_EVENT_MOUSE_UP	Mouse button is de-pressed.
MOUSE_EVENT_WHEEL_UP	Mouse wheel has been rotated up.
MOUSE_EVENT_WHEEL_DOWN	Mouse wheel has been rotated down.
MOUSE_EVENT_MOUSE_ENTER	Mouse has entered component's area.
MOUSE_EVENT_MOUSE_LEAVE	Mouse has left component's area.

7.5.2.69 anonymous enum

anonymous enum

Mouse button type enumeration that can be passed as one of [MouseButton](#) values.

Enumerator

MOUSE_BUTTON_NONE	None or unknown button (typically specified during mouse move event).
-------------------	---

Enumerator

MOUSE_BUTTON_LEFT	Left mouse button.
MOUSE_BUTTON_RIGHT	Right mouse button.
MOUSE_BUTTON_MIDDLE	Middle mouse button.

7.5.2.70 anonymous enum

anonymous enum

Keyboard event type enumeration that can be passed as one of [KeyEvent](#) values.

Enumerator

KEY_EVENT_PRESSED	Key has been pressed.
KEY_EVENT_RELEASED	Key has been released.
KEY_EVENT_TYPING	Key is being typed (for Unicode input).

7.5.2.71 anonymous enum

anonymous enum

Application event type enumeration that can be passed as one of [ApplicationEvent](#) values.

Enumerator

APPLICATION_EVENT_ACTIVATE	Application has been activated.
APPLICATION_EVENT_DEACTIVATE	Application has been de-activated.
APPLICATION_EVENT_MINIMIZE	Application has been minimized.
APPLICATION_EVENT_RESTORE	Application has been restored.
APPLICATION_EVENT_APPEARANCE	DPI changed or user switched between dark/white styles.

7.5.2.72 anonymous enum

anonymous enum

Application cursor enumeration that can be passed as one of [AppCursor](#) values.

Enumerator

APP_CURSOR_BLANK	Blank (invisible) cursor.
APP_CURSOR_ARROW	Standard arrow cursor.
APP_CURSOR_TEXT_SELECT	Text selection cursor.
APP_CURSOR_TEXT_SELECT_VERTICAL	
APP_CURSOR_WAIT	Wait cursor.
APP_CURSOR_ARROW_WAIT	Arrow cursor with wait icon.

Enumerator

APP_CURSOR_CROSSHAIR	Crosshair cursor.
APP_CURSOR_CELL	Cell cursor.
APP_CURSOR_LINK_SELECT	Hyperlink selection cursor.
APP_CURSOR_NOT_ALLOWED	Not allowed cursor.
APP_CURSOR_HELP	Help cursor.
APP_CURSOR_MOVE	Move cursor.
APP_CURSOR_DRAG	Drag cursor.
APP_CURSOR_DRAGGING	Dragging cursor.
APP_CURSOR_RESIZE_VERT	Vertical resize cursor.
APP_CURSOR_RESIZE_HORIZ	Horizontal resize cursor.
APP_CURSOR_RESIZE1	Diagonal north-east south-west resize cursor.
APP_CURSOR_RESIZE2	Diagonal north-west south-east resize cursor.
APP_CURSOR_RESIZE_TOP	Top resize cursor.
APP_CURSOR_RESIZE_BOTTOM	Bottom resize cursor.
APP_CURSOR_RESIZE_LEFT	Left resize cursor.
APP_CURSOR_RESIZE_RIGHT	Right resize cursor.
APP_CURSOR_RESIZE_TOP_LEFT	Top/left resize cursor.
APP_CURSOR_RESIZE_TOP_RIGHT	Top/right resize cursor.
APP_CURSOR_RESIZE_BOTTOM_RIGHT	Bottom/right resize cursor.
APP_CURSOR_RESIZE_BOTTOM_LEFT	Bottom/left resize cursor.
APP_CURSOR_UNDEFINED	Undefined or unspecified cursor.

7.5.2.73 anonymous enum

anonymous enum

File chooser dialog type enumeration that can be passed as one of [FileChooserDialog](#) values.

Enumerator

FILE_CHOOSER_DIALOG_OPEN_FILE	Open file dialog.
FILE_CHOOSER_DIALOG_SAVE_FILE	Save file dialog.
FILE_CHOOSER_DIALOG_DIRECTORY	Select directory dialog.

7.5.2.74 anonymous enum

anonymous enum

Widget alignment enumeration that can be passed as one of [WidgetAlignment](#) values.

Enumerator

WIDGET_ALIGNMENT_NONE	No alignment is applied to widget.
WIDGET_ALIGNMENT_CLIENT	Widget occupies the entire client area.
WIDGET_ALIGNMENT_LEFT	Widget is placed on the left occupying the entire client height.

Enumerator

WIDGET_ALIGNMENT_TOP	Widget is placed on the top occupying the entire client width.
WIDGET_ALIGNMENT_RIGHT	Widget is placed on the right occupying the entire client height.
WIDGET_ALIGNMENT_BOTTOM	Widget is placed on the bottom occupying the entire client width.

7.5.2.75 anonymous enum

anonymous enum

Widget property type enumeration that can be passed as one of [WidgetPropertyType](#) values.

Enumerator

WIDGET_PROPERTY_TYPE_NONE	Unknown or undefined type. This may also refer to an event rather than property.
WIDGET_PROPERTY_TYPE_INTEGER	32-bit signed integer.
WIDGET_PROPERTY_TYPE_INT64	64-bit signed integer.
WIDGET_PROPERTY_TYPE_FLOAT	32-bit floating-point type.
WIDGET_PROPERTY_TYPE_REAL	64-bit floating-point type.
WIDGET_PROPERTY_TYPE_BOOLEAN	Boolean value (either yes/no or true/false).
WIDGET_PROPERTY_TYPE_POINT	Floating-point 2D vector.
WIDGET_PROPERTY_TYPE_VECTOR	Floating-point 3D vector.
WIDGET_PROPERTY_TYPE_RECT	Rectangle defined by four floating-point values: left, top, width and height.
WIDGET_PROPERTY_TYPE_MARGINS	Margins defined by four floating-point values: left, top, right and bottom.
WIDGET_PROPERTY_TYPE_COLOR	Color value.
WIDGET_PROPERTY_TYPE_COLOR_PAIR	A pair of two colors.
WIDGET_PROPERTY_TYPE_COLOR_RECT	A set of four colors.
WIDGET_PROPERTY_TYPE_ENUMERATION	Enumeration (e.g. alignment) that is represented as a single byte.
WIDGET_PROPERTY_TYPE_STRING	UTF-8 encoded string.
WIDGET_PROPERTY_TYPE_FONT	Text font.

7.5.2.76 anonymous enum

anonymous enum

Widget property behavior enumeration that can be passed as one of [WidgetPropertyBehavior](#) values.

Enumerator

WIDGET_PROPERTY_BEHAVIOR_NORMAL	Standard property behavior.
WIDGET_PROPERTY_BEHAVIOR_STATIC	Property wraps an existing variable of the parent.
WIDGET_PROPERTY_BEHAVIOR_INDIRECT	Property is accessed through callback events.
WIDGET_PROPERTY_BEHAVIOR_VOLATILE	Property is considered temporary or "volatile".
WIDGET_PROPERTY_BEHAVIOR_EVENT	Property represents an event.

7.5.2.77 anonymous enum

anonymous enum

Widget property attribute bitmasks.

Enumerator

WIDGET_PROPERTY_ATTRIBUTE_ASSIGNED	The property is assigned: that is, it has an accompanying value.
------------------------------------	--

7.5.2.78 anonymous enum

anonymous enum

Widget manager texture type enumeration that can be passed as one of [WidgetManagerTextureType](#) values.

Enumerator

WIDGET_MANAGER_TEXTURE_TYPE_CANVAS	Texture with multisampling that is used for rendering using canvas.
WIDGET_MANAGER_TEXTURE_TYPE_SCREEN	Final compositioning texture.
WIDGET_MANAGER_TEXTURE_TYPE_AUXILIARY	Auxiliary texture used for compositioning effects.

7.5.2.79 anonymous enum

anonymous enum

Pixel format enumeration that can be passed as one of [PixelFormat](#) values.

Enumerator

PIXEL_FORMAT_UNKNOWN	Unknown pixel format.
PIXEL_FORMAT_R8	8-bit unsigned normalized format with red channel only.
PIXEL_FORMAT_RG8	16-bit format with 8-bit unsigned normalized red and green channels.
PIXEL_FORMAT_RGBA8	32-bit format with 8-bit unsigned normalized red, green, blue and alpha channels.
PIXEL_FORMAT_R16	8-bit unsigned normalized format with red channel only.
PIXEL_FORMAT_RG16	32-bit format with 8-bit unsigned normalized red and green channels.
PIXEL_FORMAT_RGBA16	64-bit ARGB pixel format with each channel having 16 bits.
PIXEL_FORMAT_R8S	8-bit signed normalized format with red channel only.
PIXEL_FORMAT_RG8S	16-bit format with 8-bit signed normalized red and green channels.
PIXEL_FORMAT_RGBA8S	32-bit format with 8-bit signed normalized red, green, blue and alpha channels.
PIXEL_FORMAT_R16S	16-bit signed normalized format with red channel only.
PIXEL_FORMAT_RG16S	32-bit format with 16-bit signed normalized red and green channels.
PIXEL_FORMAT_RGBA16S	64-bit format with 16-bit signed normalized red, green, blue and alpha channels.

Enumerator

PIXEL_FORMAT_R8U	8-bit format with 8-bit unsigned integer red channel.
PIXEL_FORMAT_RG8U	16-bit format with 8-bit unsigned integer red and green channels.
PIXEL_FORMAT_RGBA8U	32-bit format with 8-bit unsigned integer red and green channels.
PIXEL_FORMAT_R16U	16-bit format with 16-bit unsigned integer red channel.
PIXEL_FORMAT_RG16U	32-bit format with 16-bit unsigned integer red and green channels.
PIXEL_FORMAT_RGBA16U	64-bit format with 16-bit unsigned integer red and green channels.
PIXEL_FORMAT_R32U	32-bit format with 32-bit unsigned integer red channel.
PIXEL_FORMAT_RG32U	64-bit format with 32-bit unsigned integer red and green channels.
PIXEL_FORMAT_RGBA32U	128-bit format with 32-bit unsigned integer red and green channels.
PIXEL_FORMAT_R8I	8-bit format with 8-bit signed integer red channel.
PIXEL_FORMAT_RG8I	16-bit format with 8-bit signed integer red and green channels.
PIXEL_FORMAT_RGBA8I	32-bit format with 8-bit signed integer red and green channels.
PIXEL_FORMAT_R16I	16-bit format with 16-bit signed integer red channel.
PIXEL_FORMAT_RG16I	32-bit format with 16-bit signed integer red and green channels.
PIXEL_FORMAT_RGBA16I	64-bit format with 16-bit signed integer red and green channels.
PIXEL_FORMAT_R32I	32-bit format with 32-bit signed integer red channel.
PIXEL_FORMAT_RG32I	64-bit format with 32-bit signed integer red and green channels.
PIXEL_FORMAT_RGBA32I	128-bit format with 16-bit unsigned integer red and green channels.
PIXEL_FORMAT_R16F	16-bit floating-point pixel format, which has only one component.
PIXEL_FORMAT_RG16F	32-bit floating-point pixel format containing two components with 16 bits each.
PIXEL_FORMAT_RGBA16F	64-bit floating-point ARGB pixel format with each component having 16 bits.
PIXEL_FORMAT_R32F	32-bit floating-point pixel format, which has only one component.
PIXEL_FORMAT_RG32F	64-bit floating-point pixel format containing two components with 32 bits each.
PIXEL_FORMAT_RGBA32F	128-bit floating-point ARGB pixel format with each component having 32 bits.
PIXEL_FORMAT_RG11B10F	32-bit format with 11-bit unsigned floating-point red and green channels, and 10-bit blue channel.
PIXEL_FORMAT_RGB9E5F	32-bit format with 9-bit floating-point red, green and blue channels with shared 5-bit exponent.
PIXEL_FORMAT_RGB10A2	32-bit format with 10-bit unsigned normalized red, green and blue channels, and 2-bit alpha-channel.
PIXEL_FORMAT_RGB10A2U	32-bit format with 10-bit unsigned integer red, green and blue channels, and 2-bit alpha-channel.
PIXEL_FORMAT_RGBX8	32-bit format with 8-bit unsigned normalized red, green and blue channels (8 bits unused).
PIXEL_FORMAT_RGBA8_SRGB	32-bit format with 8-bit unsigned normalized red, green, blue and alpha channels with sRGB curve.
PIXEL_FORMAT_BGRA8	32-bit format with 8-bit unsigned normalized blue, green, red and alpha channels.
PIXEL_FORMAT_BGRX8	32-bit format with 8-bit unsigned normalized blue, green and red channels (8 bits unused).
PIXEL_FORMAT_BGRA8_SRGB	32-bit format with 8-bit unsigned normalized blue, green, red and alpha channels with sRGB curve.
PIXEL_FORMAT_BGR10A2	32-bit format with 10-bit unsigned normalized blue, green and red channels, and 2-bit alpha-channel.
PIXEL_FORMAT_BGR10X2	32-bit format with 10-bit unsigned normalized blue, green and red channels (2 bits unused).

Enumerator

PIXEL_FORMAT_BGRA4	16-bit format with 4-bit unsigned normalized blue, green, red and alpha channels.
PIXEL_FORMAT_BGRX4	16-bit format with 4-bit unsigned normalized blue, green and red channels (4 bits unused).
PIXEL_FORMAT_B5G6R5	16-bit format with 5-bit unsigned normalized blue and red, and 6 bits for green channels.
PIXEL_FORMAT_BGR5A1	16-bit format with 5-bit unsigned normalized blue, green and red, 1 bit for alpha channels.
PIXEL_FORMAT_BGR5X1	16-bit format with 5-bit unsigned normalized blue, green and red channels (1 bit unused).
PIXEL_FORMAT_RGB8	24-bit format with 8-bit unsigned normalized red, green and blue channels.
PIXEL_FORMAT_BGR8	24-bit format with 8-bit unsigned normalized blue, green and red channels.
PIXEL_FORMAT_A8	8-bit alpha-channel pixel format.
PIXEL_FORMAT_L8	8-bit luminance pixel format.
PIXEL_FORMAT_L16	16-bit luminance pixel format.
PIXEL_FORMAT_LA4	8-bit luminance/alpha pixel format with 4 bits per channel.
PIXEL_FORMAT_LA8	16-bit luminance/alpha pixel format with 8 bits per channel.
PIXEL_FORMAT_I8	8-bit palette indexed format.
PIXEL_FORMAT_R_BC	Compressed format with red channel only.
PIXEL_FORMAT_RG_BC	Compressed format with red and green channels.
PIXEL_FORMAT_RGB_BC	Compressed format with red, green and blue channels.
PIXEL_FORMAT_RGBA_BC	Compressed format with red, green, blue and alpha channels.
PIXEL_FORMAT_RGB_BC_SRGB	Compressed format with red, green and blue channels with sRGB curve.
PIXEL_FORMAT_RGBA_BC_SRGB	Compressed format with red, green and blue channels with sRGB curve, and an alpha-channel.
PIXEL_FORMAT_D16	Depth/stencil format with 16-bit (unsigned normalized) depth values and no storage for stencil.
PIXEL_FORMAT_D24S8	Depth/stencil format with 24-bit (unsigned normalized) depth values and 8 bits for stencil.
PIXEL_FORMAT_D32F	Depth/stencil format with 32-bit (floating-point) depth values and no storage for stencil.
PIXEL_FORMAT_D32S8F	Depth/stencil format with 32-bit (floating-point) depth values and 8 bits for stencil.

7.5.2.80 Key

enum [Key](#)

Portable key type enumeration that can be passed as one of [Key](#) values.

Enumerator

KEY_NULL	
KEY_HOME	
KEY_END	
KEY_PRINT_SCREEN	

Enumerator

KEY_BACKSPACE	
KEY_TAB	
KEY_LINEFEED	
KEY_CLEAR	
KEY_RETURN	
KEY_PAGE_UP	
KEY_PAGE_DOWN	
KEY_CAPSLOCK	
KEY_NUM_LOCK	
KEY_PAUSE	
KEY_SCROLL_LOCK	
KEY_SYS_REQ	
KEY_CANCEL	
KEY_INSERT	
KEY_ESCAPE	
KEY_LEFT	
KEY_UP	
KEY_RIGHT	
KEY_DOWN	
KEY_SPACE	
KEY_DELETE	
KEY_F1	
KEY_F2	
KEY_F3	
KEY_F4	
KEY_F5	
KEY_F6	
KEY_F7	
KEY_F8	
KEY_F9	
KEY_F10	
KEY_F11	
KEY_F12	
KEY_SHIFT_LEFT	
KEY_SHIFT_RIGHT	
KEY_CTRL_LEFT	
KEY_CTRL_RIGHT	
KEY_ALT_LEFT	
KEY_ALT_RIGHT	
KEY_SUPER_LEFT	
KEY_SUPER_RIGHT	
KEY_NUMPAD_0	
KEY_NUMPAD_1	
KEY_NUMPAD_2	
KEY_NUMPAD_3	
KEY_NUMPAD_4	
KEY_NUMPAD_5	
KEY_NUMPAD_6	
KEY_NUMPAD_7	
KEY_NUMPAD_8	

Enumerator

KEY_NUMPAD_9	
KEY_MULTIPLY	
KEY_DIVIDE	
KEY_ADD	
KEY_SUBTRACT	
KEY_SEPARATOR	
KEY_DECIMAL	
KEY_SLEEP	
KEY_VOLUME_UP	
KEY_VOLUME_DOWN	
KEY_VOLUME_MUTE	
KEY_MEDIA_NEXT_TRACK	
KEY_MEDIA_PREV_TRACK	
KEY_MEDIA_STOP	
KEY_MEDIA_PLAY_PAUSE	
KEY_SELECT	
KEY_PRINT	
KEY_EXECUTE	
KEY_HELP	
KEY_APPS	

7.5.2.81 WidgetManagerAttribute

enum [WidgetManagerAttribute](#)

Widget manager cumulative attribute flags that define its behavior.

Enumerator

WIDGET_MANAGER_ATTRIBUTE_COMPOSITION	Widgets support of composition effects such as glass and blur.
WIDGET_MANAGER_ATTRIBUTE_DESIGN	Design mode bit that enables additional functionality for creating interfaces on the fly.

7.6 Afterwarp.Types.h

[Go to the documentation of this file.](#)

```

00001 /*
00002  * This file is part of Afterwarp Framework (http://afterwarp.io).
00003  * Copyright (c) 2015 - 2025 Dr. Yuriy Kotsarenko. All rights reserved.
00004  *
00005  * Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in
00006  * compliance with the License. You may obtain a copy of the License at
00007  * http://www.apache.org/licenses/LICENSE-2.0
00008  *
00009  * Unless required by applicable law or agreed to in writing, software distributed under the License
00010  * is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
00011  * implied.
00012  * See the License for the specific language governing permissions and limitations under the License.
00012  */

```

```
00013
00014 // Afterwarp.Types.h
00015 #pragma once
00016
00017 #include <stdbool.h>
00018 #include <stddef.h>
00019 #include <stdint.h>
00020
00022 typedef uint8_t PixelFormat;
00023
00025 enum {
00027     PIXEL_FORMAT_UNKNOWN,
00028
00029     // Unsigned Normalized formats.
00030
00032     PIXEL_FORMAT_R8,
00033
00035     PIXEL_FORMAT_RG8,
00036
00038     PIXEL_FORMAT_RGBA8,
00039
00041     PIXEL_FORMAT_R16,
00042
00044     PIXEL_FORMAT_RG16,
00045
00047     PIXEL_FORMAT_RGBA16,
00048
00049     // Signed Normalized formats.
00050
00052     PIXEL_FORMAT_R8S,
00053
00055     PIXEL_FORMAT_RG8S,
00056
00058     PIXEL_FORMAT_RGBA8S,
00059
00061     PIXEL_FORMAT_R16S,
00062
00064     PIXEL_FORMAT_RG16S,
00065
00067     PIXEL_FORMAT_RGBA16S,
00068
00069     // Unsigned formats.
00070
00072     PIXEL_FORMAT_R8U,
00073
00075     PIXEL_FORMAT_RG8U,
00076
00078     PIXEL_FORMAT_RGBA8U,
00079
00081     PIXEL_FORMAT_R16U,
00082
00084     PIXEL_FORMAT_RG16U,
00085
00087     PIXEL_FORMAT_RGBA16U,
00088
00090     PIXEL_FORMAT_R32U,
00091
00093     PIXEL_FORMAT_RG32U,
00094
00096     PIXEL_FORMAT_RGBA32U,
00097
00098     // Signed formats.
00099
00101     PIXEL_FORMAT_R8I,
00102
00104     PIXEL_FORMAT_RG8I,
00105
00107     PIXEL_FORMAT_RGBA8I,
00108
00110     PIXEL_FORMAT_R16I,
00111
00113     PIXEL_FORMAT_RG16I,
00114
00116     PIXEL_FORMAT_RGBA16I,
00117
00119     PIXEL_FORMAT_R32I,
00120
00122     PIXEL_FORMAT_RG32I,
00123
00125     PIXEL_FORMAT_RGBA32I,
00126
00127     // Floating-point formats.
00128
00130     PIXEL_FORMAT_R16F,
00131
00133     PIXEL_FORMAT_RG16F,
00134
```

```

00136     PIXEL_FORMAT_RGBA16F,
00137
00139     PIXEL_FORMAT_R32F,
00140
00142     PIXEL_FORMAT_RG32F,
00143
00145     PIXEL_FORMAT_RGBA32F,
00146
00147     // Specialized floating-point formats.
00148
00150     PIXEL_FORMAT_RG11B10F,
00151
00153     PIXEL_FORMAT_RGB9E5F,
00154
00155     // Specialized color formats.
00156
00158     PIXEL_FORMAT_RGB10A2,
00159
00161     PIXEL_FORMAT_RGB10A2U,
00162
00164     PIXEL_FORMAT_RGBX8,
00165
00167     PIXEL_FORMAT_RGBA8_SRGB,
00168
00170     PIXEL_FORMAT_BGRA8,
00171
00173     PIXEL_FORMAT_BGRX8,
00174
00176     PIXEL_FORMAT_BGRA8_SRGB,
00177
00179     PIXEL_FORMAT_BGR10A2,
00180
00182     PIXEL_FORMAT_BGR10X2,
00183
00185     PIXEL_FORMAT_BGRA4,
00186
00188     PIXEL_FORMAT_BGRX4,
00189
00191     PIXEL_FORMAT_B5G6R5,
00192
00194     PIXEL_FORMAT_BGR5A1,
00195
00197     PIXEL_FORMAT_BGR5X1,
00198
00199     // Storage formats (not hardware accelerated).
00200
00202     PIXEL_FORMAT_RGB8,
00203
00205     PIXEL_FORMAT_BGR8,
00206
00207     // Luminance and alpha formats that might require channel swizzling in shader.
00208
00210     PIXEL_FORMAT_A8,
00211
00213     PIXEL_FORMAT_L8,
00214
00216     PIXEL_FORMAT_L16,
00217
00219     PIXEL_FORMAT_LA4,
00220
00222     PIXEL_FORMAT_LA8,
00223
00224     // Legacy palette formats.
00225
00227     PIXEL_FORMAT_I8,
00228
00229     // Compressed formats.
00230
00232     PIXEL_FORMAT_R_BC,
00233
00235     PIXEL_FORMAT_RG_BC,
00236
00238     PIXEL_FORMAT_RGB_BC,
00239
00241     PIXEL_FORMAT_RGBA_BC,
00242
00244     PIXEL_FORMAT_RGB_BC_SRGB,
00245
00247     PIXEL_FORMAT_RGBA_BC_SRGB,
00248
00249     // Depth/stencil formats.
00250
00252     PIXEL_FORMAT_D16,
00253
00255     PIXEL_FORMAT_D24S8,
00256
00258     PIXEL_FORMAT_D32F,

```

```

00259
00261     PIXEL_FORMAT_D32S8F};
00262
00264 typedef uint8_t AlphaFormatRequest;
00265
00267 enum {
00269     ALPHA_FORMAT_REQUEST_DONT_CARE,
00270
00274     ALPHA_FORMAT_REQUEST_NON_PREMULTIPLIED,
00275
00280     ALPHA_FORMAT_REQUEST_PREMULTIPLIED};
00281
00282 // Device enumerations.
00283
00284 enum {
00286     LEGACY_BIT_MISSING_DEPTH_TEXTURE = 0x00000001,
00287
00289     LEGACY_BIT_DEPTH_FORMAT_RAWZ = 0x00000002,
00290
00292     LEGACY_BIT_DIFFERENT_BITDEPTH_MRT = 0x00000004,
00293 };
00294
00296 enum {
00298     DEVICE_CLEAR_LAYER_COLOR = 0x01,
00299
00301     DEVICE_CLEAR_LAYER_DEPTH = 0x02,
00302
00304     DEVICE_CLEAR_LAYER_STENCIL = 0x04};
00305
00307 enum {
00309     DEVICE_ATTRIBUTE_DEBUG = 0x01,
00310
00312     DEVICE_ATTRIBUTE_SOFTWARE = 0x02,
00313
00315     DEVICE_ATTRIBUTE_LEGACY = 0x04,
00316
00320     DEVICE_ATTRIBUTE_LIMITED_EXTENSIONS = 0x80};
00321
00323 enum {
00325     DEVICE_BEHAVIOR_PER_SAMPLE_SHADING = 0x00000001,
00326
00329     DEVICE_BEHAVIOR_DEPTH_CLIP_NEGATIVE = 0x00000002,
00330
00332     DEVICE_BEHAVIOR_COMPUTE = 0x00000004,
00333
00335     DEVICE_BEHAVIOR_TESSELLATION = 0x00000008,
00336
00338     DEVICE_BEHAVIOR_VARIABLE_RATE_REFRESH = 0x00000100,
00339
00341     DEVICE_BEHAVIOR_POST_DEPTH_COVERAGE = 0x00000200,
00342
00344     DEVICE_BEHAVIOR_FORCE_BUFFER_UNBIND = 0x00001000,
00345
00347     DEVICE_BEHAVIOR_DEPTH_CLEAR_BAD_PRECISION = 0x00002000,
00348
00350     DEVICE_BEHAVIOR_SIGNED_NORM_INT_FORMAT_BUGGED = 0x00004000
00351 };
00352
00354 enum {
00357     DEVICE_BARRIER_SHADER_BUFFER = 0x00000001,
00358
00361     DEVICE_BARRIER_TEXTURE_FETCH = 0x00000002,
00362
00367     DEVICE_BARRIER_SHADER_IMAGE_ACCESS = 0x00000004,
00368
00373     DEVICE_BARRIER_TEXTURE_UPDATE = 0x00000008,
00374
00379     DEVICE_BARRIER_BUFFER_UPDATE = 0x00000010,
00380
00384     DEVICE_BARRIER_DRAWABLES = 0x00000020,
00385
00387     DEVICE_BARRIER_ATOMIC_COUNTER = 0x00000040,
00388
00390     DEVICE_BARRIER_STRUCTURED = 0x00000080
00391 };
00392
00394 typedef uint8_t DeviceTechnology;
00395
00397 enum {
00399     DEVICE_TECHNOLOGY_UNKNOWN,
00400
00402     DEVICE_TECHNOLOGY_DIRECT3D,
00403
00405     DEVICE_TECHNOLOGY_OPENGL,
00406
00408     DEVICE_TECHNOLOGY_OPENGL_ES,
00409

```

```
00411     DEVICE_TECHNOLOGY_VULKAN,
00412
00414     DEVICE_TECHNOLOGY_METAL,
00415
00417     DEVICE_TECHNOLOGY_WEBGL,
00418
00420     DEVICE_TECHNOLOGY_SOFTWARE,
00421
00423     DEVICE_TECHNOLOGY_PROPRIETARY};
00424
00426 typedef uint8_t DevicePlatform;
00427
00429 enum
00430 {
00432     DEVICE_PLATFORM_UNKNOWN,
00433
00435     DEVICE_PLATFORM_WINDOWS,
00436
00438     DEVICE_PLATFORM_LINUX,
00439
00441     DEVICE_PLATFORM_UNIX,
00442
00444     DEVICE_PLATFORM_OSX,
00445
00447     DEVICE_PLATFORM_IOS,
00448
00450     DEVICE_PLATFORM_ANDROID
00451 };
00452
00454 enum {
00456     DEVICE_STATE_DEPTH_TEST = 0x0001,
00457
00459     DEVICE_STATE_DEPTH_WRITE = 0x0002,
00460
00463     DEVICE_STATE_STENCIL_TEST = 0x0004,
00464
00466     DEVICE_STATE_DEPTH_CLIP = 0x0008,
00467
00469     DEVICE_STATE_SCISSOR_CLIP = 0x0010,
00470
00472     DEVICE_STATE_WIREFRAME = 0x0020,
00473
00476     DEVICE_STATE_CULL_CLOCKWISE = 0x0040,
00477
00479     DEVICE_STATE_MULTISAMPLING = 0x0080,
00480
00482     DEVICE_STATE_LINE_ANTIALIAS = 0x0100,
00483
00485     DEVICE_STATE_COLOR_WRITE = 0x0200,
00486
00488     DEVICE_STATE_BLEND_ENABLE = 0x0400,
00489
00491     DEVICE_STATE_ALPHA_TO_COVERAGE = 0x0800,
00492
00494     DEVICE_STATE_CUBE_MAP_SEAMLESS = 0x1000,
00495
00498     DEVICE_STATE_PER_SAMPLE_SHADING = 0x2000
00499 };
00500
00502 typedef uint8_t TriangleFace;
00503
00505 enum {
00507     TRIANGLE_FACE_NONE,
00508
00510     TRIANGLE_FACE_BACK,
00511
00513     TRIANGLE_FACE_FRONT,
00514
00516     TRIANGLE_FACE_BOTH};
00517
00519 typedef uint8_t ComparisonFunc;
00520
00522 enum {
00524     COMPARISON_FUNC_ALWAYS,
00525
00527     COMPARISON_FUNC_LESS,
00528
00530     COMPARISON_FUNC_LESS_EQUAL,
00531
00533     COMPARISON_FUNC_GREATER,
00534
00536     COMPARISON_FUNC_GREATER_EQUAL,
00537
00539     COMPARISON_FUNC_EQUAL,
00540
00542     COMPARISON_FUNC_NOT_EQUAL,
00543
```

```
00545     COMPARISON_FUNC_NEVER};
00546
00548 typedef uint8_t StencilOp;
00549
00551 enum {
00553     STENCIL_OP_KEEP,
00554
00556     STENCIL_OP_ZERO,
00557
00559     STENCIL_OP_REPLACE,
00560
00562     STENCIL_OP_INVERT,
00563
00565     STENCIL_OP_INCREMENT,
00566
00568     STENCIL_OP_DECREMENT,
00569
00571     STENCIL_OP_INCREMENT_WARP,
00572
00574     STENCIL_OP_DECREMENT_WARP};
00575
00577 typedef uint8_t BlendFactor;
00578
00580 enum {
00582     BLEND_FACTOR_ONE,
00583
00585     BLEND_FACTOR_ZERO,
00586
00588     BLEND_FACTOR_SOURCE_COLOR,
00589
00591     BLEND_FACTOR_INV_SOURCE_COLOR,
00592
00594     BLEND_FACTOR_SOURCE_ALPHA,
00595
00597     BLEND_FACTOR_INV_SOURCE_ALPHA,
00598
00600     BLEND_FACTOR_DEST_COLOR,
00601
00603     BLEND_FACTOR_INV_DEST_COLOR,
00604
00606     BLEND_FACTOR_DEST_ALPHA,
00607
00609     BLEND_FACTOR_INV_DEST_ALPHA,
00610
00612     BLEND_FACTOR_SOURCE_ALPHA_SAT,
00613
00615     BLEND_FACTOR_CONSTANT_COLOR,
00616
00618     BLEND_FACTOR_INV_CONSTANT_COLOR,
00619
00621     BLEND_FACTOR_CONSTANT_ALPHA,
00622
00624     BLEND_FACTOR_INV_CONSTANT_ALPHA,
00625
00627     BLEND_FACTOR_SOURCE_COLOR_1,
00628
00630     BLEND_FACTOR_INV_SOURCE_COLOR_1,
00631
00633     BLEND_FACTOR_SOURCE_ALPHA_1,
00634
00636     BLEND_FACTOR_INV_SOURCE_ALPHA_1};
00637
00639 typedef uint8_t BlendOp;
00640
00642 enum {
00644     BLEND_OP_ADD,
00645
00647     BLEND_OP_SUBTRACT,
00648
00650     BLEND_OP_INV_SUBTRACT,
00651
00653     BLEND_OP_MINIMUM,
00654
00656     BLEND_OP_MAXIMUM};
00657
00658 // Buffer enumerations.
00659
00661 typedef uint8_t BufferDataType;
00662
00664 enum {
00666     BUFFER_DATA_TYPE_VERTEX,
00667
00669     BUFFER_DATA_TYPE_INDEX,
00670
00672     BUFFER_DATA_TYPE_CONSTANT,
00673
00675     BUFFER_DATA_TYPE_TYPED,
```

```
00676
00678     BUFFER_DATA_TYPE_RW_TYPED,
00679
00681     BUFFER_DATA_TYPE_STRUCTURED,
00682
00684     BUFFER_DATA_TYPE_RW_STRUCTURED};
00685
00687 typedef uint8_t BufferAccessType;
00688
00690 enum {
00692     BUFFER_ACCESS_TYPE_DEFAULT,
00693
00695     BUFFER_ACCESS_TYPE_STATIC,
00696
00698     BUFFER_ACCESS_TYPE_DYNAMIC,
00699
00701     BUFFER_ACCESS_TYPE_COMPUTE,
00702
00704     BUFFER_ACCESS_TYPE_STAGING};
00705
00706 // Program enumerations.
00707
00709 typedef uint8_t PrimitiveTopology;
00710
00712 enum {
00714     PRIMITIVE_TOPOLOGY_UNKNOWN,
00715
00717     PRIMITIVE_TOPOLOGY_POINTS,
00718
00720     PRIMITIVE_TOPOLOGY_LINES,
00721
00723     PRIMITIVE_TOPOLOGY_LINE_STRIP,
00724
00726     PRIMITIVE_TOPOLOGY_TRIANGLES,
00727
00729     PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP,
00730
00732     PRIMITIVE_TOPOLOGY_LINES_ADJACENCY,
00733
00735     PRIMITIVE_TOPOLOGY_LINE_STRIP_ADJACENCY,
00736
00738     PRIMITIVE_TOPOLOGY_TRIANGLES_ADJACENCY,
00739
00741     PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_ADJACENCY};
00742
00744 typedef uint16_t ElementFormat;
00745
00747 enum {
00749     ELEMENT_FORMAT_UNDEFINED,
00750
00752     ELEMENT_FORMAT_FLOAT,
00753
00755     ELEMENT_FORMAT_HALF_FLOAT,
00756
00758     ELEMENT_FORMAT_DOUBLE,
00759
00761     ELEMENT_FORMAT_INT,
00762
00764     ELEMENT_FORMAT_UNSIGNED_INT,
00765
00767     ELEMENT_FORMAT_SHORT,
00768
00770     ELEMENT_FORMAT_UNSIGNED_SHORT,
00771
00773     ELEMENT_FORMAT_BYTE,
00774
00776     ELEMENT_FORMAT_UNSIGNED_BYTE,
00777
00780     ELEMENT_FORMAT_FLOAT_11_11_10};
00781
00783 typedef uint8_t ShaderType;
00784
00786 enum {
00788     SHADER_TYPE_UNDEFINED,
00789
00791     SHADER_TYPE_FRAGMENT,
00792
00794     SHADER_TYPE_VERTEX,
00795
00797     SHADER_TYPE_GEOMETRY,
00798
00800     SHADER_TYPE_COMPUTE,
00801
00803     SHADER_TYPE_HULL,
00804
00806     SHADER_TYPE_DOMAIN};
00807
```

```

00809 typedef uint8_t ShaderElement;
00810
00812 enum {
00814     SHADER_ELEMENT_UNDEFINED,
00815
00817     SHADER_ELEMENT_TEXTURE,
00818
00820     SHADER_ELEMENT_CONSTANT_BUFFER,
00821
00823     SHADER_ELEMENT_TYPED_BUFFER,
00824
00828     SHADER_ELEMENT_RW_TYPED_BUFFER,
00829
00831     SHADER_ELEMENT_STRUCTURED_BUFFER,
00832
00834     SHADER_ELEMENT_RW_STRUCTURED_BUFFER};
00835
00836 enum
00837 {
00839     VERTEX_CHANNEL_INDEX_BUFFER = 0x7FFFFFFF,
00840
00842     VERTEX_CHANNEL_NO_BUFFERS = 0x7FFFFFFE,
00843
00845     SHADER_INDEX_UNDEFINED = 0x7FFFFFFF,
00846
00848     VERTEX_CHANNEL_NORMALIZED = 0x80000000
00849 };
00850
00851 // Compute enumerations.
00852
00854 typedef uint8_t ComputeTextureAccess;
00855
00857 enum {
00859     COMPUTE_TEXTURE_ACCESS_READ,
00860
00862     COMPUTE_TEXTURE_ACCESS_WRITE,
00863
00865     COMPUTE_TEXTURE_ACCESS_READ_WRITE
00866 };
00867
00868 // Sampler enumerations.
00869
00871 typedef uint8_t TextureFilter;
00872
00874 enum {
00876     TEXTURE_FILTER_NONE,
00877
00879     TEXTURE_FILTER_NEAREST,
00880
00882     TEXTURE_FILTER_LINEAR,
00883
00885     TEXTURE_FILTER_ANISOTROPIC};
00886
00888 typedef uint8_t TextureAddress;
00889
00891 enum {
00893     TEXTURE_ADDRESS_WRAP,
00894
00896     TEXTURE_ADDRESS_CLAMP,
00897
00899     TEXTURE_ADDRESS_MIRROR,
00900
00902     TEXTURE_ADDRESS_MIRROR_ONCE,
00903
00905     TEXTURE_ADDRESS_BORDER};
00906
00907 // Texture enumerations.
00908
00910 enum {
00913     TEXTURE_ATTRIBUTE_MIPMAPPING = 0x01,
00914
00917     TEXTURE_ATTRIBUTE_DRAWABLE = 0x02,
00918
00921     TEXTURE_ATTRIBUTE_DYNAMIC = 0x04,
00922
00928     TEXTURE_ATTRIBUTE_PREMULTIPLIED_ALPHA = 0x08,
00929
00931     TEXTURE_COMPUTE = 0x40,
00932
00935     TEXTURE_SCRATCH = 0x8000};
00936
00938 typedef uint8_t TextureType;
00939
00941 enum {
00943     TEXTURE_TYPE_SURFACE,
00944
00946     TEXTURE_TYPE_CUBE_MAP,

```



```

00947
00949     TEXTURE_TYPE_VOLUME};
00950
00951 // Canvas enumerations.
00952
00954 typedef uint8_t BlendingEffect;
00955
00957 enum {
00960     BLENDING_EFFECT_UNDEFINED = 0,
00961
00963     BLENDING_EFFECT_NORMAL,
00964
00966     BLENDING_EFFECT_SHADOW,
00967
00969     BLENDING_EFFECT_ADD,
00970
00972     BLENDING_EFFECT_SUBTRACT,
00973
00975     BLENDING_EFFECT_MULTIPLY,
00976
00978     BLENDING_EFFECT_INVERSE_MULTIPLY,
00979
00981     BLENDING_EFFECT_SOURCE_COLOR,
00982
00984     BLENDING_EFFECT_SOURCE_COLOR_ADD,
00985
00987     BLENDING_EFFECT_COPY = 254,
00988
00990     BLENDING_EFFECT_CUSTOM = 255};
00991
00993 typedef uint8_t CanvasContextState;
00994
00996 enum {
00998     CANVAS_CONTEXT_STATE_FLAT_SCENE,
00999
01001     CANVAS_CONTEXT_STATE_FLAT_TEXT,
01002
01004     CANVAS_CONTEXT_STATE_PROJECTED,
01005
01007     CANVAS_CONTEXT_STATE_UNDEFINED = UINT8_MAX};
01008
01010 enum {
01013     CANVAS_ATTRIBUTE_PROJECTED = 0x01,
01014
01016     CANVAS_ATTRIBUTE_SDF = 0x02,
01017
01019     CANVAS_ATTRIBUTE_OUTLINE_SDF = 0x04,
01020
01022     CANVAS_ATTRIBUTE_CUBIC = 0x08,
01023
01025     CANVAS_ATTRIBUTE_COLOR_ADJUST = 0x10,
01026
01028     CANVAS_ATTRIBUTE_TRANSFORM = 0x20};
01029
01031 typedef uint8_t SuperSampleSDF;
01032
01034 enum {
01038     SUPER_SAMPLE_SDF_NO_SUPER_SAMPLE,
01039
01041     SUPER_SAMPLE_SDF_SUPER_SAMPLE_4X,
01042
01044     SUPER_SAMPLE_SDF_SUPER_SAMPLE_16X};
01045
01047 enum
01048 {
01050     IMAGE_REGION_FLIP = 0x01,
01051
01053     IMAGE_REGION_MIRROR = 0x02,
01054
01056     IMAGE_REGION_ROTATE = 0x04
01057 };
01058
01059 // Vector graphics enumerations.
01060
01062 typedef uint8_t PathJoint;
01063
01065 enum {
01067     PATH_JOINT_NONE,
01068
01070     PATH_JOINT_SIMPLE,
01071
01073     PATH_JOINT_MITER,
01074
01076     PATH_JOINT_BEVEL,
01077
01079     PATH_JOINT_ROUND,
01080

```

```
01082     PATH_JOINT_MITER_BEVEL};
01083
01085 typedef uint8_t LineCaps;
01086
01088 enum {
01090     LINE_CAPS_BUTT,
01091
01093     LINE_CAPS_SQUARE,
01094
01096     LINE_CAPS_ROUND};
01097
01099 typedef uint8_t PointShape;
01100
01102 enum {
01104     POINT_SHAPE_SQUARE,
01105
01107     POINT_SHAPE_ROUND,
01108
01110     POINT_SHAPE_TRIANGLE,
01111
01113     POINT_SHAPE_STAR,
01114
01116     POINT_SHAPE_CROSS};
01117
01118 // Font enumerations.
01119
01121 typedef uint8_t FontWeight;
01122
01124 enum {
01126     FONT_WEIGHT_THIN,
01127
01129     FONT_WEIGHT_EXTRA_LIGHT,
01130
01132     FONT_WEIGHT_LIGHT,
01133
01135     FONT_WEIGHT_SEMI_LIGHT,
01136
01138     FONT_WEIGHT_NORMAL,
01139
01141     FONT_WEIGHT_MEDIUM,
01142
01144     FONT_WEIGHT_SEMI_BOLD,
01145
01147     FONT_WEIGHT_BOLD,
01148
01150     FONT_WEIGHT_EXTRA_BOLD,
01151
01153     FONT_WEIGHT_HEAVY,
01154
01156     FONT_WEIGHT_EXTRA_HEAVY};
01157
01159 typedef uint8_t FontStretch;
01160
01162 enum {
01164     FONT_STRETCH_ULTRA_CONDENSED,
01165
01167     FONT_STRETCH_EXTRA_CONDENSED,
01168
01170     FONT_STRETCH_CONDENSED,
01171
01173     FONT_STRETCH_SEMI_CONDENSED,
01174
01176     FONT_STRETCH_NORMAL,
01177
01179     FONT_STRETCH_SEMI_EXPANDED,
01180
01182     FONT_STRETCH_EXPANDED,
01183
01185     FONT_STRETCH_EXTRA_EXPANDED,
01186
01188     FONT_STRETCH_ULTRA_EXPANDED};
01189
01191 typedef uint8_t FontSlant;
01192
01194 enum {
01196     FONT_SLANT_NONE,
01197
01199     FONT_SLANT_OBLIQUE,
01200
01202     FONT_SLANT_ITALIC};
01203
01205 typedef uint8_t FontBorder;
01206
01208 typedef uint8_t FontAttributes;
01209
01211 enum
01212 {
```

```

01214     FONT_ATTRIBUTE_UNDERLINE = 0x01,
01215
01217     FONT_ATTRIBUTE_STRIKE_OUT = 0x02
01218 };
01219
01221 enum {
01223     FONT_BORDER_NONE,
01224
01226     FONT_BORDER_NORMAL,
01227
01229     FONT_BORDER_SEMI_HEAVY,
01230
01232     FONT_BORDER_HEAVY};
01233
01235 typedef uint8_t TextAlignment;
01236
01238 enum {
01240     TEXT_ALIGNMENT_START,
01241
01243     TEXT_ALIGNMENT_MIDDLE,
01244
01246     TEXT_ALIGNMENT_END};
01247
01248 // Post-processing enumerations.
01249
01251 typedef uint8_t ColorDitheringFormat;
01252
01254 enum {
01256     COLOR_DITHERING_FORMAT_RGB8,
01257
01259     COLOR_DITHERING_FORMAT_RGB6,
01260
01262     COLOR_DITHERING_FORMAT_R5G6B5,
01263
01265     COLOR_DITHERING_FORMAT_RGB4,
01266
01268     COLOR_DITHERING_FORMAT_RG3B2};
01269
01271 typedef uint8_t SelectionHighlightTextureType;
01272
01274 enum {
01276     SELECTION_HIGHLIGHT_TEXTURE_HIGHLIGHT,
01277
01279     SELECTION_HIGHLIGHT_TEXTURE_HIGHLIGHT_MASK
01280 };
01281
01282 // Fog enumerations.
01283
01285 typedef uint8_t FogFormula;
01286
01288 enum {
01290     FOG_FORMULA_LINEAR,
01291
01293     FOG_FORMULA_EXPONENTIAL,
01294
01296     FOG_FORMULA_GROUND};
01297
01299 typedef uint8_t DepthFogDistance;
01300
01302 enum {
01304     DEPTH_FOG_DISTANCE_Z_BASED,
01305
01307     DEPTH_FOG_DISTANCE_EYE_RELATIVE};
01308
01309 // Scene enumerations.
01310
01312 typedef uint8_t TechniqueLighting;
01313
01315 enum {
01317     TECHNIQUE_LIGHTING_PHONG,
01318
01320     TECHNIQUE_LIGHTING_MINNAERT,
01321
01323     TECHNIQUE_LIGHTING_COOK_TORRANCE,
01324
01326     TECHNIQUE_LIGHTING_OREN_NAYER};
01327
01329 typedef uint8_t TechniqueShadows;
01330
01332 enum {
01334     TECHNIQUE_SHADOWS_NONE,
01335
01337     TECHNIQUE_SHADOWS_ESM,
01338
01340     TECHNIQUE_SHADOWS_ESM_WARP,
01341
01343     TECHNIQUE_SHADOWS_EVSM

```

```

01344 };
01345
01347 typedef uint8_t ClustersCullingMode;
01348
01350 enum {
01352     CLUSTERS_CULLING_MODE_QUALITY,
01353     CLUSTERS_CULLING_MODE_PERFORMANCE};
01356
01358 typedef uint8_t TextureFidelity;
01359
01361 enum {
01364     TEXTURE_FIDELITY_LOW,
01365     TEXTURE_FIDELITY_MEDIUM,
01366     TEXTURE_FIDELITY_HIGH,
01367     TEXTURE_FIDELITY_EXTREME};
01371
01374 TEXTURE_FIDELITY_EXTREME};
01375
01377 typedef uint8_t TextureCabinetType;
01378
01380 enum {
01382     TEXTURE_CABINET_TYPE_DEPTH = 0,
01383     TEXTURE_CABINET_TYPE_NORMALS = 1,
01384     TEXTURE_CABINET_TYPE_COLOR_HDR = 4,
01385     TEXTURE_CABINET_TYPE_COLOR = 5,
01386     TEXTURE_CABINET_TYPE_LINEAR_DEPTHS = 11};
01395
01397 typedef uint8_t TextureCabinetPass;
01398
01400 enum {
01402     TEXTURE_CABINET_PASS_COLOR,
01403     TEXTURE_CABINET_PASS_GLASSY,
01404     TEXTURE_CABINET_PASS_DEPTH};
01408
01409
01411 typedef uint8_t TextureCabinetFilterType;
01412
01414 enum {
01416     TEXTURE_CABINET_FILTER_TYPE_OCCLUSION,
01417     TEXTURE_CABINET_FILTER_TYPE_BLOOM,
01418     TEXTURE_CABINET_FILTER_TYPE_GLASSY,
01419     TEXTURE_CABINET_FILTER_TYPE_GLASSY_FOG};
01426
01428 enum {
01430     TEXTURE_CABINET_ATTRIBUTE_AMBIENT_OCCLUSION = 0x0001,
01431     TEXTURE_CABINET_ATTRIBUTE_AMBIENT_OCCLUSION_DOWNSCALE = 0x0002,
01432     TEXTURE_CABINET_ATTRIBUTE_BLOOM = 0x0004,
01433     TEXTURE_CABINET_ATTRIBUTE_BLOOM_DOWNSCALE = 0x0008,
01434     TEXTURE_CABINET_ATTRIBUTE_BLOOM_CUBIC = 0x0010,
01435     TEXTURE_CABINET_ATTRIBUTE_LINEAR_DEPTHS = 0x0020,
01436     TEXTURE_CABINET_ATTRIBUTE_GLASSY = 0x0040,
01437     TEXTURE_CABINET_ATTRIBUTE_GLASSY_FAST = 0x0080,
01438     TEXTURE_CABINET_ATTRIBUTE_GLASSY_FOG = 0x0100,
01439     TEXTURE_CABINET_ATTRIBUTE_GLASSY_FROSTED = 0x0200};
01458
01460 enum {
01462     SCENE_ATTRIBUTE_INSTANCING = 0x01,
01463     SCENE_ATTRIBUTE_NORMALS = 0x02,
01464     SCENE_ATTRIBUTE_COLORING = 0x04,
01465     SCENE_ATTRIBUTE_TEXTUREING_CUBIC = 0x08,
01466     SCENE_ATTRIBUTE_SHADOWS_CUBIC = 0x10,
01467     SCENE_ATTRIBUTE_GLASSY = 0x20,
01468

```

```

01480     SCENE_ATTRIBUTE_SINGLE_PASS = 0x40,
01481
01482     // Attributes specific to depth/normals scene.
01483
01486     SCENE_ATTRIBUTE_DEPTH_PREPASS = 0x80,
01487
01490     SCENE_ATTRIBUTE_LINEAR_DEPTHS = 0x100,
01491
01492     // Status attributes.
01493
01495     SCENE_ATTRIBUTE_MODELING = 0x1000};
01496
01498 typedef uint8_t SceneTextureType;
01499
01501 enum {
01503     SCENE_TEXTURE_TYPE_ALBEDO,
01504
01506     SCENE_TEXTURE_TYPE_NORMAL_MAP,
01507
01509     SCENE_TEXTURE_TYPE_PARALLAX_MAP};
01510
01511
01513 typedef uint8_t SceneSamplerType;
01514
01516 enum {
01518     SCENE_SAMPLER_TYPE_ALBEDO,
01519
01521     SCENE_SAMPLER_TYPE_NORMAL_MAP,
01522
01524     SCENE_SAMPLER_TYPE_PARALLAX_MAP,
01525
01527     SCENE_SAMPLER_TYPE_SHADOW_MAP};
01528
01530 typedef uint8_t ModelTransform;
01531
01532 // MeshMetaTagType enumerations.
01533
01535 enum {
01537     MESH_META_TAG_TYPE_INDETERMINATE = 0,
01538
01540     MESH_META_TAG_TYPE_GEOMETRY = 1,
01541
01543     MESH_META_TAG_TYPE_OBJECT = 2
01544 };
01545
01546 // SceneMesh enumerations.
01547
01549 enum {
01551     SCENE_MESH_TEXTURE_DIFFUSE = 0,
01552
01554     SCENE_MESH_TEXTURE_NORMALS = 1,
01555
01557     SCENE_MESH_TEXTURE_PARALLAX = 2
01558 };
01559
01561 enum {
01563     SCENE_MESH_LATCH_TYPE_JOINT,
01564
01566     SCENE_MESH_LATCH_TYPE_WAYPOINT
01567 };
01568
01570 enum {
01572     SCENE_MESH_VERTEX_ELEMENTS_INDEX_UNDEFINED = 0xFF,
01573 };
01574
01576 enum {
01579     AUTO_DRAW_OPTION_POST_UNBIND = 0x01,
01580
01583     AUTO_DRAW_OPTION_RESET_TEXTURES = 0x02,
01584
01587     AUTO_DRAW_OPTION_MATERIAL_IGNORE = 0x04,
01588
01592     AUTO_DRAW_OPTION_MATERIAL_TRANSPARENCY = 0x08,
01593
01595     AUTO_DRAW_OPTION_MATERIAL_SKIP_TRANSPARENT = 0x10,
01596
01598     AUTO_DRAW_OPTION_MATERIAL_SKIP_OPAQUE = 0x20,
01599
01601     AUTO_DRAW_OPTION_OBJECT_SKIP_TRANSPARENT = 0x40,
01602
01604     AUTO_DRAW_OPTION_OBJECT_SKIP_NON_TRANSPARENT = 0x80,
01605
01607     AUTO_DRAW_OPTION_OBJECT_SKIP_HAVING_MATERIALS = 0x100,
01608
01610     AUTO_DRAW_OPTION_OBJECT_SKIP_NOT_HAVING_MATERIALS = 0x200,
01611
01613     AUTO_DRAW_OPTION_OBJECT_HIGHLIGHTED = 0x400,

```

```
01614
01616     AUTO_DRAW_OPTION_OBJECT_DISABLE_INSTANCING = 0x800
01617 };
01618
01619 // ObjectModel enumerations.
01620
01621 enum {
01622     MODEL_TRANSFORM_LOCAL,
01623     MODEL_TRANSFORM_LOCAL_MODEL,
01624     MODEL_TRANSFORM_LOCAL_VOLUME,
01625     MODEL_TRANSFORM_GLOBAL,
01626     MODEL_TRANSFORM_GLOBAL_MODEL,
01627     MODEL_TRANSFORM_GLOBAL_VOLUME
01628 };
01629
01630 typedef uint8_t MeshAlign;
01631
01632 enum {
01633     MESH_ALIGN_POSITIVE,
01634     MESH_ALIGN_ORIGIN,
01635     MESH_ALIGN_NEGATIVE,
01636     MESH_ALIGN_UNALIGNED};
01637
01638 typedef uint8_t ObjectModelViewCompare;
01639
01640 enum {
01641     OBJECT_MODEL_COMPARE_DEPTH,
01642     OBJECT_MODEL_COMPARE_DEPTH_REVERSE,
01643     OBJECT_MODEL_COMPARE_ORDERED,
01644     OBJECT_MODEL_COMPARE_MESHES,
01645     OBJECT_MODEL_COMPARE_OBJECTS
01646 };
01647
01648 enum {
01649     OBJECT_MODEL_ATTRIBUTE_VISIBLE = 0x01,
01650     OBJECT_MODEL_ATTRIBUTE_SELECTABLE = 0x02,
01651     OBJECT_MODEL_ATTRIBUTE_TRANSPARENT = 0x04,
01652     OBJECT_MODEL_ATTRIBUTE_HIERARCHYLESS = 0x08,
01653     OBJECT_MODEL_ATTRIBUTE_DISABLE_REALIGN = 0x10,
01654     OBJECT_MODEL_ATTRIBUTE_NON_VIEWABLE = 0x20
01655 };
01656
01657 enum {
01658     MESH_LOADING_OPTION_MATERIAL_VERTEX_COLOR = 0x01,
01659     MESH_LOADING_OPTION_MATERIAL_VERTEX_COLOR_PREFERRED = 0x02,
01660     MESH_LOADING_OPTION_MATERIAL_TEXTURE_MISSING = 0x80,
01661     MESH_LOADING_OPTION_EXPORT_NORMALS = 0x100,
01662     MESH_LOADING_OPTION_EXPORT_TANGENTS = 0x200,
01663     MESH_LOADING_OPTION_EXPORT_TEXTURE_COORDINATES = 0x400,
01664     MESH_LOADING_OPTION_EXPORT_COLORS = 0x800,
01665     MESH_LOADING_OPTION_EXPORT_WEIGHTS = 0x1000,
01666     MESH_LOADING_OPTION_STRIP_GEOMETRY_NAMES = 0x2000};
01667
01668 // ActorCamera enumerations.
01669
01670 typedef uint8_t CameraCommand;
01671
01672 enum {
01673     CAMERA_COMMAND_MOVEMENT,
01674     CAMERA_COMMAND_DRAGGING,
```

```

01750     CAMERA_COMMAND_ROTATION,
01751
01753     CAMERA_COMMAND_UPDATE,
01754
01756     CAMERA_COMMAND_STOP
01757 };
01758
01759 // Application enumerations.
01760
01762 typedef uint8_t ApplicationWindowState;
01763
01765 enum {
01767     APPLICATION_WINDOW_STATE_WINDOWED,
01768
01770     APPLICATION_WINDOW_STATE_MINIMIZED,
01771
01773     APPLICATION_WINDOW_STATE_MAXIMIZED,
01774
01776     APPLICATION_WINDOW_STATE_FULLSCREEN
01777 };
01778
01780 typedef uint8_t MouseEvent;
01781
01783 enum {
01785     MOUSE_EVENT_MOUSE_DOWN,
01786
01788     MOUSE_EVENT_MOUSE_MOVE,
01789
01791     MOUSE_EVENT_MOUSE_UP,
01792
01794     MOUSE_EVENT_WHEEL_UP,
01795
01797     MOUSE_EVENT_WHEEL_DOWN,
01798
01800     MOUSE_EVENT_MOUSE_ENTER,
01801
01803     MOUSE_EVENT_MOUSE_LEAVE};
01804
01806 typedef uint8_t MouseButton;
01807
01809 enum {
01811     MOUSE_BUTTON_NONE,
01812
01814     MOUSE_BUTTON_LEFT,
01815
01817     MOUSE_BUTTON_RIGHT,
01818
01820     MOUSE_BUTTON_MIDDLE};
01821
01823 typedef uint8_t KeyEvent;
01824
01826 enum {
01828     KEY_EVENT_PRESSED,
01829
01831     KEY_EVENT_RELEASED,
01832
01834     KEY_EVENT_TYPING};
01835
01837 typedef uint8_t ApplicationEvent;
01838
01840 enum {
01842     APPLICATION_EVENT_ACTIVATE,
01843
01845     APPLICATION_EVENT_DEACTIVATE,
01846
01848     APPLICATION_EVENT_MINIMIZE,
01849
01851     APPLICATION_EVENT_RESTORE,
01852
01854     APPLICATION_EVENT_APPEARANCE};
01855
01857 typedef uint8_t Key;
01858
01860 enum Key
01861 {
01862     KEY_NULL = 0x00,
01863     // 0x01: SOH (Start of Heading)
01864     KEY_HOME = 0x02, // STX (Start of Text)
01865     KEY_END = 0x03, // ETX (End of Text)
01866     // 0x04: EOT (End of Transmission)
01867     KEY_PRINT_SCREEN = 0x05, // ENQ (Enquiry)
01868     // 0x06: ACK (Acknowledge)
01869     // 0x07: BEL (Bell)
01870     KEY_BACKSPACE = 0x08,
01871     KEY_TAB = 0x09,
01872     KEY_LINEFEED = 0x0A, // LF, NL (Newline)
01873     KEY_CLEAR = 0x0B, // VT (Vertical Tab)

```

```

01874 // 0x0C: FF (NP form feed, new page)
01875 KEY_RETURN = 0x0D, // Enter, CR (Carriage Return)
01876 KEY_PAGE_UP = 0x0E, // SO (Shift Out)
01877 KEY_PAGE_DOWN = 0x0F, // SI (Shift In)
01878 // 0x10: DLE (Data Link Escape)
01879 KEY_CAPSLOCK = 0x11, // DC1 (Device Control 1)
01880 KEY_NUM_LOCK = 0x12, // DC2 (Device Control 2)
01881 KEY_PAUSE = 0x13, // Hold, DC3 (Device Control 3)
01882 KEY_SCROLL_LOCK = 0x14, // DC4 (Device Control 4)
01883 KEY_SYS_REQ = 0x15, // SysRq, NAK (Negative Acknowledge)
01884 // 0x16: SYN (Synchronous Idle)
01885 // 0x17: ETB (End of Trans. Block)
01886 KEY_CANCEL = 0x18,
01887 // 0x19: EM (End of Medium)
01888 KEY_INSERT = 0x1A, // SUB (Substitute)
01889 KEY_ESCAPE = 0x1B,
01890 KEY_LEFT = 0x1C, // 0x1C: FS (File Separator)
01891 KEY_UP = 0x1D, // 0x1D: GS (Group Separator)
01892 KEY_RIGHT = 0x1E, // 0x1E: RS (Record Separator)
01893 KEY_DOWN = 0x1F, // 0x1F: US (Unit Separator)
01894 KEY_SPACE = 0x20,
01895 // 0x21 - 0x7E: ASCII characters.
01896 KEY_DELETE = 0x7F, // DEL
01897 // Extended keys
01898 KEY_F1 = 0x80,
01899 KEY_F2 = 0x81,
01900 KEY_F3 = 0x82,
01901 KEY_F4 = 0x83,
01902 KEY_F5 = 0x84,
01903 KEY_F6 = 0x85,
01904 KEY_F7 = 0x86,
01905 KEY_F8 = 0x87,
01906 KEY_F9 = 0x88,
01907 KEY_F10 = 0x89,
01908 KEY_F11 = 0x8A,
01909 KEY_F12 = 0x8B,
01910 // 0x8C - 0x97 (F13-F24 keys)
01911 // Control keys
01912 KEY_SHIFT_LEFT = 0x98,
01913 KEY_SHIFT_RIGHT = 0x99,
01914 KEY_CTRL_LEFT = 0x9A,
01915 KEY_CTRL_RIGHT = 0x9B,
01916 KEY_ALT_LEFT = 0x9C,
01917 KEY_ALT_RIGHT = 0x9D,
01918 KEY_SUPER_LEFT = 0x9E, // Left Windows key
01919 KEY_SUPER_RIGHT = 0x9F, // Right Windows key
01920 // Numpad keys
01921 KEY_NUMPAD_0 = 0xA0,
01922 KEY_NUMPAD_1 = 0xA1,
01923 KEY_NUMPAD_2 = 0xA2,
01924 KEY_NUMPAD_3 = 0xA3,
01925 KEY_NUMPAD_4 = 0xA4,
01926 KEY_NUMPAD_5 = 0xA5,
01927 KEY_NUMPAD_6 = 0xA6,
01928 KEY_NUMPAD_7 = 0xA7,
01929 KEY_NUMPAD_8 = 0xA8,
01930 KEY_NUMPAD_9 = 0xA9,
01931 // Numpad operators
01932 KEY_MULTIPLY = 0xAA,
01933 KEY_DIVIDE = 0xAB,
01934 KEY_ADD = 0xAC,
01935 KEY_SUBTRACT = 0xAD,
01936 // Numpad misc
01937 KEY_SEPARATOR = 0xAE,
01938 KEY_DECIMAL = 0xAF,
01939 // 0xB0 - 0xBF: reserved
01940 // Power keys
01941 KEY_SLEEP = 0xC0,
01942 // 0xC1 - 0xCB: reserved
01943 KEY_VOLUME_UP = 0xCC,
01944 KEY_VOLUME_DOWN = 0xCE,
01945 KEY_VOLUME_MUTE = 0xCF,
01946 // Media playback keys
01947 KEY_MEDIA_NEXT_TRACK = 0xD0,
01948 KEY_MEDIA_PREV_TRACK = 0xD1,
01949 KEY_MEDIA_STOP = 0xD2,
01950 KEY_MEDIA_PLAY_PAUSE = 0xD3,
01951 // 0xD4 - 0xDF: reserved
01952 // Misc keys
01953 KEY_SELECT = 0xE0,
01954 KEY_PRINT = 0xE1,
01955 KEY_EXECUTE = 0xE2,
01956 KEY_HELP = 0xE3,
01957 KEY_APPS = 0xE4, // Applications key
01958 // 0xE5 - 0xFE: reserved
01959 // 0xFF: reserved
01960 };

```



```

01961
01963 typedef uint8_t AppCursor;
01964
01966 enum {
01968     APP_CURSOR_BLANK = 0x00,
01969
01971     APP_CURSOR_ARROW = 0x01,
01972
01974     APP_CURSOR_TEXT_SELECT,
01975
01976     // Vertical text selection cursor.
01977     APP_CURSOR_TEXT_SELECT_VERTICAL,
01978
01980     APP_CURSOR_WAIT,
01981
01983     APP_CURSOR_ARROW_WAIT,
01984
01986     APP_CURSOR_CROSSHAIR,
01987
01989     APP_CURSOR_CELL,
01990
01992     APP_CURSOR_LINK_SELECT,
01993
01995     APP_CURSOR_NOT_ALLOWED,
01996
01998     APP_CURSOR_HELP,
01999
02001     APP_CURSOR_MOVE,
02002
02004     APP_CURSOR_DRAG,
02005
02007     APP_CURSOR_DRAGGING,
02008
02010     APP_CURSOR_RESIZE_VERT,
02011
02013     APP_CURSOR_RESIZE_HORIZ,
02014
02016     APP_CURSOR_RESIZE1,
02017
02019     APP_CURSOR_RESIZE2,
02020
02022     APP_CURSOR_RESIZE_TOP,
02023
02025     APP_CURSOR_RESIZE_BOTTOM,
02026
02028     APP_CURSOR_RESIZE_LEFT,
02029
02031     APP_CURSOR_RESIZE_RIGHT,
02032
02034     APP_CURSOR_RESIZE_TOP_LEFT,
02035
02037     APP_CURSOR_RESIZE_TOP_RIGHT,
02038
02040     APP_CURSOR_RESIZE_BOTTOM_RIGHT,
02041
02043     APP_CURSOR_RESIZE_BOTTOM_LEFT,
02044
02046     APP_CURSOR_UNDEFINED = 255};
02047
02049 typedef uint8_t FileChooserDialog;
02050
02052 enum {
02054     FILE_CHOOSER_DIALOG_OPEN_FILE,
02055
02057     FILE_CHOOSER_DIALOG_SAVE_FILE,
02058
02060     FILE_CHOOSER_DIALOG_DIRECTORY};
02061
02063 typedef uint8_t WidgetAlignment;
02064
02066 enum {
02068     WIDGET_ALIGNMENT_NONE,
02069
02071     WIDGET_ALIGNMENT_CLIENT,
02072
02074     WIDGET_ALIGNMENT_LEFT,
02075
02077     WIDGET_ALIGNMENT_TOP,
02078
02080     WIDGET_ALIGNMENT_RIGHT,
02081
02083     WIDGET_ALIGNMENT_BOTTOM};
02084
02086 typedef uint8_t WidgetPropertyType;
02087
02089 enum {
02091     WIDGET_PROPERTY_TYPE_NONE,

```

```

02092
02094     WIDGET_PROPERTY_TYPE_INTEGER,
02095
02097     WIDGET_PROPERTY_TYPE_INT64,
02098
02100     WIDGET_PROPERTY_TYPE_FLOAT,
02101
02103     WIDGET_PROPERTY_TYPE_REAL,
02104
02106     WIDGET_PROPERTY_TYPE_BOOLEAN,
02107
02109     WIDGET_PROPERTY_TYPE_POINT,
02110
02112     WIDGET_PROPERTY_TYPE_VECTOR,
02113
02115     WIDGET_PROPERTY_TYPE_RECT,
02116
02118     WIDGET_PROPERTY_TYPE_MARGINS,
02119
02121     WIDGET_PROPERTY_TYPE_COLOR,
02122
02124     WIDGET_PROPERTY_TYPE_COLOR_PAIR,
02125
02127     WIDGET_PROPERTY_TYPE_COLOR_RECT,
02128
02130     WIDGET_PROPERTY_TYPE_ENUMERATION,
02131
02133     WIDGET_PROPERTY_TYPE_STRING,
02134
02136     WIDGET_PROPERTY_TYPE_FONT};
02137
02139 typedef uint8_t WidgetPropertyBehavior;
02140
02142 enum {
02144     WIDGET_PROPERTY_BEHAVIOR_NORMAL,
02145
02147     WIDGET_PROPERTY_BEHAVIOR_STATIC,
02148
02150     WIDGET_PROPERTY_BEHAVIOR_INDIRECT,
02151
02153     WIDGET_PROPERTY_BEHAVIOR_VOLATILE,
02154
02156     WIDGET_PROPERTY_BEHAVIOR_EVENT
02157 };
02158
02160 enum {
02162     WIDGET_PROPERTY_ATTRIBUTE_ASSIGNED = 0x1000
02163 };
02164
02166 typedef uint8_t WidgetManagerTextureType;
02167
02169 enum {
02171     WIDGET_MANAGER_TEXTURE_TYPE_CANVAS,
02172
02174     WIDGET_MANAGER_TEXTURE_TYPE_SCREEN,
02175
02177     WIDGET_MANAGER_TEXTURE_TYPE_AUXILIARY
02178 };
02179
02181 enum WidgetManagerAttribute
02182 {
02184     WIDGET_MANAGER_ATTRIBUTE_COMPOSITION = 0x01,
02185
02187     WIDGET_MANAGER_ATTRIBUTE_DESIGN = 0x80
02188 };
02189
02192 typedef uint32_t Color;
02193
02195 static Color const COLOR_BLACK = 0xFF000000;
02196
02198 static Color const COLOR_WHITE = 0xFFFFFFFF;
02199
02201 static Color const COLOR_TRANSLUCENT_BLACK = 0x00000000;
02202
02204 static Color const COLOR_TRANSLUCENT_WHITE = 0x00FFFFFF;
02205
02208 typedef struct ColorPair {
02210     Color first;
02211
02213     Color second;
02214
02215 #ifdef __cplusplus
02217     ColorPair(Color first, Color second) : first(first), second(second) {};
02218
02220     ColorPair(Color color) : first(color), second(color) {};
02221 #endif
02222 } ColorPair;

```

```

02223
02224 #ifndef __cplusplus
02226 #define COLOR_PAIR(x) ((ColorPair){x, x})
02227 #endif
02228
02232 typedef struct ColorRect
02233 {
02235     Color topLeft;
02236
02238     Color topRight;
02239
02241     Color bottomRight;
02242
02244     Color bottomLeft;
02245 } ColorRect;
02246
02247 #ifndef __cplusplus
02249 #define COLOR_RECT(x) ((ColorRect){x, x, x, x})
02250
02252 #define COLOR_RECT_H(x, y) ((ColorRect){x, y, y, x})
02253
02255 #define COLOR_RECT_V(x, y) ((ColorRect){x, x, y, y})
02256 #endif
02257
02261 typedef struct FloatColorRGB
02262 {
02264     float red;
02265
02267     float green;
02268
02270     float blue;
02271 } FloatColorRGB;
02272
02276 typedef struct FloatColor
02277 {
02279     float red;
02280
02282     float green;
02283
02285     float blue;
02286
02288     float alpha;
02289 } FloatColor;
02290
02292 typedef struct Point
02293 {
02295     int32_t x;
02296
02298     int32_t y;
02299 } Point;
02300
02302 typedef struct PointF
02303 {
02305     float x;
02306
02308     float y;
02309 } PointF;
02310
02312 typedef struct Vector
02313 {
02315     float x;
02316
02318     float y;
02319
02321     float z;
02322 } Vector;
02323
02325 typedef struct Vector4
02326 {
02328     float x;
02329
02331     float y;
02332
02334     float z;
02335
02337     float w;
02338 } Vector4;
02339
02341 typedef struct Matrix3x2
02342 {
02344     float data[3][2];
02345 } Matrix3x2;
02346
02348 typedef struct Matrix
02349 {
02351     float data[4][4];
02352 } Matrix;

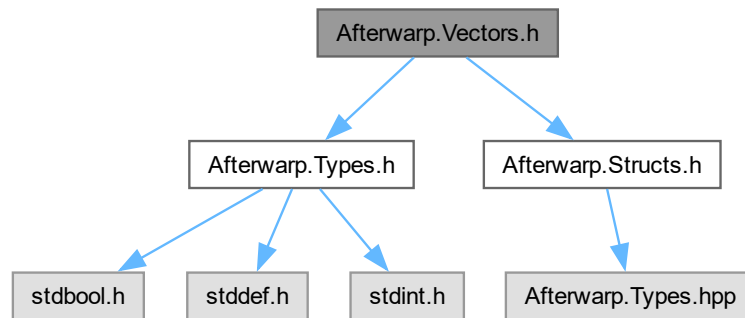
```

```
02353
02355 typedef struct Quaternion
02356 {
02358     float x;
02359
02361     float y;
02362
02364     float z;
02365
02367     float w;
02368 } Quaternion;
02369
02371 typedef struct Rect
02372 {
02374     int32_t left;
02375
02377     int32_t top;
02378
02380     int32_t width;
02381
02383     int32_t height;
02384 } Rect;
02385
02387 typedef struct RectF
02388 {
02390     float left;
02391
02393     float top;
02394
02396     float width;
02397
02399     float height;
02400 } RectF;
02401
02403 typedef struct Quad
02404 {
02406     PointF topLeft;
02407
02409     PointF topRight;
02410
02412     PointF bottomRight;
02413
02415     PointF bottomLeft;
02416 } Quad;
02417
02419 typedef struct Margins
02420 {
02422     float left;
02423
02425     float top;
02426
02428     float right;
02429
02431     float bottom;
02432 } Margins;
```

7.7 Afterwarp.Vectors.h File Reference

```
#include "Afterwarp.Types.h"
#include "Afterwarp.Structs.h"
```

Include dependency graph for Afterwarp.Vectors.h:



Macros

- `#define AFTERWARP_VECTORS_INTERFACE`
- `#define AFTERWARP_VECTORS_H`
- `#define MAX(x, y) (((x) > (y)) ? (x) : (y))`
- `#define MIN(x, y) (((x) < (y)) ? (x) : (y))`

Functions

- `ColorPair makeColorPair (Color first, Color second)`
Creates color pair with the given parameters.
- `ColorPair colorPair (Color color)`
Creates color pair with both colors equal to the given one.
- `FloatColorRGB makeFloatColorRGB (float red, float green, float blue)`
Creates floating-point color from the given parameters.
- `FloatColorRGB floatColorRGB (Color color)`
Creates floating-point color from 32-bit RGBA color value.
- `FloatColorRGB floatColorFromValueRGB (float value)`
Creates floating-point color with all components equal to the given value.
- `FloatColorRGB floatColorNegateRGB (FloatColorRGB color)`
Returns color with all components negated.
- `FloatColorRGB floatColorInverseRGB (FloatColorRGB color)`
Returns color with all components inverted (one minus value).
- `FloatColorRGB floatColorSaturateRGB (FloatColorRGB color)`
Takes floating-point color and clamps each component within [0, 1] range.
- `FloatColorRGB floatColorAddRGB (FloatColorRGB color1, FloatColorRGB color2)`
Add components of the given two colors.
- `FloatColorRGB floatColorSubtractRGB (FloatColorRGB color1, FloatColorRGB color2)`
Subtracts components of the second color from the first one.
- `FloatColorRGB floatColorMultiplyRGB (FloatColorRGB color1, FloatColorRGB color2)`
Multiplies components of the given two colors.
- `FloatColorRGB floatColorDivideRGB (FloatColorRGB color1, FloatColorRGB color2)`

- Divides components of the first color by the components of the second one.*

 - `FloatColorRGB floatColorAverageRGB` (`FloatColorRGB` color1, `FloatColorRGB` color2)

Computes average between the two given floating-point color values.
- `FloatColorRGB floatColorLerpRGB` (`FloatColorRGB` color1, `FloatColorRGB` color2, float theta)

Interpolates between current and the specified color; theta should be in range of [0, 1].
- `bool floatColorEqualsRGB` (`FloatColorRGB` color1, `FloatColorRGB` color2)

Tests whether the components of two given colors are indistinguishable.
- `bool floatColorEmptyRGB` (`FloatColorRGB` color)

Tests whether the components of given color are nearly zero.
- `float floatColorGrayRGB` (`FloatColorRGB` color)
- `Color floatToColorRGB` (`FloatColorRGB` color)

Converts current floating-point color to 32-bit RGBA integer representation.
- `FloatColor makeFloatColor` (float red, float green, float blue, float alpha)

Creates floating-point color from the given parameters.
- `FloatColor floatColor` (`Color` color)

Creates floating-point color from 32-bit RGBA color value.
- `FloatColor floatColorFromValue` (float value)

Creates floating-point color with all components equal to the given value.
- `FloatColor floatColorNegate` (`FloatColor` color)

Returns color with all components negated.
- `FloatColor floatColorInverse` (`FloatColor` color)

Returns color with all components inverted (one minus value).
- `FloatColor floatColorPremultiply` (`FloatColor` color)
- `FloatColor floatColorUnpremultiply` (`FloatColor` color)
- `FloatColor floatColorSaturate` (`FloatColor` color)

Takes floating-point color and clamps each component within [0, 1] range.
- `FloatColor floatColorAdd` (`FloatColor` color1, `FloatColor` color2)

Add components of the given two colors.
- `FloatColor floatColorSubtract` (`FloatColor` color1, `FloatColor` color2)

Subtracts components of the second color from the first one.
- `FloatColor floatColorMultiply` (`FloatColor` color1, `FloatColor` color2)

Multiplies components of the given two colors.
- `FloatColor floatColorDivide` (`FloatColor` color1, `FloatColor` color2)

Divides components of the first color by the components of the second one.
- `FloatColor floatColorAverage` (`FloatColor` color1, `FloatColor` color2)

Computes average between the two given floating-point color values.
- `FloatColor floatColorLerp` (`FloatColor` color1, `FloatColor` color2, float theta)

Interpolates between current and the specified color; theta should be in range of [0, 1].
- `bool floatColorEquals` (`FloatColor` color1, `FloatColor` color2)

Tests whether the components of two given colors are indistinguishable.
- `bool floatColorEmpty` (`FloatColor` color)

Tests whether the components of given color are nearly zero.
- `float floatColorGray` (`FloatColor` color)
- `Color floatToColor` (`FloatColor` color)

Converts current floating-point color to 32-bit RGBA integer representation.
- `FloatColorRGB floatColorToRGB` (`FloatColor` color)

Returns three components from an existing 4-component floating-point color.
- `Point makePoint` (int32_t x, int32_t y)

Creates 2D integer point from the given parameters.
- `Point pointValue` (int32_t value)

Creates 2D integer point with both X and Y set to the given value.

- [Point pointNegate](#) ([Point](#) point)
Returns 2D integer point with both components negated.
- [Point pointAdd](#) ([Point](#) point1, [Point](#) point2)
Adds components of two 2D integer points and returns the result.
- [Point pointSubtract](#) ([Point](#) point1, [Point](#) point2)
Subtracts components of the second 2D integer point from the first one and returns the result.
- [Point pointMultiply](#) ([Point](#) point1, [Point](#) point2)
Multiplies components of two 2D integer points and returns the result.
- [Point pointDivide](#) ([Point](#) point1, [Point](#) point2)
Divides components of the first 2D integer point by the components of the second one.
- [Point pointRescale](#) ([Point](#) point, [int32_t](#) theta)
Multiplies components of 2D integer point by the given coefficient and returns the result.
- [float pointLength](#) ([Point](#) point)
Returns length of the given 2D integer point.
- [float pointDistance](#) ([Point](#) point1, [Point](#) point2)
Returns distance between two 2D integer points.
- [float pointAngle](#) ([Point](#) point)
Returns angle (in radians) at which the given 2D integer vector is pointing at, in range of $[-\pi, \pi]$.
- [int32_t pointDot](#) ([Point](#) point1, [Point](#) point2)
- [int32_t pointCross](#) ([Point](#) point1, [Point](#) point2)
Calculates a so-called "cross-product" between two given 2D integer vectors, or analog of thereof.
- [Point pointLerp](#) ([Point](#) point1, [Point](#) point2, [float](#) theta)
Interpolates between first and second specified 2D integer vectors; theta should be in range of $[0, 1]$.
- [bool pointEquals](#) ([Point](#) point1, [Point](#) point2)
Tests whether two given 2D integer points match.
- [bool pointEmpty](#) ([Point](#) point)
Tests whether the given 2D integer point has both components equal to zero.
- [PointF makePointF](#) ([float](#) x, [float](#) y)
Creates floating-point 2D point from the given parameters.
- [PointF pointF](#) ([Point](#) point)
Creates floating-point 2D point from integer point.
- [PointF pointValueF](#) ([float](#) value)
Creates 2D floating-point vector with both X and Y set to the given value.
- [PointF pointNegateF](#) ([PointF](#) point)
Returns 2D floating-point vector with both components negated.
- [PointF pointAddF](#) ([PointF](#) point1, [PointF](#) point2)
Adds components of two 2D floating-point vectors and returns the result.
- [PointF pointSubtractF](#) ([PointF](#) point1, [PointF](#) point2)
Subtracts components of the second 2D floating-point vector from the first one and returns the result.
- [PointF pointMultiplyF](#) ([PointF](#) point1, [PointF](#) point2)
Multiplies components of two 2D floating-point vectors and returns the result.
- [PointF pointDivideF](#) ([PointF](#) point1, [PointF](#) point2)
Divides components of the first 2D floating-point vector by the components of the second one.
- [PointF pointRescaleF](#) ([PointF](#) point, [float](#) theta)
Multiplies components of 2D floating-point vector by the given coefficient and returns the result.
- [float pointLengthF](#) ([PointF](#) point)
Returns length of the given 2D floating-point vector.
- [float pointDistanceF](#) ([PointF](#) point1, [PointF](#) point2)
Returns distance between two 2D floating-point vectors.
- [float pointAngleF](#) ([PointF](#) point)
- [float pointDotF](#) ([PointF](#) point1, [PointF](#) point2)

- float [pointCrossF](#) ([PointF](#) point1, [PointF](#) point2)
Calculates a so-called "cross-product" between two given 2D floating-point vectors, or analog of thereof.
- [PointF](#) [pointNormalizeF](#) ([PointF](#) point)
- [PointF](#) [pointLerpF](#) ([PointF](#) point1, [PointF](#) point2, float theta)
- bool [pointEqualsF](#) ([PointF](#) point1, [PointF](#) point2)
Tests whether two given 2D floating-point vectors are indistinguishable.
- bool [pointEmptyF](#) ([PointF](#) point)
Tests whether the given 2D floating-point vector has both components nearly zero.
- [PointF](#) [pointTransformF](#) ([PointF](#) point, [Matrix3x2](#) matrix)
- [Vector](#) [makeVector](#) (float x, float y, float z)
Creates 3D vector from the given parameters.
- [Vector](#) [vectorValue](#) (float value)
Creates 3D vector with X, Y and Z set to the given value.
- [Vector](#) [vectorNegate](#) ([Vector](#) vector)
Returns 3D vector with all components negated.
- [Vector](#) [vectorAdd](#) ([Vector](#) vector1, [Vector](#) vector2)
Adds components of two 3D vectors and returns the result.
- [Vector](#) [vectorSubtract](#) ([Vector](#) vector1, [Vector](#) vector2)
Subtracts components of the second 3D vector from the first one and returns the result.
- [Vector](#) [vectorMultiply](#) ([Vector](#) vector1, [Vector](#) vector2)
Multiplies components of two 3D vectors and returns the result.
- [Vector](#) [vectorDivide](#) ([Vector](#) vector1, [Vector](#) vector2)
Divides components of the first 3D vector by the components of the second one.
- [Vector](#) [vectorRescale](#) ([Vector](#) vector, float theta)
Multiplies components of 3D vector by the given coefficient and returns the result.
- float [vectorDot](#) ([Vector](#) vector1, [Vector](#) vector2)
- [Vector](#) [vectorCross](#) ([Vector](#) vector1, [Vector](#) vector2)
- float [vectorLength](#) ([Vector](#) vector)
Returns length of the given 3D vector.
- float [vectorDistance](#) ([Vector](#) vector1, [Vector](#) vector2)
Returns distance between two 3D vectors.
- float [vectorAngle](#) ([Vector](#) vector1, [Vector](#) vector2)
Calculates angle between two 3D vectors. The returned value has range of [0, Pi].
- [Vector](#) [vectorNormalize](#) ([Vector](#) vector)
Normalizes given 3D vector to unity length. If the vector has zero length, it will be returned unchanged.
- [Vector](#) [vectorParallel](#) ([Vector](#) vector, [Vector](#) direction)
Calculates portion of given 3D vector that is parallel to the direction vector.
- [Vector](#) [vectorPerpendicular](#) ([Vector](#) vector, [Vector](#) direction)
Calculates portion of given 3D vector that is perpendicular to the direction vector.
- [Vector](#) [vectorReflect](#) ([Vector](#) vector, [Vector](#) normal)
Calculates 3D vector that is a reflection of given vector from surface given by the specified normal.
- [Vector](#) [vectorLerp](#) ([Vector](#) vector1, [Vector](#) vector2, float theta)
Interpolates between first and second specified 3D vectors; theta should be in range of [0, 1].
- bool [vectorEquals](#) ([Vector](#) vector1, [Vector](#) vector2)
Tests whether two given 3D vectors are indistinguishable.
- bool [vectorEmpty](#) ([Vector](#) vector)
Tests whether the given 3D vector has all components nearly zero.
- [Vector](#) [vectorTransform](#) ([Vector](#) vector, float w, [Matrix](#) matrix)
- [Vector](#) [vectorProject](#) ([Vector](#) vector, float w, [Matrix](#) matrix)
- [PointF](#) [vectorProjectTarget](#) ([Vector](#) vector, [PointF](#) targetSize, [Matrix](#) matrix)
- [Vector4](#) [makeVector3D](#) (float x, float y, float z, float w)

- Creates 4-component vector from the given parameters.*
- [Vector4 vectorFromValue3D](#) (float value)
Creates 4-component vector with X, Y, Z and W set to the given value.
- [Vector4 vectorNegate3D](#) ([Vector4](#) vector)
Returns 4-component vector with all components negated.
- [Vector4 vectorAdd3D](#) ([Vector4](#) vector1, [Vector4](#) vector2)
Adds components of two 4-component vectors and returns the result.
- [Vector4 vectorSubtract3D](#) ([Vector4](#) vector1, [Vector4](#) vector2)
Subtracts components of the second 4-component vector from the first one and returns the result.
- [Vector4 vectorMultiply3D](#) ([Vector4](#) vector1, [Vector4](#) vector2)
Multiplies components of two 4-component vectors and returns the result.
- [Vector4 vectorDivide3D](#) ([Vector4](#) vector1, [Vector4](#) vector2)
Divides components of the first 4-component vector by the components of the second one.
- [Vector4 vectorRescale3D](#) ([Vector4](#) vector, float theta)
Multiplies components of 4-component vector by the given coefficient and returns the result.
- float [vectorDot3D](#) ([Vector4](#) vector1, [Vector4](#) vector2)
- float [vectorLength3D](#) ([Vector4](#) vector)
Returns length of the given 4-component vector.
- float [vectorDistance3D](#) ([Vector4](#) vector1, [Vector4](#) vector2)
Returns distance between two 4-component vectors.
- [Vector4 vectorNormalize3D](#) ([Vector4](#) vector)
- [Vector4 vectorLerp3D](#) ([Vector4](#) vector1, [Vector4](#) vector2, float theta)
Interpolates between first and second specified 4-component vectors; theta should be in range of [0, 1].
- bool [vectorEquals3D](#) ([Vector4](#) vector1, [Vector4](#) vector2)
Tests whether two given 4-component vectors are indistinguishable.
- bool [vectorEmpty3D](#) ([Vector4](#) vector)
Tests whether the given 4-component vector has all components nearly zero.
- [Vector4 vectorTransform3D](#) ([Vector4](#) vector, [Matrix](#) matrix)
Multiplies given 4-component vector by the given 4x4 matrix.
- [Matrix3x2 matrixMultiply3x2](#) ([Matrix3x2](#) matrix1, [Matrix3x2](#) matrix2)
Multiplies two 3x2 matrices and returns the result.
- [Matrix3x2 matrixRescale3x2](#) ([Matrix3x2](#) matrix, float theta)
Multiplies 3x2 matrix by a coefficient and returns the result.
- float [matrixDeterminant3x2](#) ([Matrix3x2](#) matrix)
Calculates determinant of current matrix (as if it was 3x2 with right column being identity).
- [Matrix3x2 matrixInverse3x2](#) ([Matrix3x2](#) matrix)
Calculates inverse matrix of the current matrix (as if it was 3x2 with right column being identity).
- [Matrix3x2 matrixInverseTranspose3x2](#) ([Matrix3x2](#) matrix)
Calculates inverse and transpose matrix of the current matrix as if it was 2x2.
- [Matrix3x2 matrixScale3x2](#) ([PointF](#) scale)
Creates 2D scale 3x2 matrix with the specified coefficients.
- [Matrix3x2 matrixScaleBy3x2](#) ([PointF](#) center, [PointF](#) scale)
Creates 2D scale 3x2 matrix around a specific center point.
- [Matrix3x2 matrixTranslate3x2](#) ([PointF](#) offset)
Creates 2D translation 3x2 matrix with the specified offset.
- [Matrix3x2 matrixRotate3x2](#) (float angle)
Creates 2D rotation 3x2 matrix with specified angle (in radians).
- [Matrix3x2 matrixRotateBy3x2](#) ([PointF](#) center, float angle)
Creates 2D rotation 3x2 matrix with specified angle (in radians) around a specific center point.
- bool [matrixEquals3x2](#) ([Matrix3x2](#) matrix1, [Matrix3x2](#) matrix2)
Tests whether both 3x2 matrices are indistinguishable.

- bool [matrixEmpty3x2](#) ([Matrix3x2](#) matrix)
Tests whether given 3x2 matrix has all components nearly zero.
- [Matrix3x2 matrixGet3x2](#) ([Matrix](#) matrix)
Retrieves 3x2 portion from the given matrix.
- [Matrix matrixMultiply](#) ([Matrix](#) matrix1, [Matrix](#) matrix2)
Multiplies two 4x4 matrices and returns the result.
- [Matrix matrixRescale](#) ([Matrix](#) matrix, float theta)
Multiplies 4x4 matrix by a coefficient and returns the result.
- float [matrixDeterminant](#) ([Matrix](#) matrix)
Calculates determinant of the given 4x4 matrix.
- [Matrix matrixTranspose](#) ([Matrix](#) matrix)
Calculates transposed matrix from the given 4x4 matrix.
- [Matrix matrixAdjoint](#) ([Matrix](#) matrix)
Calculates adjoint matrix from the given 4x4 matrix.
- [Matrix matrixInverse](#) ([Matrix](#) matrix)
Calculates inverse matrix from the given 4x4 matrix.
- [Matrix matrixSub3x3](#) ([Matrix](#) matrix)
Copies 3x3 portion of source matrix while leaving remaining portion equal to identity 4x4 matrix.
- [Vector matrixEyePosition](#) ([Matrix](#) matrix)
- [Vector matrixWorldPosition](#) ([Matrix](#) matrix)
- [Matrix matrixTranslate](#) ([Vector](#) offset)
Creates 3D translation 4x4 matrix with specified offset.
- [Matrix matrixScale](#) ([Vector](#) scale)
Creates 3D scale 4x4 matrix with specified coefficients.
- [Matrix matrixRotateX](#) (float angle)
Creates 3D rotation 4x4 matrix around X axis with the specified angle.
- [Matrix matrixRotateY](#) (float angle)
Creates 3D rotation 4x4 matrix around Y axis with the specified angle.
- [Matrix matrixRotateZ](#) (float angle)
Creates 3D rotation 4x4 matrix around Z axis with the specified angle.
- [Matrix matrixRotate](#) ([Vector](#) axis, float angle)
Creates 3D rotation 4x4 matrix around specified axis and angle (in radians).
- [Matrix matrixHeadingPitchBank](#) ([Vector](#) angles)
- [Matrix matrixYawPitchRoll](#) ([Vector](#) angles)
- [Matrix matrixReflect](#) ([Vector](#) axis)
Creates a reflection matrix specified by the given vector defining orientation of that reflection.
- [Matrix matrixLookAt](#) ([Vector](#) origin, [Vector](#) target, [Vector](#) ceiling)
- [Matrix matrixPerspectiveFOVY](#) (float fieldOfView, float aspectRatio, float minRange, float maxRange, bool negativeDepthRange)
- [Matrix matrixPerspectiveFOVX](#) (float fieldOfView, float aspectRatio, float minRange, float maxRange, bool negativeDepthRange)
- [Matrix matrixPerspectiveVOL](#) (float width, float height, float minRange, float maxRange)
Creates perspective projection matrix defined by the viewing volume in 3D space.
- [Matrix matrixPerspectiveBDS](#) (float left, float top, float right, float bottom, float minRange, float maxRange)
Creates perspective projection matrix defined by the individual axis's boundaries.
- [Matrix matrixOrthogonalVOL](#) (float width, float height, float minRange, float maxRange)
Creates orthogonal projection matrix defined by the viewing volume in 3D space.
- [Matrix matrixOrthogonalBDS](#) (float left, float top, float right, float bottom, float minRange, float maxRange)
Creates orthogonal projection matrix defined by the individual axis's boundaries.
- bool [matrixEquals](#) ([Matrix](#) matrix1, [Matrix](#) matrix2)
Tests whether both 4x4 matrices are indistinguishable.

- bool [matrixEmpty](#) ([Matrix](#) matrix)
Tests whether given 4x4 matrix has all components nearly zero.
- [Quaternion quaternionMultiply](#) ([Quaternion](#) quat1, [Quaternion](#) quat2)
Multiplies quaternion by another quaternion, effectively combining their transformations.
- float [quaternionDot](#) ([Quaternion](#) quat1, [Quaternion](#) quat2)
Computes dot product between two given quaternions.
- float [quaternionLength](#) ([Quaternion](#) quat)
Returns the magnitude of the given quaternion.
- float [quaternionAngle](#) ([Quaternion](#) quat)
Returns rotational angle that is present in the given quaternion.
- [Vector quaternionAxis](#) ([Quaternion](#) quat)
Returns rotational axis that is present in the given quaternion.
- [Quaternion quaternionNormalize](#) ([Quaternion](#) quat)
- [Quaternion quaternionConjugate](#) ([Quaternion](#) quat)
Computes conjugate of the given quaternion. The resulting quaternion has opposite rotation.
- [Quaternion quaternionExponentiate](#) ([Quaternion](#) quat, float exponent)
Computes exponentiation of the given quaternion.
- [Quaternion quaternionSlerp](#) ([Quaternion](#) quat1, [Quaternion](#) quat2, float theta)
Applies spherical linear interpolation between two given quaternions.
- [Matrix quaternionMatrix](#) ([Quaternion](#) quat)
Converts the given quaternion into 4x4 matrix representation.
- [Quaternion matrixQuaternion](#) ([Matrix](#) matrix)
Creates quaternion with transformation converted from the given matrix.
- [Quaternion quaternionRotateX](#) (float angle)
Creates 3D quaternion containing rotation around X axis with given angle (in radians).
- [Quaternion quaternionRotateY](#) (float angle)
Creates 3D quaternion containing rotation around Y axis with given angle (in radians).
- [Quaternion quaternionRotateZ](#) (float angle)
Creates 3D quaternion containing rotation around Z axis with given angle (in radians).
- [Quaternion quaternionRotate](#) ([Vector](#) axis, float angle)
Creates 3D quaternion containing rotation around an arbitrary axis with given angle (in radians).
- [Quaternion quaternionRotateObjectToInertial](#) ([Vector](#) angles)
- [Quaternion quaternionRotateInertialToObject](#) ([Vector](#) angles)
- [Rect makeRect](#) (int32_t left, int32_t top, int32_t width, int32_t height)
Creates rectangle with the given parameters.
- [Rect makeBounds](#) (int32_t left, int32_t top, int32_t right, int32_t bottom)
Creates rectangle from the given boundaries.
- [Point rectGetSize](#) ([Rect](#) rect)
Returns the size of given rectangle as 2D integer vector.
- [Rect rectSetSize](#) ([Rect](#) rect, [Point](#) size)
Specifies new size for given rectangle from values of 2D integer vector.
- int32_t [rectGetRight](#) ([Rect](#) rect)
Returns right edge of the given rectangle.
- [Rect rectSetRight](#) ([Rect](#) rect, int32_t right)
Specifies new right edge for the given rectangle.
- int32_t [rectGetBottom](#) ([Rect](#) rect)
Returns bottom edge of the given rectangle.
- [Rect rectSetBottom](#) ([Rect](#) rect, int32_t bottom)
Specifies new bottom edge for the given rectangle.
- [Point rectGetTopLeft](#) ([Rect](#) rect)
Returns top and left of the given rectangle as 2D integer vector.

- [Rect rectSetTopLeft](#) ([Rect](#) rect, [Point](#) topLeft)
Specifies Top and Left for the given rectangle from values of 2D integer vector.
- [Point rectGetBottomRight](#) ([Rect](#) rect)
Returns Bottom and Right of the given rectangle as 2D integer vector.
- [Rect rectSetBottomRight](#) ([Rect](#) rect, [Point](#) bottomRight)
Specifies Bottom and Right for the given rectangle from values of 2D integer vector.
- [Point rectGetTopRight](#) ([Rect](#) rect)
Returns top and right of the given rectangle as 2D integer vector.
- [Point rectGetBottomLeft](#) ([Rect](#) rect)
Returns bottom and left of the given rectangle as 2D integer vector.
- [bool rectEquals](#) ([Rect](#) rect1, [Rect](#) rect2)
Tests whether both given rectangles are exactly the same.
- [bool rectEmpty](#) ([Rect](#) rect)
Tests whether the given rectangle is empty, that is, having width or height of zero or less.
- [bool pointInRect](#) ([Point](#) point, [Rect](#) rect)
Tests whether the specified point is contained within the given rectangle.
- [bool rectInRect](#) ([Rect](#) rect, [Rect](#) largerRect)
Tests whether the specified rectangle is contained within the given rectangle.
- [bool rectOverlap](#) ([Rect](#) rect1, [Rect](#) rect2)
Tests whether two given rectangles overlap.
- [Rect rectIntersect](#) ([Rect](#) rect1, [Rect](#) rect2)
Calculates rectangle that results from intersection between two given rectangles.
- [Rect rectJoin](#) ([Rect](#) rect1, [Rect](#) rect2)
Calculates rectangle that results from union between two given rectangles.
- [Rect rectOffset](#) ([Rect](#) rect, [Point](#) delta)
Calculates displaced rectangle by certain offset.
- [Rect rectInflate](#) ([Rect](#) rect, [Point](#) delta)
Returns rectangle with left and top decremented, while right and bottom incremented by given offset.
- [RectF makeRectF](#) (float left, float top, float width, float height)
Creates rectangle with the given parameters.
- [RectF makeBoundsF](#) (float left, float top, float right, float bottom)
Creates rectangle from the given boundaries.
- [RectF rectF](#) ([Rect](#) rect)
Creates rectangle with floating-point coordinates from rectangle with integer coordinates.
- [Rect rectToIntRectF](#) ([RectF](#) rect)
Creates rectangle with integer coordinates from rectangle with floating-point coordinates.
- [PointF rectGetSizeF](#) ([RectF](#) rect)
Returns the size of given rectangle as 2D integer vector.
- [RectF rectSetSizeF](#) ([RectF](#) rect, [PointF](#) size)
Specifies new size for given rectangle from values of 2D integer vector.
- [float rectGetRightF](#) ([RectF](#) rect)
Returns right edge of the given rectangle.
- [RectF rectSetRightF](#) ([RectF](#) rect, float right)
Specifies new right edge for the given rectangle.
- [float rectGetBottomF](#) ([RectF](#) rect)
Returns bottom edge of the given rectangle.
- [RectF rectSetBottomF](#) ([RectF](#) rect, float bottom)
Specifies new bottom edge for the given rectangle.
- [PointF rectGetTopLeftF](#) ([RectF](#) rect)
Returns top and left of the given rectangle as 2D integer vector.
- [RectF rectSetTopLeftF](#) ([RectF](#) rect, [PointF](#) topLeft)

- Specifies Top and Left for the given rectangle from values of 2D integer vector.*

 - [PointF rectGetBottomRightF](#) ([RectF](#) rect)

Returns Bottom and Right of the given rectangle as 2D integer vector.
- [RectF rectSetBottomRightF](#) ([RectF](#) rect, [PointF](#) bottomRight)

Specifies Bottom and Right for the given rectangle from values of 2D integer vector.
- [PointF rectGetTopRightF](#) ([RectF](#) rect)

Returns top and right of the given rectangle as 2D integer vector.
- [PointF rectGetBottomLeftF](#) ([RectF](#) rect)

Returns bottom and left of the given rectangle as 2D integer vector.
- [bool rectEqualsF](#) ([RectF](#) rect1, [RectF](#) rect2)

Tests whether both given rectangles are nearly same..
- [bool rectEmptyF](#) ([RectF](#) rect)

Tests whether the given rectangle is empty, that is, having width or height of nearly zero or less.
- [bool pointInRectF](#) ([PointF](#) point, [RectF](#) rect)

Tests whether the specified point is contained within the given rectangle.
- [bool rectInRectF](#) ([RectF](#) rect, [RectF](#) largerRect)

Tests whether the specified rectangle is contained within the given rectangle.
- [bool rectOverlapF](#) ([RectF](#) rect1, [RectF](#) rect2)

Tests whether two given rectangles overlap.
- [RectF rectIntersectF](#) ([RectF](#) rect1, [RectF](#) rect2)

Calculates rectangle that results from intersection between two given rectangles.
- [RectF rectJoinF](#) ([RectF](#) rect1, [RectF](#) rect2)

Calculates rectangle that results from union between two given rectangles.
- [RectF rectOffsetF](#) ([RectF](#) rect, [PointF](#) delta)

Calculates displaced rectangle by certain offset.
- [RectF rectInflateF](#) ([RectF](#) rect, [PointF](#) delta)

Returns rectangle with left and top decremented, while right and bottom incremented by given offset.
- [Quad makeQuadWith](#) ([PointF](#) topLeft, [PointF](#) topRight, [PointF](#) bottomRight, [PointF](#) bottomLeft)

Creates quadrilateral with the given parameters.
- [Quad makeQuad](#) (float left, float top, float width, float height)

Creates quadrilateral as rectangle with the given parameters.
- [Quad makeQuadScaled](#) (float left, float top, float width, float height, float scale, bool centered)
- [Quad makeQuadRotatedAt](#) ([PointF](#) origin, [PointF](#) size, [PointF](#) center, float angle, float scale)
- [Quad makeQuadRotated](#) ([PointF](#) origin, [PointF](#) size, float angle, float scale)
- [Quad makeQuadRotatedTL](#) ([PointF](#) topLeft, [PointF](#) size, [PointF](#) center, float angle, float scale)
- [Quad makeQuadSkewedHoriz](#) ([PointF](#) origin, [PointF](#) size, float angle)
- [Quad makeQuadSkewedVert](#) ([PointF](#) origin, [PointF](#) size, float angle)
- [Quad quadFromRect](#) ([Rect](#) rect)

Creates quadrilateral from rectangle with integer coordinates.
- [Quad quadFromRectF](#) ([RectF](#) rect)

Creates quadrilateral from rectangle with floating-point coordinates.
- [Quad quadScale](#) ([Quad](#) quad, float scale, bool centered)
- [Quad quadMirror](#) ([Quad](#) quad)
- [Quad quadFlip](#) ([Quad](#) quad)
- [Quad quadTransform](#) ([Quad](#) quad, [Matrix3x2](#) matrix)

Transforms (multiplies) vertices of the given quadrilateral by the specified matrix.
- [Quad quadOffset](#) ([Quad](#) quad, [PointF](#) delta)

Displaces vertices of the given quadrilateral by the specified offset.

Variables

- float const [vectorEpsilon](#)
- [FontEffect](#) const [fontEffectDefault](#)
Default font effect parameters provided for calling convenience.
- [Color](#) const [colorBlack](#)
Predefined constant for opaque Black color.
- [Color](#) const [colorWhite](#)
Predefined constant for opaque White color.
- [Color](#) const [colorTranslucentBlack](#)
Predefined constant for translucent Black color.
- [Color](#) const [colorTranslucentWhite](#)
Predefined constant for translucent White color.
- [ColorPair](#) const [colorPairBlack](#)
Predefined constant for a pair of opaque Black colors.
- [ColorPair](#) const [colorPairWhite](#)
Predefined constant for a pair of opaque White colors.
- [ColorPair](#) const [colorPairTraslucentBlack](#)
Predefined constant for a pair of translucent Black colors.
- [ColorPair](#) const [colorPairTraslucentWhite](#)
Predefined constant for a pair of translucent White colors.
- [ColorRect](#) const [colorRectBlack](#)
Predefined constant for four opaque Black colors.
- [ColorRect](#) const [colorRectWhite](#)
Predefined constant for four opaque White colors.
- [ColorRect](#) const [colorRectTraslucentBlack](#)
Predefined constant for four translucent Black colors.
- [ColorRect](#) const [colorRectTraslucentWhite](#)
Predefined constant for four translucent White colors.
- [FloatColorRGB](#) const [floatColorBlackRGB](#)
Predefined constant for opaque Black color.
- [FloatColorRGB](#) const [floatColorWhiteRGB](#)
Predefined constant for opaque White color.
- [FloatColor](#) const [floatColorBlack](#)
Predefined constant for opaque Black color.
- [FloatColor](#) const [floatColorWhite](#)
Predefined constant for opaque White color.
- [FloatColor](#) const [floatColorTraslucentBlack](#)
Predefined constant for translucent Black color.
- [FloatColor](#) const [floatColorTraslucentWhite](#)
Predefined constant for translucent White color.
- [Point](#) const [pointZero](#)
Predefined constant, where X and Y are zero.
- [Point](#) const [pointUnity](#)
Predefined constant, where X and Y are one.
- [Point](#) const [pointAxisX](#)
Predefined constant, where X = 1 and Y = 0.
- [Point](#) const [pointAxisY](#)
Predefined constant, where X = 0 and Y = 1.
- [PointF](#) const [pointZeroF](#)
Predefined constant, where X and Y are zero.

- [PointF](#) const [pointUnityF](#)
Predefined constant, where X and Y are one.
- [PointF](#) const [pointAxisXF](#)
Predefined constant, where X = 1 and Y = 0.
- [PointF](#) const [pointAxisYF](#)
Predefined constant, where X = 0 and Y = 1.
- [Vector](#) const [vectorZero](#)
Predefined constant, where X, Y and Z are zero.
- [Vector](#) const [vectorUnity](#)
Predefined constant, where X, Y and Z are one.
- [Vector](#) const [vectorAxisX](#)
Predefined constant, where X = 1, Y = 0 and Z = 0.
- [Vector](#) const [vectorAxisY](#)
Predefined constant, where X = 0, Y = 1 and Z = 0.
- [Vector](#) const [vectorAxisZ](#)
Predefined constant, where X = 0, Y = 0 and Z = 1.
- [Vector4](#) const [vectorZero3D](#)
Predefined constant, where X, Y, Z and W are zero.
- [Vector4](#) const [vectorUnity3D](#)
Predefined constant, where X, Y, Z and W are one.
- [Vector4](#) const [vectorAxisX3D](#)
Predefined constant, where X = 1, Y = 0, Z = 0 and W = 0.
- [Vector4](#) const [vectorAxisY3D](#)
Predefined constant, where X = 0, Y = 1, Z = 0 and W = 0.
- [Vector4](#) const [vectorAxisZ3D](#)
Predefined constant, where X = 0, Y = 0, Z = 1 and W = 0.
- [Vector4](#) const [vectorAxisW3D](#)
Predefined constant, where X = 0, Y = 0, Z = 0 and W = 1.
- [Matrix3x2](#) const [matrixZero3x2](#)
Predefined constant, where all matrix values are zero.
- [Matrix3x2](#) const [matrixIdentity3x2](#)
Predefined constant with values corresponding to an identity matrix.
- [Matrix](#) const [matrixZero](#)
Predefined constant, where all matrix values are zero.
- [Matrix](#) const [matrixIdentity](#)
Predefined constant with values corresponding to an identity matrix.
- [Quaternion](#) const [quaternionIdentity](#)
Predefined constant with values corresponding to an identity quaternion.
- [Rect](#) const [rectZero](#)
Empty rectangle with all members set to zero.
- [RectF](#) const [rectZeroF](#)
Empty rectangle with all members set to zero.
- [Quad](#) const [quadZero](#)
Quadrilateral with all vertices set to zero.
- [Quad](#) const [quadUnity](#)
Quadrilateral with vertices set at four corners set between (0, 0) and (1, 1).

7.7.1 Macro Definition Documentation

7.7.1.1 AFTERWARP_VECTORS_H

```
#define AFTERWARP_VECTORS_H
```

7.7.1.2 AFTERWARP_VECTORS_INTERFACE

```
#define AFTERWARP_VECTORS_INTERFACE
```

7.7.1.3 MAX

```
#define MAX(  
    x,  
    y ) ((x) > (y)) ? (x) : (y)
```

7.7.1.4 MIN

```
#define MIN(  
    x,  
    y ) ((x) < (y)) ? (x) : (y)
```

7.7.2 Function Documentation

7.7.2.1 colorPair()

```
ColorPair colorPair (  
    Color color ) [extern]
```

Creates color pair with both colors equal to the given one.

7.7.2.2 floatColor()

```
FloatColor floatColor (  
    Color color ) [extern]
```

Creates floating-point color from 32-bit RGBA color value.

7.7.2.3 floatColorAdd()

```
FloatColor floatColorAdd (  
    FloatColor color1,  
    FloatColor color2 ) [extern]
```

Add components of the given two colors.

7.7.2.4 floatColorAddRGB()

```
FloatColorRGB floatColorAddRGB (  
    FloatColorRGB color1,  
    FloatColorRGB color2 ) [extern]
```

Add components of the given two colors.

7.7.2.5 floatColorAverage()

```
FloatColor floatColorAverage (
    FloatColor color1,
    FloatColor color2 ) [extern]
```

Computes average between the two given floating-point color values.

7.7.2.6 floatColorAverageRGB()

```
FloatColorRGB floatColorAverageRGB (
    FloatColorRGB color1,
    FloatColorRGB color2 ) [extern]
```

Computes average between the two given floating-point color values.

7.7.2.7 floatColorDivide()

```
FloatColor floatColorDivide (
    FloatColor color1,
    FloatColor color2 ) [extern]
```

Divides components of the first color by the components of the second one.

7.7.2.8 floatColorDivideRGB()

```
FloatColorRGB floatColorDivideRGB (
    FloatColorRGB color1,
    FloatColorRGB color2 ) [extern]
```

Divides components of the first color by the components of the second one.

7.7.2.9 floatColorEmpty()

```
bool floatColorEmpty (
    FloatColor color ) [extern]
```

Tests whether the components of given color are nearly zero.

7.7.2.10 floatColorEmptyRGB()

```
bool floatColorEmptyRGB (
    FloatColorRGB color ) [extern]
```

Tests whether the components of given color are nearly zero.

7.7.2.11 floatColorEquals()

```
bool floatColorEquals (
    FloatColor color1,
    FloatColor color2 ) [extern]
```

Tests whether the components of two given colors are indistinguishable.

7.7.2.12 floatColorEqualsRGB()

```
bool floatColorEqualsRGB (
    FloatColorRGB color1,
    FloatColorRGB color2 ) [extern]
```

Tests whether the components of two given colors are indistinguishable.

7.7.2.13 floatColorFromValue()

```
FloatColor floatColorFromValue (
    float value ) [extern]
```

Creates floating-point color with all components equal to the given value.

7.7.2.14 floatColorFromValueRGB()

```
FloatColorRGB floatColorFromValueRGB (
    float value ) [extern]
```

Creates floating-point color with all components equal to the given value.

7.7.2.15 floatColorGray()

```
float floatColorGray (
    FloatColor color ) [extern]
```

Returns grayscale value in range of [0, 1] from the given floating-point color value. The resulting value can be considered the color's Luma. Alpha-channel is ignored.

7.7.2.16 floatColorGrayRGB()

```
float floatColorGrayRGB (
    FloatColorRGB color ) [extern]
```

Returns grayscale value in range of [0, 1] from the given floating-point color value. The resulting value can be considered the color's Luma. Alpha-channel is ignored.

7.7.2.17 floatColorInverse()

```
FloatColor floatColorInverse (
    FloatColor color ) [extern]
```

Returns color with all components inverted (one minus value).

7.7.2.18 floatColorInverseRGB()

```
FloatColorRGB floatColorInverseRGB (
    FloatColorRGB color ) [extern]
```

Returns color with all components inverted (one minus value).

7.7.2.19 floatColorLerp()

```
FloatColor floatColorLerp (
    FloatColor color1,
    FloatColor color2,
    float theta ) [extern]
```

Interpolates between current and the specified color; theta should be in range of [0, 1].

7.7.2.20 floatColorLerpRGB()

```
FloatColorRGB floatColorLerpRGB (
    FloatColorRGB color1,
    FloatColorRGB color2,
    float theta ) [extern]
```

Interpolates between current and the specified color; theta should be in range of [0, 1].

7.7.2.21 floatColorMultiply()

```
FloatColor floatColorMultiply (
    FloatColor color1,
    FloatColor color2 ) [extern]
```

Multiplies components of the given two colors.

7.7.2.22 floatColorMultiplyRGB()

```
FloatColorRGB floatColorMultiplyRGB (
    FloatColorRGB color1,
    FloatColorRGB color2 ) [extern]
```

Multiplies components of the given two colors.

7.7.2.23 floatColorNegate()

```
FloatColor floatColorNegate (
    FloatColor color ) [extern]
```

Returns color with all components negated.

7.7.2.24 floatColorNegateRGB()

```
FloatColorRGB floatColorNegateRGB (
    FloatColorRGB color ) [extern]
```

Returns color with all components negated.

7.7.2.25 floatColorPremultiply()

```
FloatColor floatColorPremultiply (
    FloatColor color ) [extern]
```

Takes floating-point color with unpremultiplied alpha and multiplies each of red, green, and blue components by its alpha-channel, resulting in premultiplied alpha color.

7.7.2.26 floatColorRGB()

```
FloatColorRGB floatColorRGB (
    Color color ) [extern]
```

Creates floating-point color from 32-bit RGBA color value.

7.7.2.27 floatColorSaturate()

```
FloatColor floatColorSaturate (
    FloatColor color ) [extern]
```

Takes floating-point color and clamps each component within [0, 1] range.

7.7.2.28 floatColorSaturateRGB()

```
FloatColorRGB floatColorSaturateRGB (
    FloatColorRGB color ) [extern]
```

Takes floating-point color and clamps each component within [0, 1] range.

7.7.2.29 floatColorSubtract()

```
FloatColor floatColorSubtract (
    FloatColor color1,
    FloatColor color2 ) [extern]
```

Subtracts components of the second color from the first one.

7.7.2.30 floatColorSubtractRGB()

```
FloatColorRGB floatColorSubtractRGB (
    FloatColorRGB color1,
    FloatColorRGB color2 ) [extern]
```

Subtracts components of the second color from the first one.

7.7.2.31 floatColorToRGB()

```
FloatColorRGB floatColorToRGB (
    FloatColor color ) [extern]
```

Returns three components from an existing 4-component floating-point color.

7.7.2.32 floatColorUnpremultiply()

```
FloatColor floatColorUnpremultiply (
    FloatColor color ) [extern]
```

Takes floating-point color with premultiplied alpha-channel and divides each of its red, green, and blue components by alpha, resulting in unpremultiplied alpha color.

7.7.2.33 floatToColor()

```
Color floatToColor (
    FloatColor color ) [extern]
```

Converts current floating-point color to 32-bit RGBA integer representation.

7.7.2.34 floatToColorRGB()

```
Color floatToColorRGB (
    FloatColorRGB color ) [extern]
```

Converts current floating-point color to 32-bit RGBA integer representation.

7.7.2.35 makeBounds()

```
Rect makeBounds (
    int32_t left,
    int32_t top,
    int32_t right,
    int32_t bottom )
```

Creates rectangle from the given boundaries.

7.7.2.36 makeBoundsF()

```
RectF makeBoundsF (
    float left,
    float top,
    float right,
    float bottom )
```

Creates rectangle from the given boundaries.

7.7.2.37 makeColorPair()

```
ColorPair makeColorPair (
    Color first,
    Color second ) [extern]
```

Creates color pair with the given parameters.

7.7.2.38 makeFloatColor()

```
FloatColor makeFloatColor (
    float red,
    float green,
    float blue,
    float alpha ) [extern]
```

Creates floating-point color from the given parameters.

7.7.2.39 makeFloatColorRGB()

```
FloatColorRGB makeFloatColorRGB (
    float red,
    float green,
    float blue ) [extern]
```

Creates floating-point color from the given parameters.

7.7.2.40 makePoint()

```
Point makePoint (
    int32_t x,
    int32_t y ) [extern]
```

Creates 2D integer point from the given parameters.

7.7.2.41 makePointF()

```
PointF makePointF (
    float x,
    float y ) [extern]
```

Creates floating-point 2D point from the given parameters.

7.7.2.42 makeQuad()

```
Quad makeQuad (
    float left,
    float top,
    float width,
    float height )
```

Creates quadrilateral as rectangle with the given parameters.

7.7.2.43 makeQuadRotated()

```
Quad makeQuadRotated (
    PointF origin,
    PointF size,
    float angle,
    float scale )
```

Creates quadrilateral specified by its dimensions. The rectangle is then rotated and scaled around its center and placed at the specified origin.

7.7.2.44 makeQuadRotatedAt()

```
Quad makeQuadRotatedAt (
    PointF origin,
    PointF size,
    PointF center,
    float angle,
    float scale )
```

Creates quadrilateral specified by its dimensions. The rectangle is then rotated and scaled around the specified middle point (assumed to be inside rectangle's dimensions) and placed in center of the specified origin.

7.7.2.45 makeQuadRotatedTL()

```
Quad makeQuadRotatedTL (
    PointF topLeft,
    PointF size,
    PointF center,
    float angle,
    float scale )
```

Creates quadrilateral specified by top-left corner and size. The rectangle is then rotated and scaled around the specified middle point (assumed to be inside rectangle's dimensions) and placed in the center of the specified origin. The difference between this method and makeQuadRotatedAt() is that the rotation does not preserve centering of the rectangle in case where middle point is not actually located in the middle.

7.7.2.46 makeQuadScaled()

```
Quad makeQuadScaled (
    float left,
    float top,
    float width,
    float height,
    float scale,
    bool centered )
```

Creates quadrilateral with the specified top left corner and the given dimensions, which are scaled by the provided coefficient.

7.7.2.47 makeQuadSkewedHoriz()

```
Quad makeQuadSkewedHoriz (
    PointF origin,
    PointF size,
    float angle )
```

Creates a horizontally skewed quadrilateral, oriented around its origin and with the given dimensions. A skew angle defines quadrilateral's top/left corner.

7.7.2.48 makeQuadSkewedVert()

```
Quad makeQuadSkewedVert (
    PointF origin,
    PointF size,
    float angle )
```

Creates a vertically skewed quadrilateral, oriented around its origin and with the given dimensions. A skew angle defines quadrilateral's top/left corner.

7.7.2.49 makeQuadWith()

```
Quad makeQuadWith (
    PointF topLeft,
    PointF topRight,
    PointF bottomRight,
    PointF bottomLeft )
```

Creates quadrilateral with the given parameters.

7.7.2.50 makeRect()

```
Rect makeRect (
    int32_t left,
    int32_t top,
    int32_t width,
    int32_t height )
```

Creates rectangle with the given parameters.

7.7.2.51 makeRectF()

```
RectF makeRectF (
    float left,
    float top,
    float width,
    float height )
```

Creates rectangle with the given parameters.

7.7.2.52 makeVector()

```
Vector makeVector (
    float x,
    float y,
    float z )
```

Creates 3D vector from the given parameters.

7.7.2.53 makeVector3D()

```
Vector4 makeVector3D (
    float x,
    float y,
    float z,
    float w )
```

Creates 4-component vector from the given parameters.

7.7.2.54 matrixAdjoint()

```
Matrix matrixAdjoint (
    Matrix matrix )
```

Calculates adjoint matrix from the given 4x4 matrix.

7.7.2.55 matrixDeterminant()

```
float matrixDeterminant (
    Matrix matrix )
```

Calculates determinant of the given 4x4 matrix.

7.7.2.56 matrixDeterminant3x2()

```
float matrixDeterminant3x2 (
    Matrix3x2 matrix )
```

Calculates determinant of current matrix (as if it was 3x2 with right column being identity).

7.7.2.57 matrixEmpty()

```
bool matrixEmpty (
    Matrix matrix )
```

Tests whether given 4x4 matrix has all components nearly zero.

7.7.2.58 matrixEmpty3x2()

```
bool matrixEmpty3x2 (
    Matrix3x2 matrix )
```

Tests whether given 3x2 matrix has all components nearly zero.

7.7.2.59 matrixEquals()

```
bool matrixEquals (
    Matrix matrix1,
    Matrix matrix2 )
```

Tests whether both 4x4 matrices are indistinguishable.

7.7.2.60 matrixEquals3x2()

```
bool matrixEquals3x2 (
    Matrix3x2 matrix1,
    Matrix3x2 matrix2 )
```

Tests whether both 3x2 matrices are indistinguishable.

7.7.2.61 matrixEyePosition()

```
Vector matrixEyePosition (
    Matrix matrix )
```

Assuming that the given 4x4 matrix is a view matrix, calculates the 3D position where the camera (or "eye") is supposedly located.

7.7.2.62 matrixGet3x2()

```
Matrix3x2 matrixGet3x2 (
    Matrix matrix )
```

Retrieves 3x2 portion from the given matrix.

7.7.2.63 matrixHeadingPitchBank()

```
Matrix matrixHeadingPitchBank (
    Vector angles )
```

Creates 3D rotation 4x4 matrix based on parameters similar to flight dynamics, specifically Heading, Pitch and Bank. The components are taken from the specified vector with Y corresponding to Heading, X to Pitch and Z to Bank.

7.7.2.64 matrixInverse()

```
Matrix matrixInverse (
    Matrix matrix )
```

Calculates inverse matrix from the given 4x4 matrix.

7.7.2.65 matrixInverse3x2()

```
Matrix3x2 matrixInverse3x2 (
    Matrix3x2 matrix )
```

Calculates inverse matrix of the current matrix (as if it was 3x2 with right column being identity).

7.7.2.66 matrixInverseTranspose3x2()

```
Matrix3x2 matrixInverseTranspose3x2 (
    Matrix3x2 matrix )
```

Calculates inverse and transpose matrix of the current matrix as if it was 2x2.

7.7.2.67 matrixLookAt()

```
Matrix matrixLookAt (
    Vector origin,
    Vector target,
    Vector ceiling )
```

Creates a view matrix that is defined by the camera's position, its target and vertical axis or "ceiling".

7.7.2.68 matrixMultiply()

```
Matrix matrixMultiply (
    Matrix matrix1,
    Matrix matrix2 )
```

Multiplies two 4x4 matrices and returns the result.

7.7.2.69 matrixMultiply3x2()

```
Matrix3x2 matrixMultiply3x2 (
    Matrix3x2 matrix1,
    Matrix3x2 matrix2 )
```

Multiplies two 3x2 matrices and returns the result.

7.7.2.70 matrixOrthogonalBDS()

```
Matrix matrixOrthogonalBDS (
    float left,
    float top,
    float right,
    float bottom,
    float minRange,
    float maxRange )
```

Creates orthogonal projection matrix defined by the individual axis's boundaries.

7.7.2.71 matrixOrthogonalVOL()

```
Matrix matrixOrthogonalVOL (
    float width,
    float height,
    float minRange,
    float maxRange )
```

Creates orthogonal projection matrix defined by the viewing volume in 3D space.

7.7.2.72 matrixPerspectiveBDS()

```
Matrix matrixPerspectiveBDS (
    float left,
    float top,
    float right,
    float bottom,
    float minRange,
    float maxRange )
```

Creates perspective projection matrix defined by the individual axis's boundaries.

7.7.2.73 matrixPerspectiveFOVX()

```
Matrix matrixPerspectiveFOVX (
    float fieldOfView,
    float aspectRatio,
    float minRange,
    float maxRange,
    bool negativeDepthRange )
```

Creates perspective projection matrix defined by a field of view on X axis. In 3D shooters the field of view needs to be adjusted to allow more visible area on wide-screen monitors. The parameters that define the viewed range are important for defining the precision of the depth transformation or a depth-buffer. FieldOfView - The camera's field of view in radians. For example $\text{Pi}/4$. AspectRatio - The screen's aspect ratio. Can be calculated as x/y . MinRange - The closest range at which the scene will be viewed. MaxRange - The farthest range at which the scene will be viewed.

7.7.2.74 matrixPerspectiveFOVY()

```
Matrix matrixPerspectiveFOVY (
    float fieldOfView,
    float aspectRatio,
    float minRange,
    float maxRange,
    bool negativeDepthRange )
```

Creates perspective projection matrix defined by a field of view on Y axis. This is a common way for typical 3D applications. In 3D shooters special care is to be taken because on wide-screen monitors the visible area will be bigger when using this method. The parameters that define the viewed range are important for defining the precision of the depth transformation or a depth-buffer. FieldOfView - The camera's field of view in radians. For example $\pi/4$. AspectRatio - The screen's aspect ratio. Can be calculated as y/x . MinRange - The closest range at which the scene will be viewed. MaxRange - The farthest range at which the scene will be viewed.

7.7.2.75 matrixPerspectiveVOL()

```
Matrix matrixPerspectiveVOL (
    float width,
    float height,
    float minRange,
    float maxRange )
```

Creates perspective projection matrix defined by the viewing volume in 3D space.

7.7.2.76 matrixQuaternion()

```
Quaternion matrixQuaternion (
    Matrix matrix )
```

Creates quaternion with transformation converted from the given matrix.

7.7.2.77 matrixReflect()

```
Matrix matrixReflect (
    Vector axis )
```

Creates a reflection matrix specified by the given vector defining orientation of that reflection.

7.7.2.78 matrixRescale()

```
Matrix matrixRescale (
    Matrix matrix,
    float theta )
```

Multiplies 4x4 matrix by a coefficient and returns the result.

7.7.2.79 matrixRescale3x2()

```
Matrix3x2 matrixRescale3x2 (
    Matrix3x2 matrix,
    float theta )
```

Multiplies 3x2 matrix by a coefficient and returns the result.

7.7.2.80 matrixRotate()

```
Matrix matrixRotate (
    Vector axis,
    float angle )
```

Creates 3D rotation 4x4 matrix around specified axis and angle (in radians).

7.7.2.81 matrixRotate3x2()

```
Matrix3x2 matrixRotate3x2 (
    float angle )
```

Creates 2D rotation 3x2 matrix with specified angle (in radians).

7.7.2.82 matrixRotateBy3x2()

```
Matrix3x2 matrixRotateBy3x2 (
    PointF center,
    float angle )
```

Creates 2D rotation 3x2 matrix with specified angle (in radians) around a specific center point.

7.7.2.83 matrixRotateX()

```
Matrix matrixRotateX (
    float angle )
```

Creates 3D rotation 4x4 matrix around X axis with the specified angle.

7.7.2.84 matrixRotateY()

```
Matrix matrixRotateY (
    float angle )
```

Creates 3D rotation 4x4 matrix around Y axis with the specified angle.

7.7.2.85 matrixRotateZ()

```
Matrix matrixRotateZ (
    float angle )
```

Creates 3D rotation 4x4 matrix around Z axis with the specified angle.

7.7.2.86 matrixScale()

```
Matrix matrixScale (
    Vector scale )
```

Creates 3D scale 4x4 matrix with specified coefficients.

7.7.2.87 matrixScale3x2()

```
Matrix3x2 matrixScale3x2 (
    PointF scale )
```

Creates 2D scale 3x2 matrix with the specified coefficients.

7.7.2.88 matrixScaleBy3x2()

```
Matrix3x2 matrixScaleBy3x2 (
    PointF center,
    PointF scale )
```

Creates 2D scale 3x2 matrix around a specific center point.

7.7.2.89 matrixSub3x3()

```
Matrix matrixSub3x3 (
    Matrix matrix )
```

Copies 3x3 portion of source matrix while leaving remaining portion equal to identity 4x4 matrix.

7.7.2.90 matrixTranslate()

```
Matrix matrixTranslate (
    Vector offset )
```

Creates 3D translation 4x4 matrix with specified offset.

7.7.2.91 matrixTranslate3x2()

```
Matrix3x2 matrixTranslate3x2 (
    PointF offset )
```

Creates 2D translation 3x2 matrix with the specified offset.

7.7.2.92 matrixTranspose()

```
Matrix matrixTranspose (  
    Matrix matrix )
```

Calculates transposed matrix from the given 4x4 matrix.

7.7.2.93 matrixWorldPosition()

```
Vector matrixWorldPosition (  
    Matrix matrix )
```

Assuming that the given 4x4 matrix is a world matrix, calculates the 3D position where the object (or "world") is supposedly located.

7.7.2.94 matrixYawPitchRoll()

```
Matrix matrixYawPitchRoll (  
    Vector angles )
```

Creates 3D rotation 4x4 matrix based on parameters similar to flight dynamics, specifically Yaw, Pitch and Roll. The components are taken from the specified vector with Y corresponding to Yaw, X to Pitch and Z to Roll.

7.7.2.95 pointAdd()

```
Point pointAdd (  
    Point point1,  
    Point point2 ) [extern]
```

Adds components of two 2D integer points and returns the result.

7.7.2.96 pointAddF()

```
PointF pointAddF (  
    PointF point1,  
    PointF point2 ) [extern]
```

Adds components of two 2D floating-point vectors and returns the result.

7.7.2.97 pointAngle()

```
float pointAngle (  
    Point point ) [extern]
```

Returns angle (in radians) at which the given 2D integer vector is pointing at, in range of [-Pi, Pi].

7.7.2.98 pointAngleF()

```
float pointAngleF (
    PointF point )
```

Returns angle (in radians) at which the given 2D floating-point vector is pointing at, in range of $[-\pi, \pi]$.

7.7.2.99 pointCross()

```
int32_t pointCross (
    Point point1,
    Point point2 ) [extern]
```

Calculates a so-called "cross-product" between two given 2D integer vectors, or analog of thereof.

7.7.2.100 pointCrossF()

```
float pointCrossF (
    PointF point1,
    PointF point2 )
```

Calculates a so-called "cross-product" between two given 2D floating-point vectors, or analog of thereof.

7.7.2.101 pointDistance()

```
float pointDistance (
    Point point1,
    Point point2 ) [extern]
```

Returns distance between two 2D integer points.

7.7.2.102 pointDistanceF()

```
float pointDistanceF (
    PointF point1,
    PointF point2 ) [extern]
```

Returns distance between two 2D floating-point vectors.

7.7.2.103 pointDivide()

```
Point pointDivide (
    Point point1,
    Point point2 ) [extern]
```

Divides components of the first 2D integer point by the components of the second one.

7.7.2.104 pointDivideF()

```
PointF pointDivideF (
    PointF point1,
    PointF point2 ) [extern]
```

Divides components of the first 2D floating-point vector by the components of the second one.

7.7.2.105 pointDot()

```
int32_t pointDot (
    Point point1,
    Point point2 ) [extern]
```

Calculates dot-product between two given 2D integer vectors. The dot-product is an indirect measure of the angle between two vectors.

7.7.2.106 pointDotF()

```
float pointDotF (
    PointF point1,
    PointF point2 )
```

Calculates dot-product between two given 2D floating-point vectors. The dot-product is an indirect measure of the angle between two vectors.

7.7.2.107 pointEmpty()

```
bool pointEmpty (
    Point point ) [extern]
```

Tests whether the given 2D integer point has both components equal to zero.

7.7.2.108 pointEmptyF()

```
bool pointEmptyF (
    PointF point )
```

Tests whether the given 2D floating-point vector has both components nearly zero.

7.7.2.109 pointEquals()

```
bool pointEquals (
    Point point1,
    Point point2 ) [extern]
```

Tests whether two given 2D integer points match.

7.7.2.110 pointEqualsF()

```
bool pointEqualsF (
    PointF point1,
    PointF point2 )
```

Tests whether two given 2D floating-point vectors are indistinguishable.

7.7.2.111 pointF()

```
PointF pointF (
    Point point ) [extern]
```

Creates floating-point 2D point from integer point.

7.7.2.112 pointInRect()

```
bool pointInRect (
    Point point,
    Rect rect )
```

Tests whether the specified point is contained within the given rectangle.

7.7.2.113 pointInRectF()

```
bool pointInRectF (
    PointF point,
    RectF rect )
```

Tests whether the specified point is contained within the given rectangle.

7.7.2.114 pointLength()

```
float pointLength (
    Point point ) [extern]
```

Returns length of the given 2D integer point.

7.7.2.115 pointLengthF()

```
float pointLengthF (
    PointF point ) [extern]
```

Returns length of the given 2D floating-point vector.

7.7.2.116 pointLerp()

```
Point pointLerp (
    Point point1,
    Point point2,
    float theta ) [extern]
```

Interpolates between first and second specified 2D integer vectors; theta should be in range of [0, 1].

7.7.2.117 pointLerpF()

```
PointF pointLerpF (
    PointF point1,
    PointF point2,
    float theta )
```

Interpolates between first and second specified 2D floating-point vectors; theta should be in range of [0, 1].

7.7.2.118 pointMultiply()

```
Point pointMultiply (
    Point point1,
    Point point2 ) [extern]
```

Multiplies components of two 2D integer points and returns the result.

7.7.2.119 pointMultiplyF()

```
PointF pointMultiplyF (
    PointF point1,
    PointF point2 ) [extern]
```

Multiplies components of two 2D floating-point vectors and returns the result.

7.7.2.120 pointNegate()

```
Point pointNegate (
    Point point ) [extern]
```

Returns 2D integer point with both components negated.

7.7.2.121 pointNegateF()

```
PointF pointNegateF (
    PointF point ) [extern]
```

Returns 2D floating-point vector with both components negated.

7.7.2.122 pointNormalizeF()

```
PointF pointNormalizeF (
    PointF point )
```

Normalizes given 2D floating-point vector to unity length. If the vector has zero length, it will be returned unchanged.

7.7.2.123 pointRescale()

```
Point pointRescale (
    Point point,
    int32_t theta ) [extern]
```

Multiplies components of 2D integer point by the given coefficient and returns the result.

7.7.2.124 pointRescaleF()

```
PointF pointRescaleF (
    PointF point,
    float theta ) [extern]
```

Multiplies components of 2D floating-point vector by the given coefficient and returns the result.

7.7.2.125 pointSubtract()

```
Point pointSubtract (
    Point point1,
    Point point2 ) [extern]
```

Subtracts components of the second 2D integer point from the first one and returns the result.

7.7.2.126 pointSubtractF()

```
PointF pointSubtractF (
    PointF point1,
    PointF point2 ) [extern]
```

Subtracts components of the second 2D floating-point vector from the first one and returns the result.

7.7.2.127 pointTransformF()

```
PointF pointTransformF (
    PointF point,
    Matrix3x2 matrix )
```

Multiplies given vector treated as 3D vector with $Z = 1$, by 3x2 matrix, treated as 3x3 with right side matching identity matrix.

7.7.2.128 pointValue()

```
Point pointValue (
    int32_t value ) [extern]
```

Creates 2D integer point with both X and Y set to the given value.

7.7.2.129 pointValueF()

```
PointF pointValueF (
    float value ) [extern]
```

Creates 2D floating-point vector with both X and Y set to the given value.

7.7.2.130 quadFlip()

```
Quad quadFlip (
    Quad quad )
```

Creates quadrilateral from the given one but having top vertices exchanged with the bottom ones, effectively flipping it vertically.

7.7.2.131 quadFromRect()

```
Quad quadFromRect (
    Rect rect )
```

Creates quadrilateral from rectangle with integer coordinates.

7.7.2.132 quadFromRectF()

```
Quad quadFromRectF (
    RectF rect )
```

Creates quadrilateral from rectangle with floating-point coordinates.

7.7.2.133 quadMirror()

```
Quad quadMirror (
    Quad quad )
```

Creates quadrilateral from the given one but having left vertices exchanged with the right ones, effectively mirroring it horizontally.

7.7.2.134 quadOffset()

```
Quad quadOffset (
    Quad quad,
    PointF delta )
```

Displaces vertices of the given quadrilateral by the specified offset.

7.7.2.135 quadScale()

```
Quad quadScale (
    Quad quad,
    float scale,
    bool centered )
```

Rescales vertices of the given quadrilateral by the provided coefficient, optionally centering them around zero origin.

7.7.2.136 quadTransform()

```
Quad quadTransform (
    Quad quad,
    Matrix3x2 matrix )
```

Transforms (multiplies) vertices of the given quadrilateral by the specified matrix.

7.7.2.137 quaternionAngle()

```
float quaternionAngle (
    Quaternion quat )
```

Returns rotational angle that is present in the given quaternion.

7.7.2.138 quaternionAxis()

```
Vector quaternionAxis (
    Quaternion quat )
```

Returns rotational axis that is present in the given quaternion.

7.7.2.139 quaternionConjugate()

```
Quaternion quaternionConjugate (
    Quaternion quat )
```

Computes conjugate of the given quaternion. The resulting quaternion has opposite rotation.

7.7.2.140 quaternionDot()

```
float quaternionDot (
    Quaternion quat1,
    Quaternion quat2 )
```

Computes dot product between two given quaternions.

7.7.2.141 quaternionExponentiate()

```
Quaternion quaternionExponentiate (
    Quaternion quat,
    float exponent )
```

Computes exponentiation of the given quaternion.

7.7.2.142 quaternionLength()

```
float quaternionLength (
    Quaternion quat )
```

Returns the magnitude of the given quaternion.

7.7.2.143 quaternionMatrix()

```
Matrix quaternionMatrix (
    Quaternion quat )
```

Converts the given quaternion into 4x4 matrix representation.

7.7.2.144 quaternionMultiply()

```
Quaternion quaternionMultiply (
    Quaternion quat1,
    Quaternion quat2 )
```

Multiplies quaternion by another quaternion, effectively combining their transformations.

7.7.2.145 quaternionNormalize()

```
Quaternion quaternionNormalize (
    Quaternion quat )
```

Normalizes the given quaternion. Note that normally quaternions are always normalized (of course, within limits of numerical precision). This function is provided mainly to combat floating point "error creep", which occurs after many successive quaternion operations.

7.7.2.146 quaternionRotate()

```
Quaternion quaternionRotate (
    Vector axis,
    float angle )
```

Creates 3D quaternion containing rotation around an arbitrary axis with given angle (in radians).

7.7.2.147 quaternionRotateInertialToObject()

```
Quaternion quaternionRotateInertialToObject (
    Vector angles )
```

Creates 3D quaternion setup to perform Inertial-To-Object rotation using the angles specified in Euler format. The components are taken from the specified vector with Y corresponding to Heading, X to Pitch and Z to Bank.

7.7.2.148 quaternionRotateObjectToInertial()

```
Quaternion quaternionRotateObjectToInertial (
    Vector angles )
```

Creates 3D quaternion setup to perform Object-To-Inertial rotation using the angles specified in Euler format. The components are taken from the specified vector with Y corresponding to Heading, X to Pitch and Z to Bank.

7.7.2.149 quaternionRotateX()

```
Quaternion quaternionRotateX (
    float angle )
```

Creates 3D quaternion containing rotation around X axis with given angle (in radians).

7.7.2.150 quaternionRotateY()

```
Quaternion quaternionRotateY (
    float angle )
```

Creates 3D quaternion containing rotation around Y axis with given angle (in radians).

7.7.2.151 quaternionRotateZ()

```
Quaternion quaternionRotateZ (
    float angle )
```

Creates 3D quaternion containing rotation around Z axis with given angle (in radians).

7.7.2.152 quaternionSlerp()

```
Quaternion quaternionSlerp (
    Quaternion quat1,
    Quaternion quat2,
    float theta )
```

Applies spherical linear interpolation between two given quaternions.

7.7.2.153 rectEmpty()

```
bool rectEmpty (
    Rect rect )
```

Tests whether the given rectangle is empty, that is, having width or height of zero or less.

7.7.2.154 rectEmptyF()

```
bool rectEmptyF (
    RectF rect )
```

Tests whether the given rectangle is empty, that is, having width or height of nearly zero or less.

7.7.2.155 rectEquals()

```
bool rectEquals (
    Rect rect1,
    Rect rect2 )
```

Tests whether both given rectangles are exactly the same.

7.7.2.156 rectEqualsF()

```
bool rectEqualsF (
    RectF rect1,
    RectF rect2 )
```

Tests whether both given rectangles are nearly same..

7.7.2.157 rectF()

```
RectF rectF (
    Rect rect )
```

Creates rectangle with floating-point coordinates from rectangle with integer coordinates.

7.7.2.158 rectGetBottom()

```
int32_t rectGetBottom (
    Rect rect )
```

Returns bottom edge of the given rectangle.

7.7.2.159 rectGetBottomF()

```
float rectGetBottomF (
    RectF rect )
```

Returns bottom edge of the given rectangle.

7.7.2.160 rectGetBottomLeft()

```
Point rectGetBottomLeft (
    Rect rect )
```

Returns bottom and left of the given rectangle as 2D integer vector.

7.7.2.161 rectGetBottomLeftF()

```
PointF rectGetBottomLeftF (
    RectF rect )
```

Returns bottom and left of the given rectangle as 2D integer vector.

7.7.2.162 rectGetBottomRight()

```
Point rectGetBottomRight (
    Rect rect )
```

Returns Bottom and Right of the given rectangle as 2D integer vector.

7.7.2.163 rectGetBottomRightF()

```
PointF rectGetBottomRightF (
    RectF rect )
```

Returns Bottom and Right of the given rectangle as 2D integer vector.

7.7.2.164 rectGetRight()

```
int32_t rectGetRight (
    Rect rect )
```

Returns right edge of the given rectangle.

7.7.2.165 rectGetRightF()

```
float rectGetRightF (
    RectF rect )
```

Returns right edge of the given rectangle.

7.7.2.166 rectGetSize()

```
Point rectGetSize (
    Rect rect )
```

Returns the size of given rectangle as 2D integer vector.

7.7.2.167 rectGetSizeF()

```
PointF rectGetSizeF (
    RectF rect )
```

Returns the size of given rectangle as 2D integer vector.

7.7.2.168 rectGetTopLeft()

```
Point rectGetTopLeft (
    Rect rect )
```

Returns top and left of the given rectangle as 2D integer vector.

7.7.2.169 rectGetTopLeftF()

```
PointF rectGetTopLeftF (
    RectF rect )
```

Returns top and left of the given rectangle as 2D integer vector.

7.7.2.170 rectGetTopRight()

```
Point rectGetTopRight (
    Rect rect )
```

Returns top and right of the given rectangle as 2D integer vector.

7.7.2.171 rectGetTopRightF()

```
PointF rectGetTopRightF (
    RectF rect )
```

Returns top and right of the given rectangle as 2D integer vector.

7.7.2.172 rectInflate()

```
Rect rectInflate (
    Rect rect,
    Point delta )
```

Returns rectangle with left and top decremented, while right and bottom incremented by given offset.

7.7.2.173 rectInflateF()

```
RectF rectInflateF (
    RectF rect,
    PointF delta )
```

Returns rectangle with left and top decremented, while right and bottom incremented by given offset.

7.7.2.174 rectInRect()

```
bool rectInRect (
    Rect rect,
    Rect largerRect )
```

Tests whether the specified rectangle is contained within the given rectangle.

7.7.2.175 rectInRectF()

```
bool rectInRectF (
    RectF rect,
    RectF largerRect )
```

Tests whether the specified rectangle is contained within the given rectangle.

7.7.2.176 rectIntersect()

```
Rect rectIntersect (
    Rect rect1,
    Rect rect2 )
```

Calculates rectangle that results from intersection between two given rectangles.

7.7.2.177 rectIntersectF()

```
RectF rectIntersectF (
    RectF rect1,
    RectF rect2 )
```

Calculates rectangle that results from intersection between two given rectangles.

7.7.2.178 rectJoin()

```
Rect rectJoin (
    Rect rect1,
    Rect rect2 )
```

Calculates rectangle that results from union between two given rectangles.

7.7.2.179 rectJoinF()

```
RectF rectJoinF (
    RectF rect1,
    RectF rect2 )
```

Calculates rectangle that results from union between two given rectangles.

7.7.2.180 rectOffset()

```
Rect rectOffset (
    Rect rect,
    Point delta )
```

Calculates displaced rectangle by certain offset.

7.7.2.181 rectOffsetF()

```
RectF rectOffsetF (
    RectF rect,
    PointF delta )
```

Calculates displaced rectangle by certain offset.

7.7.2.182 rectOverlap()

```
bool rectOverlap (
    Rect rect1,
    Rect rect2 )
```

Tests whether two given rectangles overlap.

7.7.2.183 rectOverlapF()

```
bool rectOverlapF (
    RectF rect1,
    RectF rect2 )
```

Tests whether two given rectangles overlap.

7.7.2.184 rectSetBottom()

```
Rect rectSetBottom (
    Rect rect,
    int32_t bottom )
```

Specifies new bottom edge for the given rectangle.

7.7.2.185 rectSetBottomF()

```
RectF rectSetBottomF (
    RectF rect,
    float bottom )
```

Specifies new bottom edge for the given rectangle.

7.7.2.186 rectSetBottomRight()

```
Rect rectSetBottomRight (
    Rect rect,
    Point bottomRight )
```

Specifies Bottom and Right for the given rectangle from values of 2D integer vector.

7.7.2.187 rectSetBottomRightF()

```
RectF rectSetBottomRightF (
    RectF rect,
    PointF bottomRight )
```

Specifies Bottom and Right for the given rectangle from values of 2D integer vector.

7.7.2.188 rectSetRight()

```
Rect rectSetRight (
    Rect rect,
    int32_t right )
```

Specifies new right edge for the given rectangle.

7.7.2.189 rectSetRightF()

```
RectF rectSetRightF (
    RectF rect,
    float right )
```

Specifies new right edge for the given rectangle.

7.7.2.190 rectSetSize()

```
Rect rectSetSize (
    Rect rect,
    Point size )
```

Specifies new size for given rectangle from values of 2D integer vector.

7.7.2.191 rectSetSizeF()

```
RectF rectSetSizeF (
    RectF rect,
    PointF size )
```

Specifies new size for given rectangle from values of 2D integer vector.

7.7.2.192 rectSetTopLeft()

```
Rect rectSetTopLeft (
    Rect rect,
    Point topLeft )
```

Specifies Top and Left for the given rectangle from values of 2D integer vector.

7.7.2.193 rectSetTopLeftF()

```
RectF rectSetTopLeftF (
    RectF rect,
    PointF topLeft )
```

Specifies Top and Left for the given rectangle from values of 2D integer vector.

7.7.2.194 rectToIntRectF()

```
Rect rectToIntRectF (
    RectF rect )
```

Creates rectangle with integer coordinates from rectangle with floating-point coordinates.

7.7.2.195 vectorAdd()

```
Vector vectorAdd (
    Vector vector1,
    Vector vector2 )
```

Adds components of two 3D vectors and returns the result.

7.7.2.196 vectorAdd3D()

```
Vector4 vectorAdd3D (
    Vector4 vector1,
    Vector4 vector2 )
```

Adds components of two 4-component vectors and returns the result.

7.7.2.197 vectorAngle()

```
float vectorAngle (
    Vector vector1,
    Vector vector2 )
```

Calculates angle between two 3D vectors. The returned value has range of [0, Pi].

7.7.2.198 vectorCross()

```
Vector vectorCross (
    Vector vector1,
    Vector vector2 )
```

Calculates cross product between two given 3D vectors. The resulting vector is perpendicular to both vectors and normal to the plane containing them.

7.7.2.199 vectorDistance()

```
float vectorDistance (
    Vector vector1,
    Vector vector2 )
```

Returns distance between two 3D vectors.

7.7.2.200 vectorDistance3D()

```
float vectorDistance3D (
    Vector4 vector1,
    Vector4 vector2 )
```

Returns distance between two 4-component vectors.

7.7.2.201 vectorDivide()

```
Vector vectorDivide (
    Vector vector1,
    Vector vector2 )
```

Divides components of the first 3D vector by the components of the second one.

7.7.2.202 vectorDivide3D()

```
Vector4 vectorDivide3D (
    Vector4 vector1,
    Vector4 vector2 )
```

Divides components of the first 4-component vector by the components of the second one.

7.7.2.203 vectorDot()

```
float vectorDot (
    Vector vector1,
    Vector vector2 )
```

Calculates dot-product between two given 3D vectors. The dot-product is an indirect measure of the angle between two vectors.

7.7.2.204 vectorDot3D()

```
float vectorDot3D (
    Vector4 vector1,
    Vector4 vector2 )
```

Calculates dot-product between two given 4-component vectors. The dot-product is an indirect measure of the angle between two vectors.

7.7.2.205 vectorEmpty()

```
bool vectorEmpty (
    Vector vector )
```

Tests whether the given 3D vector has all components nearly zero.

7.7.2.206 vectorEmpty3D()

```
bool vectorEmpty3D (
    Vector4 vector )
```

Tests whether the given 4-component vector has all components nearly zero.

7.7.2.207 vectorEquals()

```
bool vectorEquals (
    Vector vector1,
    Vector vector2 )
```

Tests whether two given 3D vectors are indistinguishable.

7.7.2.208 vectorEquals3D()

```
bool vectorEquals3D (
    Vector4 vector1,
    Vector4 vector2 )
```

Tests whether two given 4-component vectors are indistinguishable.

7.7.2.209 vectorFromValue3D()

```
Vector4 vectorFromValue3D (
    float value )
```

Creates 4-component vector with X, Y, Z and W set to the given value.

7.7.2.210 vectorLength()

```
float vectorLength (
    Vector vector )
```

Returns length of the given 3D vector.

7.7.2.211 vectorLength3D()

```
float vectorLength3D (
    Vector4 vector )
```

Returns length of the given 4-component vector.

7.7.2.212 vectorLerp()

```
Vector vectorLerp (
    Vector vector1,
    Vector vector2,
    float theta )
```

Interpolates between first and second specified 3D vectors; theta should be in range of [0, 1].

7.7.2.213 vectorLerp3D()

```
Vector4 vectorLerp3D (
    Vector4 vector1,
    Vector4 vector2,
    float theta )
```

Interpolates between first and second specified 4-component vectors; theta should be in range of [0, 1].

7.7.2.214 vectorMultiply()

```
Vector vectorMultiply (
    Vector vector1,
    Vector vector2 )
```

Multiplies components of two 3D vectors and returns the result.

7.7.2.215 vectorMultiply3D()

```
Vector4 vectorMultiply3D (
    Vector4 vector1,
    Vector4 vector2 )
```

Multiplies components of two 4-component vectors and returns the result.

7.7.2.216 vectorNegate()

```
Vector vectorNegate (
    Vector vector )
```

Returns 3D vector with all components negated.

7.7.2.217 vectorNegate3D()

```
Vector4 vectorNegate3D (
    Vector4 vector )
```

Returns 4-component vector with all components negated.

7.7.2.218 vectorNormalize()

```
Vector vectorNormalize (
    Vector vector )
```

Normalizes given 3D vector to unity length. If the vector has zero length, it will be returned unchanged.

7.7.2.219 vectorNormalize3D()

```
Vector4 vectorNormalize3D (
    Vector4 vector )
```

Normalizes given 4-component vector to unity length. If the vector has zero length, it will be returned unchanged.

7.7.2.220 vectorParallel()

```
Vector vectorParallel (
    Vector vector,
    Vector direction )
```

Calculates portion of given 3D vector that is parallel to the direction vector.

7.7.2.221 vectorPerpendicular()

```
Vector vectorPerpendicular (
    Vector vector,
    Vector direction )
```

Calculates portion of given 3D vector that is perpendicular to the direction vector.

7.7.2.222 vectorProject()

```
Vector vectorProject (
    Vector vector,
    float w,
    Matrix matrix )
```

Multiplies given 3D vector as if it was 4D (3D + W) vector by 4x4 matrix, then divides each of the resulting components by the resulting value of W (which is then discarded), returning X, Y and Z.

7.7.2.223 vectorProjectTarget()

```
PointF vectorProjectTarget (
    Vector vector,
    PointF targetSize,
    Matrix matrix )
```

Multiplies given 3D vector as if it was 4D (3D + 1) vector by 4x4 matrix, then divides each of the resulting components by the resulting value of W (which is then discarded), and converts absolute coordinates to screen pixels by using the given target size.

7.7.2.224 vectorReflect()

```
Vector vectorReflect (
    Vector vector,
    Vector normal )
```

Calculates 3D vector that is a reflection of given vector from surface given by the specified normal.

7.7.2.225 vectorRescale()

```
Vector vectorRescale (
    Vector vector,
    float theta )
```

Multiplies components of 3D vector by the given coefficient and returns the result.

7.7.2.226 vectorRescale3D()

```
Vector4 vectorRescale3D (
    Vector4 vector,
    float theta )
```

Multiplies components of 4-component vector by the given coefficient and returns the result.

7.7.2.227 vectorSubtract()

```
Vector vectorSubtract (
    Vector vector1,
    Vector vector2 )
```

Subtracts components of the second 3D vector from the first one and returns the result.

7.7.2.228 vectorSubtract3D()

```
Vector4 vectorSubtract3D (
    Vector4 vector1,
    Vector4 vector2 )
```

Subtracts components of the second 4-component vector from the first one and returns the result.

7.7.2.229 vectorTransform()

```
Vector vectorTransform (
    Vector vector,
    float w,
    Matrix matrix )
```

Multiplies given 3D vector as if it was 4D (3D + W) vector by 4x4 matrix and returns the result, discarding the fourth element.

7.7.2.230 vectorTransform3D()

```
Vector4 vectorTransform3D (
    Vector4 vector,
    Matrix matrix )
```

Multiplies given 4-component vector by the given 4x4 matrix.

7.7.2.231 vectorValue()

```
Vector vectorValue (
    float value )
```

Creates 3D vector with X, Y and Z set to the given value.

7.7.3 Variable Documentation

7.7.3.1 colorBlack

```
Color const colorBlack [extern]
```

Predefined constant for opaque Black color.

7.7.3.2 colorPairBlack

```
ColorPair const colorPairBlack [extern]
```

Predefined constant for a pair of opaque Black colors.

7.7.3.3 colorPairTraslucentBlack

```
ColorPair const colorPairTraslucentBlack [extern]
```

Predefined constant for a pair of translucent Black colors.

7.7.3.4 colorPairTraslucentWhite

```
ColorPair const colorPairTraslucentWhite [extern]
```

Predefined constant for a pair of translucent White colors.

7.7.3.5 colorPairWhite

```
ColorPair const colorPairWhite [extern]
```

Predefined constant for a pair of opaque White colors.

7.7.3.6 colorRectBlack

```
ColorRect const colorRectBlack [extern]
```

Predefined constant for four opaque Black colors.

7.7.3.7 colorRectTraslucentBlack

```
ColorRect const colorRectTraslucentBlack [extern]
```

Predefined constant for four translucent Black colors.

7.7.3.8 colorRectTraslucentWhite

```
ColorRect const colorRectTraslucentWhite [extern]
```

Predefined constant for four translucent White colors.

7.7.3.9 colorRectWhite

```
ColorRect const colorRectWhite [extern]
```

Predefined constant for four opaque White colors.

7.7.3.10 colorTranslucentBlack

```
Color const colorTranslucentBlack [extern]
```

Predefined constant for translucent Black color.

7.7.3.11 colorTranslucentWhite

```
Color const colorTranslucentWhite [extern]
```

Predefined constant for translucent White color.

7.7.3.12 colorWhite

```
Color const colorWhite [extern]
```

Predefined constant for opaque White color.

7.7.3.13 floatColorBlack

```
FloatColor const floatColorBlack [extern]
```

Predefined constant for opaque Black color.

7.7.3.14 floatColorBlackRGB

```
FloatColorRGB const floatColorBlackRGB [extern]
```

Predefined constant for opaque Black color.

7.7.3.15 floatColorTraslucentBlack

```
FloatColor const floatColorTraslucentBlack [extern]
```

Predefined constant for translucent Black color.

7.7.3.16 floatColorTraslucentWhite

```
FloatColor const floatColorTraslucentWhite [extern]
```

Predefined constant for translucent White color.

7.7.3.17 floatColorWhite

```
FloatColor const floatColorWhite [extern]
```

Predefined constant for opaque White color.

7.7.3.18 floatColorWhiteRGB

```
FloatColorRGB const floatColorWhiteRGB [extern]
```

Predefined constant for opaque White color.

7.7.3.19 fontEffectDefault

```
FontEffect const fontEffectDefault [extern]
```

Default font effect parameters provided for calling convenience.

7.7.3.20 matrixIdentity

```
Matrix const matrixIdentity [extern]
```

Predefined constant with values corresponding to an identity matrix.

7.7.3.21 matrixIdentity3x2

```
Matrix3x2 const matrixIdentity3x2 [extern]
```

Predefined constant with values corresponding to an identity matrix.

7.7.3.22 matrixZero

```
Matrix const matrixZero [extern]
```

Predefined constant, where all matrix values are zero.

7.7.3.23 matrixZero3x2

```
Matrix3x2 const matrixZero3x2 [extern]
```

Predefined constant, where all matrix values are zero.

7.7.3.24 pointAxisX

```
Point const pointAxisX [extern]
```

Predefined constant, where $X = 1$ and $Y = 0$.

7.7.3.25 pointAxisXF

```
PointF const pointAxisXF [extern]
```

Predefined constant, where $X = 1$ and $Y = 0$.

7.7.3.26 pointAxisY

```
Point const pointAxisY [extern]
```

Predefined constant, where $X = 0$ and $Y = 1$.

7.7.3.27 pointAxisYF

```
PointF const pointAxisYF [extern]
```

Predefined constant, where $X = 0$ and $Y = 1$.

7.7.3.28 pointUnity

```
Point const pointUnity [extern]
```

Predefined constant, where X and Y are one.

7.7.3.29 pointUnityF

```
PointF const pointUnityF [extern]
```

Predefined constant, where X and Y are one.

7.7.3.30 pointZero

```
Point const pointZero [extern]
```

Predefined constant, where X and Y are zero.

7.7.3.31 pointZeroF

```
PointF const pointZeroF [extern]
```

Predefined constant, where X and Y are zero.

7.7.3.32 quadUnity

```
Quad const quadUnity [extern]
```

Quadrilateral with vertices set at four corners set between (0, 0) and (1, 1).

7.7.3.33 quadZero

```
Quad const quadZero [extern]
```

Quadrilateral with all vertices set to zero.

7.7.3.34 quaternionIdentity

```
Quaternion const quaternionIdentity [extern]
```

Predefined constant with values corresponding to an identity quaternion.

7.7.3.35 rectZero

```
Rect const rectZero [extern]
```

Empty rectangle with all members set to zero.

7.7.3.36 rectZeroF

```
RectF const rectZeroF [extern]
```

Empty rectangle with all members set to zero.

7.7.3.37 vectorAxisW3D

```
Vector4 const vectorAxisW3D [extern]
```

Predefined constant, where $X = 0$, $Y = 0$, $Z = 0$ and $W = 1$.

7.7.3.38 vectorAxisX

```
Vector const vectorAxisX [extern]
```

Predefined constant, where $X = 1$, $Y = 0$ and $Z = 0$.

7.7.3.39 vectorAxisX3D

```
Vector4 const vectorAxisX3D [extern]
```

Predefined constant, where $X = 1$, $Y = 0$, $Z = 0$ and $W = 0$.

7.7.3.40 vectorAxisY

```
Vector const vectorAxisY [extern]
```

Predefined constant, where $X = 0$, $Y = 1$ and $Z = 0$.

7.7.3.41 vectorAxisY3D

```
Vector4 const vectorAxisY3D [extern]
```

Predefined constant, where $X = 0$, $Y = 1$, $Z = 0$ and $W = 0$.

7.7.3.42 vectorAxisZ

```
Vector const vectorAxisZ [extern]
```

Predefined constant, where $X = 0$, $Y = 0$ and $Z = 1$.

7.7.3.43 vectorAxisZ3D

```
Vector4 const vectorAxisZ3D [extern]
```

Predefined constant, where $X = 0$, $Y = 0$, $Z = 1$ and $W = 0$.

7.7.3.44 vectorEpsilon

```
float const vectorEpsilon [extern]
```

A special value that determines absolute precision limit when comparing vectors and coordinates. This assumes that the corresponding vectors have their values normalized to $[0, 1]$ range.

7.7.3.45 vectorUnity

```
Vector const vectorUnity [extern]
```

Predefined constant, where X , Y and Z are one.

7.7.3.46 vectorUnity3D

```
Vector4 const vectorUnity3D [extern]
```

Predefined constant, where X , Y , Z and W are one.

7.7.3.47 vectorZero

```
Vector const vectorZero [extern]
```

Predefined constant, where X, Y and Z are zero.

7.7.3.48 vectorZero3D

```
Vector4 const vectorZero3D [extern]
```

Predefined constant, where X, Y, Z and W are zero.

7.8 Afterwarp.Vectors.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  * This file is part of Afterwarp Framework (http://afterwarp.io).
00003  * Copyright (c) 2015 - 2025 Dr. Yuriy Kotsarenko. All rights reserved.
00004  *
00005  * Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in
00006  * compliance with the License. You may obtain a copy of the License at
00007  * http://www.apache.org/licenses/LICENSE-2.0
00008  *
00009  * Unless required by applicable law or agreed to in writing, software distributed under the License
00010  * is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
00011  * implied.
00012  * See the License for the specific language governing permissions and limitations under the License.
00013  */
00014 // Afterwarp.Vectors.h
00015
00016 #ifndef AFTERWARP_VECTORS_IMPLEMENTATION
00017 #define AFTERWARP_VECTORS_INTERFACE
00018 #endif
00019
00020 #ifdef AFTERWARP_VECTORS_INTERFACE
00021
00022 #ifndef AFTERWARP_VECTORS_H
00023 #define AFTERWARP_VECTORS_H
00024
00025 #include "Afterwarp.Types.h"
00026 #include "Afterwarp.Structs.h"
00027
00028 // Global macros.
00029
00030 #ifndef MAX
00031 #define MAX(x, y) (((x) > (y)) ? (x) : (y))
00032 #endif
00033 #ifndef MIN
00034 #define MIN(x, y) (((x) < (y)) ? (x) : (y))
00035 #endif
00036
00039 extern float const vectorEpsilon;
00040
00042 extern FontEffect const fontEffectDefault;
00043
00044 // Color functions.
00045
00047 extern Color const colorBlack;
00048
00050 extern Color const colorWhite;
00051
00053 extern Color const colorTranslucentBlack;
00054
00056 extern Color const colorTranslucentWhite;
00057
00058 // ColorPair constants and functions.
00059
00061 extern ColorPair const colorPairBlack;
00062
00064 extern ColorPair const colorPairWhite;
00065
```

```

00067 extern ColorPair const colorPairTraslucentBlack;
00068
00070 extern ColorPair const colorPairTraslucentWhite;
00071
00073 extern ColorPair makeColorPair(Color first, Color second);
00074
00076 extern ColorPair colorPair(Color color);
00077
00078 // ColorRect constants.
00079
00081 extern ColorRect const colorRectBlack;
00082
00084 extern ColorRect const colorRectWhite;
00085
00087 extern ColorRect const colorRectTraslucentBlack;
00088
00090 extern ColorRect const colorRectTraslucentWhite;
00091
00092 // FloatColorRGB constants and functions.
00093
00095 extern FloatColorRGB const floatColorBlackRGB;
00096
00098 extern FloatColorRGB const floatColorWhiteRGB;
00099
00101 extern FloatColorRGB makeFloatColorRGB(float red, float green, float blue);
00102
00104 extern FloatColorRGB floatColorRGB(Color color);
00105
00107 extern FloatColorRGB floatColorFromValueRGB(float value);
00108
00110 extern FloatColorRGB floatColorNegateRGB(FloatColorRGB color);
00111
00113 extern FloatColorRGB floatColorInverseRGB(FloatColorRGB color);
00114
00116 extern FloatColorRGB floatColorSaturateRGB(FloatColorRGB color);
00117
00119 extern FloatColorRGB floatColorAddRGB(FloatColorRGB color1, FloatColorRGB color2);
00120
00122 extern FloatColorRGB floatColorSubtractRGB(FloatColorRGB color1, FloatColorRGB color2);
00123
00125 extern FloatColorRGB floatColorMultiplyRGB(FloatColorRGB color1, FloatColorRGB color2);
00126
00128 extern FloatColorRGB floatColorDivideRGB(FloatColorRGB color1, FloatColorRGB color2);
00129
00131 extern FloatColorRGB floatColorAverageRGB(FloatColorRGB color1, FloatColorRGB color2);
00132
00134 extern FloatColorRGB floatColorLerpRGB(FloatColorRGB color1, FloatColorRGB color2, float theta);
00135
00137 extern bool floatColorEqualsRGB(FloatColorRGB color1, FloatColorRGB color2);
00138
00140 extern bool floatColorEmptyRGB(FloatColorRGB color);
00141
00144 extern float floatColorGrayRGB(FloatColorRGB color);
00145
00147 extern Color floatToColorRGB(FloatColorRGB color);
00148
00149 // FloatColor constants and functions.
00150
00152 extern FloatColor const floatColorBlack;
00153
00155 extern FloatColor const floatColorWhite;
00156
00158 extern FloatColor const floatColorTraslucentBlack;
00159
00161 extern FloatColor const floatColorTraslucentWhite;
00162
00164 extern FloatColor makeFloatColor(float red, float green, float blue, float alpha);
00165
00167 extern FloatColor floatColor(Color color);
00168
00170 extern FloatColor floatColorFromValue(float value);
00171
00173 extern FloatColor floatColorNegate(FloatColor color);
00174
00176 extern FloatColor floatColorInverse(FloatColor color);
00177
00180 extern FloatColor floatColorPremultiply(FloatColor color);
00181
00184 extern FloatColor floatColorUnpremultiply(FloatColor color);
00185
00187 extern FloatColor floatColorSaturate(FloatColor color);
00188
00190 extern FloatColor floatColorAdd(FloatColor color1, FloatColor color2);
00191
00193 extern FloatColor floatColorSubtract(FloatColor color1, FloatColor color2);
00194
00196 extern FloatColor floatColorMultiply(FloatColor color1, FloatColor color2);

```

```

00197
00199 extern FloatColor floatColorDivide(FloatColor color1, FloatColor color2);
00200
00202 extern FloatColor floatColorAverage(FloatColor color1, FloatColor color2);
00203
00205 extern FloatColor floatColorLerp(FloatColor color1, FloatColor color2, float theta);
00206
00208 extern bool floatColorEquals(FloatColor color1, FloatColor color2);
00209
00211 extern bool floatColorEmpty(FloatColor color);
00212
00215 extern float floatColorGray(FloatColor color);
00216
00218 extern Color floatToColor(FloatColor color);
00219
00221 extern FloatColorRGB floatColorToRGB(FloatColor color);
00222
00223 // Point constants and functions.
00224
00226 extern Point const pointZero;
00227
00229 extern Point const pointUnity;
00230
00232 extern Point const pointAxisX;
00233
00235 extern Point const pointAxisY;
00236
00238 extern Point makePoint(int32_t x, int32_t y);
00239
00241 extern Point pointValue(int32_t value);
00242
00244 extern Point pointNegate(Point point);
00245
00247 extern Point pointAdd(Point point1, Point point2);
00248
00250 extern Point pointSubtract(Point point1, Point point2);
00251
00253 extern Point pointMultiply(Point point1, Point point2);
00254
00256 extern Point pointDivide(Point point1, Point point2);
00257
00259 extern Point pointRescale(Point point, int32_t theta);
00260
00262 extern float pointLength(Point point);
00263
00265 extern float pointDistance(Point point1, Point point2);
00266
00268 extern float pointAngle(Point point);
00269
00272 extern int32_t pointDot(Point point1, Point point2);
00273
00275 extern int32_t pointCross(Point point1, Point point2);
00276
00278 extern Point pointLerp(Point point1, Point point2, float theta);
00279
00281 extern bool pointEquals(Point point1, Point point2);
00282
00284 extern bool pointEmpty(Point point);
00285
00286 // PointF constants and functions.
00287
00289 extern PointF const pointZeroF;
00290
00292 extern PointF const pointUnityF;
00293
00295 extern PointF const pointAxisXF;
00296
00298 extern PointF const pointAxisYF;
00299
00301 extern PointF makePointF(float x, float y);
00302
00304 extern PointF pointF(Point point);
00305
00307 extern PointF pointValueF(float value);
00308
00310 extern PointF pointNegateF(PointF point);
00311
00313 extern PointF pointAddF(PointF point1, PointF point2);
00314
00316 extern PointF pointSubtractF(PointF point1, PointF point2);
00317
00319 extern PointF pointMultiplyF(PointF point1, PointF point2);
00320
00322 extern PointF pointDivideF(PointF point1, PointF point2);
00323
00325 extern PointF pointRescaleF(PointF point, float theta);
00326

```

```

00328 extern float pointLengthF(PointF point);
00329
00331 extern float pointDistanceF(PointF point1, PointF point2);
00332
00335 float pointAngleF(PointF point);
00336
00339 float pointDotF(PointF point1, PointF point2);
00340
00342 float pointCrossF(PointF point1, PointF point2);
00343
00346 PointF pointNormalizeF(PointF point);
00347
00350 PointF pointLerpF(PointF point1, PointF point2, float theta);
00351
00353 bool pointEqualsF(PointF point1, PointF point2);
00354
00356 bool pointEmptyF(PointF point);
00357
00360 PointF pointTransformF(PointF point, Matrix3x2 matrix);
00361
00362 // Vector constants and functions.
00363
00365 extern Vector const vectorZero;
00366
00368 extern Vector const vectorUnity;
00369
00371 extern Vector const vectorAxisX;
00372
00374 extern Vector const vectorAxisY;
00375
00377 extern Vector const vectorAxisZ;
00378
00380 Vector makeVector(float x, float y, float z);
00381
00383 Vector vectorValue(float value);
00384
00386 Vector vectorNegate(Vector vector);
00387
00389 Vector vectorAdd(Vector vector1, Vector vector2);
00390
00392 Vector vectorSubtract(Vector vector1, Vector vector2);
00393
00395 Vector vectorMultiply(Vector vector1, Vector vector2);
00396
00398 Vector vectorDivide(Vector vector1, Vector vector2);
00399
00401 Vector vectorRescale(Vector vector, float theta);
00402
00405 float vectorDot(Vector vector1, Vector vector2);
00406
00409 Vector vectorCross(Vector vector1, Vector vector2);
00410
00412 float vectorLength(Vector vector);
00413
00415 float vectorDistance(Vector vector1, Vector vector2);
00416
00418 float vectorAngle(Vector vector1, Vector vector2);
00419
00421 Vector vectorNormalize(Vector vector);
00422
00424 Vector vectorParallel(Vector vector, Vector direction);
00425
00427 Vector vectorPerpendicular(Vector vector, Vector direction);
00428
00430 Vector vectorReflect(Vector vector, Vector normal);
00431
00433 Vector vectorLerp(Vector vector1, Vector vector2, float theta);
00434
00436 bool vectorEquals(Vector vector1, Vector vector2);
00437
00439 bool vectorEmpty(Vector vector);
00440
00443 Vector vectorTransform(Vector vector, float w, Matrix matrix);
00444
00447 Vector vectorProject(Vector vector, float w, Matrix matrix);
00448
00452 PointF vectorProjectTarget(Vector vector, PointF targetSize, Matrix matrix);
00453
00454 // Vector4 constants and functions.
00455
00457 extern Vector4 const vectorZero3D;
00458
00460 extern Vector4 const vectorUnity3D;
00461
00463 extern Vector4 const vectorAxisX3D;
00464
00466 extern Vector4 const vectorAxisY3D;

```



```

00467
00469 extern Vector4 const vectorAxisZ3D;
00470
00472 extern Vector4 const vectorAxisW3D;
00473
00475 Vector4 makeVector3D(float x, float y, float z, float w);
00476
00478 Vector4 vectorFromValue3D(float value);
00479
00481 Vector4 vectorNegate3D(Vector4 vector);
00482
00484 Vector4 vectorAdd3D(Vector4 vector1, Vector4 vector2);
00485
00487 Vector4 vectorSubtract3D(Vector4 vector1, Vector4 vector2);
00488
00490 Vector4 vectorMultiply3D(Vector4 vector1, Vector4 vector2);
00491
00493 Vector4 vectorDivide3D(Vector4 vector1, Vector4 vector2);
00494
00496 Vector4 vectorRescale3D(Vector4 vector, float theta);
00497
00500 float vectorDot3D(Vector4 vector1, Vector4 vector2);
00501
00503 float vectorLength3D(Vector4 vector);
00504
00506 float vectorDistance3D(Vector4 vector1, Vector4 vector2);
00507
00510 Vector4 vectorNormalize3D(Vector4 vector);
00511
00513 Vector4 vectorLerp3D(Vector4 vector1, Vector4 vector2, float theta);
00514
00516 bool vectorEquals3D(Vector4 vector1, Vector4 vector2);
00517
00519 bool vectorEmpty3D(Vector4 vector);
00520
00522 Vector4 vectorTransform3D(Vector4 vector, Matrix matrix);
00523
00524 // Matrix3x2 constants and functions.
00525
00527 extern Matrix3x2 const matrixZero3x2;
00528
00530 extern Matrix3x2 const matrixIdentity3x2;
00531
00533 Matrix3x2 matrixMultiply3x2(Matrix3x2 matrix1, Matrix3x2 matrix2);
00534
00536 Matrix3x2 matrixRescale3x2(Matrix3x2 matrix, float theta);
00537
00539 float matrixDeterminant3x2(Matrix3x2 matrix);
00540
00542 Matrix3x2 matrixInverse3x2(Matrix3x2 matrix);
00543
00545 Matrix3x2 matrixInverseTranspose3x2(Matrix3x2 matrix);
00546
00548 Matrix3x2 matrixScale3x2(PointF scale);
00549
00551 Matrix3x2 matrixScaleBy3x2(PointF center, PointF scale);
00552
00554 Matrix3x2 matrixTranslate3x2(PointF offset);
00555
00557 Matrix3x2 matrixRotate3x2(float angle);
00558
00560 Matrix3x2 matrixRotateBy3x2(PointF center, float angle);
00561
00563 bool matrixEquals3x2(Matrix3x2 matrix1, Matrix3x2 matrix2);
00564
00566 bool matrixEmpty3x2(Matrix3x2 matrix);
00567
00568 // Matrix constants and functions.
00569
00571 extern Matrix const matrixZero;
00572
00574 extern Matrix const matrixIdentity;
00575
00577 Matrix3x2 matrixGet3x2(Matrix matrix);
00578
00580 Matrix matrixMultiply(Matrix matrix1, Matrix matrix2);
00581
00583 Matrix matrixRescale(Matrix matrix, float theta);
00584
00586 float matrixDeterminant(Matrix matrix);
00587
00589 Matrix matrixTranspose(Matrix matrix);
00590
00592 Matrix matrixAdjoint(Matrix matrix);
00593
00595 Matrix matrixInverse(Matrix matrix);
00596

```

```

00598 Matrix matrixSub3x3(Matrix matrix);
00599
00602 Vector matrixEyePosition(Matrix matrix);
00603
00606 Vector matrixWorldPosition(Matrix matrix);
00607
00609 Matrix matrixTranslate(Vector offset);
00610
00612 Matrix matrixScale(Vector scale);
00613
00615 Matrix matrixRotateX(float angle);
00616
00618 Matrix matrixRotateY(float angle);
00619
00621 Matrix matrixRotateZ(float angle);
00622
00624 Matrix matrixRotate(Vector axis, float angle);
00625
00629 Matrix matrixHeadingPitchBank(Vector angles);
00630
00634 Matrix matrixYawPitchRoll(Vector angles);
00635
00637 Matrix matrixReflect(Vector axis);
00638
00641 Matrix matrixLookAt(Vector origin, Vector target, Vector ceiling);
00642
00651 Matrix matrixPerspectiveFOVY(float fieldOfView, float aspectRatio, float minRange, float maxRange,
00652     bool negativeDepthRange);
00653
00662 Matrix matrixPerspectiveFOVX(float fieldOfView, float aspectRatio, float minRange, float maxRange,
00663     bool negativeDepthRange);
00664
00666 Matrix matrixPerspectiveVOL(float width, float height, float minRange, float maxRange);
00667
00669 Matrix matrixPerspectiveBDS(float left, float top, float right, float bottom, float minRange,
00670     float maxRange);
00671
00673 Matrix matrixOrthogonalVOL(float width, float height, float minRange, float maxRange);
00674
00676 Matrix matrixOrthogonalBDS(float left, float top, float right, float bottom, float minRange,
00677     float maxRange);
00678
00680 bool matrixEquals(Matrix matrix1, Matrix matrix2);
00681
00683 bool matrixEmpty(Matrix matrix);
00684
00685 // Quaternion constants and functions.
00686
00688 extern Quaternion const quaternionIdentity;
00689
00691 Quaternion quaternionMultiply(Quaternion quat1, Quaternion quat2);
00692
00694 float quaternionDot(Quaternion quat1, Quaternion quat2);
00695
00697 float quaternionLength(Quaternion quat);
00698
00700 float quaternionAngle(Quaternion quat);
00701
00703 Vector quaternionAxis(Quaternion quat);
00704
00708 Quaternion quaternionNormalize(Quaternion quat);
00709
00711 Quaternion quaternionConjugate(Quaternion quat);
00712
00714 Quaternion quaternionExponentiate(Quaternion quat, float exponent);
00715
00717 Quaternion quaternionSlerp(Quaternion quat1, Quaternion quat2, float theta);
00718
00720 Matrix quaternionMatrix(Quaternion quat);
00721
00723 Quaternion matrixQuaternion(Matrix matrix);
00724
00726 Quaternion quaternionRotateX(float angle);
00727
00729 Quaternion quaternionRotateY(float angle);
00730
00732 Quaternion quaternionRotateZ(float angle);
00733
00735 Quaternion quaternionRotate(Vector axis, float angle);
00736
00740 Quaternion quaternionRotateObjectToInertial(Vector angles);
00741
00745 Quaternion quaternionRotateInertialToObject(Vector angles);
00746
00747 // Rect constants and functions.
00748
00750 extern Rect const rectZero;

```

```

00751
00753 Rect makeRect(int32_t left, int32_t top, int32_t width, int32_t height);
00754
00756 Rect makeBounds(int32_t left, int32_t top, int32_t right, int32_t bottom);
00757
00759 Point rectGetSize(Rect rect);
00760
00762 Rect rectSetSize(Rect rect, Point size);
00763
00765 int32_t rectGetRight(Rect rect);
00766
00768 Rect rectSetRight(Rect rect, int32_t right);
00769
00771 int32_t rectGetBottom(Rect rect);
00772
00774 Rect rectSetBottom(Rect rect, int32_t bottom);
00775
00777 Point rectGetTopLeft(Rect rect);
00778
00780 Rect rectSetTopLeft(Rect rect, Point topLeft);
00781
00783 Point rectGetBottomRight(Rect rect);
00784
00786 Rect rectSetBottomRight(Rect rect, Point bottomRight);
00787
00789 Point rectGetTopRight(Rect rect);
00790
00792 Point rectGetBottomLeft(Rect rect);
00793
00795 bool rectEquals(Rect rect1, Rect rect2);
00796
00798 bool rectEmpty(Rect rect);
00799
00801 bool pointInRect(Point point, Rect rect);
00802
00804 bool rectInRect(Rect rect, Rect largerRect);
00805
00807 bool rectOverlap(Rect rect1, Rect rect2);
00808
00810 Rect rectIntersect(Rect rect1, Rect rect2);
00811
00813 Rect rectJoin(Rect rect1, Rect rect2);
00814
00816 Rect rectOffset(Rect rect, Point delta);
00817
00819 Rect rectInflate(Rect rect, Point delta);
00820
00821 // RectF constants and functions.
00822
00824 extern RectF const rectZeroF;
00825
00827 RectF makeRectF(float left, float top, float width, float height);
00828
00830 RectF makeBoundsF(float left, float top, float right, float bottom);
00831
00833 RectF rectF(Rect rect);
00834
00836 Rect rectToIntRectF(RectF rect);
00837
00839 PointF rectGetSizeF(RectF rect);
00840
00842 RectF rectSetSizeF(RectF rect, PointF size);
00843
00845 float rectGetRightF(RectF rect);
00846
00848 RectF rectSetRightF(RectF rect, float right);
00849
00851 float rectGetBottomF(RectF rect);
00852
00854 RectF rectSetBottomF(RectF rect, float bottom);
00855
00857 PointF rectGetTopLeftF(RectF rect);
00858
00860 RectF rectSetTopLeftF(RectF rect, PointF topLeft);
00861
00863 PointF rectGetBottomRightF(RectF rect);
00864
00866 RectF rectSetBottomRightF(RectF rect, PointF bottomRight);
00867
00869 PointF rectGetTopRightF(RectF rect);
00870
00872 PointF rectGetBottomLeftF(RectF rect);
00873
00875 bool rectEqualsF(RectF rect1, RectF rect2);
00876
00878 bool rectEmptyF(RectF rect);
00879

```

```

00881 bool pointInRectF(PointF point, RectF rect);
00882
00884 bool rectInRectF(RectF rect, RectF largerRect);
00885
00887 bool rectOverlapF(RectF rect1, RectF rect2);
00888
00890 RectF rectIntersectF(RectF rect1, RectF rect2);
00891
00893 RectF rectJoinF(RectF rect1, RectF rect2);
00894
00896 RectF rectOffsetF(RectF rect, PointF delta);
00897
00899 RectF rectInflateF(RectF rect, PointF delta);
00900
00901 // Quad constants and functions.
00902
00904 extern Quad const quadZero;
00905
00907 extern Quad const quadUnity;
00908
00910 Quad makeQuadWith(PointF topLeft, PointF topRight, PointF bottomRight, PointF bottomLeft);
00911
00913 Quad makeQuad(float left, float top, float width, float height);
00914
00917 Quad makeQuadScaled(float left, float top, float width, float height, float scale, bool centered);
00918
00922 Quad makeQuadRotatedAt(PointF origin, PointF size, PointF center, float angle, float scale);
00923
00926 Quad makeQuadRotated(PointF origin, PointF size, float angle, float scale);
00927
00933 Quad makeQuadRotatedTL(PointF topLeft, PointF size, PointF center, float angle, float scale);
00934
00937 Quad makeQuadSkewedHoriz(PointF origin, PointF size, float angle);
00938
00941 Quad makeQuadSkewedVert(PointF origin, PointF size, float angle);
00942
00944 Quad quadFromRect(Rect rect);
00945
00947 Quad quadFromRectF(RectF rect);
00948
00951 Quad quadScale(Quad quad, float scale, bool centered);
00952
00955 Quad quadMirror(Quad quad);
00956
00959 Quad quadFlip(Quad quad);
00960
00962 Quad quadTransform(Quad quad, Matrix3x2 matrix);
00963
00965 Quad quadOffset(Quad quad, PointF delta);
00966
00967 #endif // AFTERWARP_VECTORS_H
00968 #endif // AFTERWARP_VECTORS_INTERFACE
00969 #ifdef AFTERWARP_VECTORS_IMPLEMENTATION
00970
00971 #include <math.h>
00972 #include <string.h>
00973 #include <stdint.h>
00974
00975 // Forward declarations.
00976
00977 static bool nearlyEqualFloats(float left, float right);
00978 static bool nearlyZeroFloat(float value);
00979 static float saturateFloat(float value, float min, float max);
00980 static float lerpFloat(float value1, float value2, float theta);
00981
00982 // Global constants.
00983
00984 float const vectorEpsilon = 1E-6f;
00985
00986 // Font constants.
00987
00988 FontEffect const fontEffectDefault = {1.0f, 1.0f, FONT_BORDER_NONE, 1.0f, 0.25f, 0.75f, 3.0f,
00989 {2.0f, 2.0f}, 0.15f, 0.0f, 0.0f};
00990
00991 // Color constants and functions.
00992
00993 Color const colorBlack = 0xFF000000;
00994 Color const colorWhite = 0xFFFFFFFF;
00995 Color const colorTranslucentBlack = 0x00000000;
00996 Color const colorTranslucentWhite = 0x00FFFFFF;
00997
00998 // ColorPair constants and functions.
00999
01000 ColorPair const colorPairBlack = {.first = 0xFF000000, .second = 0xFF000000};
01001 ColorPair const colorPairWhite = {.first = 0xFFFFFFFF, .second = 0xFFFFFFFF};
01002 ColorPair const colorPairTraslucentBlack = {.first = 0x00000000, .second = 0x00000000};
01003 ColorPair const colorPairTraslucentWhite = {.first = 0x00FFFFFF, .second = 0x00FFFFFF};

```

```

01004
01005 ColorPair makeColorPair(Color first, Color second)
01006 {
01007     return (ColorPair){.first = first, .second = second};
01008 }
01009
01010 ColorPair colorPair(Color color)
01011 {
01012     return (ColorPair){.first = color, .second = color};
01013 }
01014
01015 // ColorRect constants.
01016
01017 ColorRect const colorRectBlack = {0xFF000000, 0xFF000000, 0xFF000000, 0xFF000000};
01018 ColorRect const colorRectWhite = {0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF};
01019 ColorRect const colorRectTraslucentBlack = {0x00000000, 0x00000000, 0x00000000, 0x00000000};
01020 ColorRect const colorRectTraslucentWhite = {0x00FFFFFF, 0x00FFFFFF, 0x00FFFFFF, 0x00FFFFFF};
01021
01022 // FloatColorRGB constants.
01023
01024 FloatColorRGB const floatColorBlackRGB = {.red = 0.0f, .green = 0.0f, .blue = 0.0f};
01025
01026 FloatColorRGB const floatColorWhiteRGB = {.red = 1.0f, .green = 1.0f, .blue = 1.0f};
01027
01028 // FloatColorRGB functions.
01029
01030 FloatColorRGB makeFloatColorRGB(float red, float green, float blue)
01031 {
01032     return (FloatColorRGB){.red = red, .green = green, .blue = blue};
01033 }
01034
01035 FloatColorRGB floatColorRGB(Color color)
01036 {
01037     return (FloatColorRGB){
01038         .red = (color & 0xFFu) / 255.0f,
01039         .green = ((color >> 8) & 0xFFu) / 255.0f,
01040         .blue = ((color >> 16) & 0xFFu) / 255.0f;
01041     }
01042 }
01043
01044 FloatColorRGB floatColorFromValueRGB(float value)
01045 {
01046     return (FloatColorRGB){
01047         .red = value,
01048         .green = value,
01049         .blue = value;
01050     }
01051 }
01052
01053 FloatColorRGB floatColorNegateRGB(FloatColorRGB color)
01054 {
01055     return (FloatColorRGB){
01056         .red = -color.red,
01057         .green = -color.green,
01058         .blue = -color.blue;
01059     }
01060 }
01061
01062 FloatColorRGB floatColorInverseRGB(FloatColorRGB color)
01063 {
01064     return (FloatColorRGB){
01065         .red = 1.0f - color.red,
01066         .green = 1.0f - color.green,
01067         .blue = 1.0f - color.blue;
01068     }
01069 }
01070
01071 FloatColorRGB floatColorSaturateRGB(FloatColorRGB color)
01072 {
01073     return (FloatColorRGB){
01074         .red = saturateFloat(color.red, 0.0f, 1.0f),
01075         .green = saturateFloat(color.green, 0.0f, 1.0f),
01076         .blue = saturateFloat(color.blue, 0.0f, 1.0f);
01077     }
01078 }
01079
01080 FloatColorRGB floatColorAddRGB(FloatColorRGB color1, FloatColorRGB color2)
01081 {
01082     return (FloatColorRGB){
01083         .red = color1.red + color2.red,
01084         .green = color1.green + color2.green,
01085         .blue = color1.blue + color2.blue;
01086     }
01087 }
01088
01089 FloatColorRGB floatColorSubtractRGB(FloatColorRGB color1, FloatColorRGB color2)
01090 {
01091     return (FloatColorRGB){
01092         .red = color1.red - color2.red,
01093         .green = color1.green - color2.green,
01094         .blue = color1.blue - color2.blue;
01095     }
01096 }

```

```

01091 FloatColorRGB floatColorMultiplyRGB(FloatColorRGB color1, FloatColorRGB color2)
01092 {
01093     return (FloatColorRGB){
01094         .red = color1.red * color2.red,
01095         .green = color1.green * color2.green,
01096         .blue = color1.blue * color2.blue};
01097 }
01098
01099 FloatColorRGB floatColorDivideRGB(FloatColorRGB color1, FloatColorRGB color2)
01100 {
01101     return (FloatColorRGB){
01102         .red = color1.red / color2.red,
01103         .green = color1.green / color2.green,
01104         .blue = color1.blue / color2.blue};
01105 }
01106
01107 FloatColorRGB floatColorAverageRGB(FloatColorRGB color1, FloatColorRGB color2)
01108 {
01109     return (FloatColorRGB){
01110         .red = (color1.red + color2.red) * 0.5f,
01111         .green = (color1.green + color2.green) * 0.5f,
01112         .blue = (color1.blue + color2.blue) * 0.5f};
01113 }
01114
01115 FloatColorRGB floatColorLerpRGB(FloatColorRGB color1, FloatColorRGB color2, float theta)
01116 {
01117     return (FloatColorRGB){
01118         .red = lerpFloat(color1.red, color2.red, theta),
01119         .green = lerpFloat(color1.green, color2.green, theta),
01120         .blue = lerpFloat(color1.blue, color2.blue, theta)};
01121 }
01122
01123 bool floatColorEqualsRGB(FloatColorRGB color1, FloatColorRGB color2)
01124 {
01125     return
01126         nearlyEqualFloats(color1.red, color2.red) &&
01127         nearlyEqualFloats(color1.green, color2.green) &&
01128         nearlyEqualFloats(color1.blue, color2.blue);
01129 }
01130
01131 bool floatColorEmptyRGB(FloatColorRGB color)
01132 {
01133     return
01134         nearlyZeroFloat(color.red) &&
01135         nearlyZeroFloat(color.green) &&
01136         nearlyZeroFloat(color.blue);
01137 }
01138
01139 float floatColorGrayRGB(FloatColorRGB color)
01140 {
01141     return color.red * 0.29889531f + color.green * 0.58662247f + color.blue * 0.11448222f;
01142 }
01143
01144 Color floatToColorRGB(FloatColorRGB color)
01145 {
01146     return (Color)roundf(color.red * 255.0f) | ((Color)roundf(color.green * 255.0f) << 8) |
01147         ((Color)roundf(color.blue * 255.0f) << 16) | 0xFF000000u;
01148 }
01149
01150 // FloatColor constants.
01151
01152 FloatColor const floatColorBlack = {.red = 0.0f, .green = 0.0f, .blue = 0.0f, .alpha = 1.0f};
01153
01154 FloatColor const floatColorWhite = {.red = 1.0f, .green = 1.0f, .blue = 1.0f, .alpha = 1.0f};
01155
01156 FloatColor const floatColorTraslucentBlack = {.red = 0.0f, .green = 0.0f, .blue = 0.0f, .alpha =
01157     0.0f};
01158
01159 FloatColor const floatColorTraslucentWhite = {.red = 1.0f, .green = 1.0f, .blue = 1.0f, .alpha =
01160     0.0f};
01161
01162 // FloatColor functions.
01163
01164 FloatColor makeFloatColor(float red, float green, float blue, float alpha)
01165 {
01166     return (FloatColor){.red = red, .green = green, .blue = blue, .alpha = alpha};
01167 }
01168
01169 FloatColor floatColor(Color color)
01170 {
01171     return (FloatColor){
01172         .red = (color & 0xFFu) / 255.0f,
01173         .green = ((color >> 8) & 0xFFu) / 255.0f,
01174         .blue = ((color >> 16) & 0xFFu) / 255.0f,
01175         .alpha = ((color >> 24) & 0xFFu) / 255.0f};
01176 }

```

```

01176 FloatColor floatColorFromValue(float value)
01177 {
01178     return (FloatColor){
01179         .red = value,
01180         .green = value,
01181         .blue = value,
01182         .alpha = value};
01183 }
01184
01185 FloatColor floatColorNegate(FloatColor color)
01186 {
01187     return (FloatColor){
01188         .red = -color.red,
01189         .green = -color.green,
01190         .blue = -color.blue,
01191         .alpha = -color.alpha};
01192 }
01193
01194 FloatColor floatColorInverse(FloatColor color)
01195 {
01196     return (FloatColor){
01197         .red = 1.0f - color.red,
01198         .green = 1.0f - color.green,
01199         .blue = 1.0f - color.blue,
01200         .alpha = 1.0f - color.alpha};
01201 }
01202
01203 FloatColor floatColorPremultiply(FloatColor color)
01204 {
01205     return (FloatColor){
01206         .red = color.red * color.alpha,
01207         .green = color.green * color.alpha,
01208         .blue = color.blue * color.alpha,
01209         .alpha = color.alpha};
01210 }
01211
01212 FloatColor floatColorUnpremultiply(FloatColor color)
01213 {
01214     if (fabsf(color.alpha) >= vectorEpsilon)
01215     {
01216         float const invAlpha = 1.0f / color.alpha;
01217
01218         return (FloatColor){
01219             .red = color.red * invAlpha,
01220             .green = color.green * invAlpha,
01221             .blue = color.blue * invAlpha,
01222             .alpha = color.alpha * invAlpha};
01223     }
01224     else
01225         return color;
01226 }
01227
01228 FloatColor floatColorSaturate(FloatColor color)
01229 {
01230     return (FloatColor){
01231         .red = saturateFloat(color.red, 0.0f, 1.0f),
01232         .green = saturateFloat(color.green, 0.0f, 1.0f),
01233         .blue = saturateFloat(color.blue, 0.0f, 1.0f),
01234         .alpha = saturateFloat(color.alpha, 0.0f, 1.0f)};
01235 }
01236
01237 FloatColor floatColorAdd(FloatColor color1, FloatColor color2)
01238 {
01239     return (FloatColor){
01240         .red = color1.red + color2.red,
01241         .green = color1.green + color2.green,
01242         .blue = color1.blue + color2.blue,
01243         .alpha = color1.alpha + color2.alpha};
01244 }
01245
01246 FloatColor floatColorSubtract(FloatColor color1, FloatColor color2)
01247 {
01248     return (FloatColor){
01249         .red = color1.red - color2.red,
01250         .green = color1.green - color2.green,
01251         .blue = color1.blue - color2.blue,
01252         .alpha = color1.alpha - color2.alpha};
01253 }
01254
01255 FloatColor floatColorMultiply(FloatColor color1, FloatColor color2)
01256 {
01257     return (FloatColor){
01258         .red = color1.red * color2.red,
01259         .green = color1.green * color2.green,
01260         .blue = color1.blue * color2.blue,
01261         .alpha = color1.alpha * color2.alpha};
01262 }

```

```

01263
01264 FloatColor floatColorDivide(FloatColor color1, FloatColor color2)
01265 {
01266     return (FloatColor){
01267         .red = color1.red / color2.red,
01268         .green = color1.green / color2.green,
01269         .blue = color1.blue / color2.blue,
01270         .alpha = color1.alpha / color2.alpha};
01271 }
01272
01273 FloatColor floatColorAverage(FloatColor color1, FloatColor color2)
01274 {
01275     return (FloatColor){
01276         .red = (color1.red + color2.red) * 0.5f,
01277         .green = (color1.green + color2.green) * 0.5f,
01278         .blue = (color1.blue + color2.blue) * 0.5f,
01279         .alpha = (color1.alpha + color2.alpha) * 0.5f};
01280 }
01281
01282 FloatColor floatColorLerp(FloatColor color1, FloatColor color2, float theta)
01283 {
01284     return (FloatColor){
01285         .red = lerpFloat(color1.red, color2.red, theta),
01286         .green = lerpFloat(color1.green, color2.green, theta),
01287         .blue = lerpFloat(color1.blue, color2.blue, theta),
01288         .alpha = lerpFloat(color1.alpha, color2.alpha, theta)};
01289 }
01290
01291 bool floatColorEquals(FloatColor color1, FloatColor color2)
01292 {
01293     return
01294         nearlyEqualFloats(color1.red, color2.red) &&
01295         nearlyEqualFloats(color1.green, color2.green) &&
01296         nearlyEqualFloats(color1.blue, color2.blue) &&
01297         nearlyEqualFloats(color1.alpha, color2.alpha);
01298 }
01299
01300 bool floatColorEmpty(FloatColor color)
01301 {
01302     return
01303         nearlyZeroFloat(color.red) &&
01304         nearlyZeroFloat(color.green) &&
01305         nearlyZeroFloat(color.blue) &&
01306         nearlyZeroFloat(color.alpha);
01307 }
01308
01309 float floatColorGray(FloatColor color)
01310 {
01311     return color.red * 0.29889531f + color.green * 0.58662247f + color.blue * 0.11448222f;
01312 }
01313
01314 Color floatToColor(FloatColor color)
01315 {
01316     return
01317         (Color)roundf(color.red * 255.0f) |
01318         ((Color)roundf(color.green * 255.0f) << 8) |
01319         ((Color)roundf(color.blue * 255.0f) << 16) |
01320         ((Color)roundf(color.alpha * 255.0f) << 24);
01321 }
01322
01323 FloatColorRGB floatColorToRGB(FloatColor color)
01324 {
01325     return (FloatColorRGB){.red = color.red, .green = color.green, .blue = color.blue};
01326 }
01327
01328 // Point constants.
01329
01330 Point const pointZero = {.x = 0, .y = 0};
01331 Point const pointUnity = {.x = 1, .y = 1};
01332 Point const pointAxisX = {.x = 1, .y = 0};
01333 Point const pointAxisY = {.x = 0, .y = 1};
01334
01335 // Point functions.
01336
01337 Point makePoint(int32_t x, int32_t y)
01338 {
01339     return (Point){.x = x, .y = y};
01340 }
01341
01342 Point pointValue(int32_t value)
01343 {
01344     return (Point){.x = value, .y = value};
01345 }
01346
01347 Point pointNegate(Point point)
01348 {
01349     return (Point){.x = -point.x, .y = -point.y};

```



```

01350 }
01351
01352 Point pointAdd(Point point1, Point point2)
01353 {
01354     return (Point){.x = point1.x + point2.x, .y = point1.y + point2.y};
01355 }
01356
01357 Point pointSubtract(Point point1, Point point2)
01358 {
01359     return (Point){.x = point1.x - point2.x, .y = point1.y - point2.y};
01360 }
01361
01362 Point pointMultiply(Point point1, Point point2)
01363 {
01364     return (Point){.x = point1.x * point2.x, .y = point1.y * point2.y};
01365 }
01366
01367 Point pointDivide(Point point1, Point point2)
01368 {
01369     return (Point){.x = point1.x / point2.x, .y = point1.y / point2.y};
01370 }
01371
01372 Point pointRescale(Point point, int32_t theta)
01373 {
01374     return (Point){.x = point.x * theta, .y = point.y * theta};
01375 }
01376
01377 float pointLength(Point point)
01378 {
01379     return hypotf((float)point.x, (float)point.y);
01380 }
01381
01382 float pointDistance(Point point1, Point point2)
01383 {
01384     return pointLength(pointSubtract(point1, point2));
01385 }
01386
01387 float pointAngle(Point point)
01388 {
01389     return atan2f(-(float)point.y, (float)point.x);
01390 }
01391
01392 int32_t pointDot(Point point1, Point point2)
01393 {
01394     return (point1.x * point2.x) + (point1.y * point2.y);
01395 }
01396
01397 int32_t pointCross(Point point1, Point point2)
01398 {
01399     return (point1.x * point2.y) - (point1.y * point2.x);
01400 }
01401
01402 Point pointLerp(Point point1, Point point2, float theta)
01403 {
01404     return (Point){
01405         .x = (int32_t)roundf(lerpFloat((float)point1.x, (float)point2.x, theta)),
01406         .y = (int32_t)roundf(lerpFloat((float)point1.y, (float)point2.y, theta));
01407     }
01408 }
01409 bool pointEquals(Point point1, Point point2)
01410 {
01411     return point1.x == point2.x && point1.y == point2.y;
01412 }
01413
01414 bool pointEmpty(Point point)
01415 {
01416     return point.x == 0 && point.y == 0;
01417 }
01418
01419 // PointF constants.
01420
01421 PointF const pointZeroF = {.x = 0.0f, .y = 0.0f};
01422 PointF const pointUnityF = {.x = 1.0f, .y = 1.0f};
01423 PointF const pointAxisXF = {.x = 1.0f, .y = 0.0f};
01424 PointF const pointAxisYF = {.x = 0.0f, .y = 1.0f};
01425
01426 // PointF functions.
01427
01428 PointF makePointF(float x, float y)
01429 {
01430     return (PointF){.x = x, .y = y};
01431 }
01432
01433 PointF pointF(Point point)
01434 {
01435     return (PointF){.x = (float)point.x, .y = (float)point.y};
01436 }

```

```

01437
01438 PointF pointValueF(float value)
01439 {
01440     return (PointF){.x = value, .y = value};
01441 }
01442
01443 PointF pointNegateF(PointF point)
01444 {
01445     return (PointF){.x = -point.x, .y = -point.y};
01446 }
01447
01448 PointF pointAddF(PointF point1, PointF point2)
01449 {
01450     return (PointF){.x = point1.x + point2.x, .y = point1.y + point2.y};
01451 }
01452
01453 PointF pointSubtractF(PointF point1, PointF point2)
01454 {
01455     return (PointF){.x = point1.x - point2.x, .y = point1.y - point2.y};
01456 }
01457
01458 PointF pointMultiplyF(PointF point1, PointF point2)
01459 {
01460     return (PointF){.x = point1.x * point2.x, .y = point1.y * point2.y};
01461 }
01462
01463 PointF pointDivideF(PointF point1, PointF point2)
01464 {
01465     return (PointF){.x = point1.x / point2.x, .y = point1.y / point2.y};
01466 }
01467
01468 PointF pointRescaleF(PointF point, float theta)
01469 {
01470     return (PointF){.x = point.x * theta, .y = point.y * theta};
01471 }
01472
01473 float pointLengthF(PointF point)
01474 {
01475     return hypotf(point.x, point.y);
01476 }
01477
01478 float pointDistanceF(PointF point1, PointF point2)
01479 {
01480     return pointLengthF(pointSubtractF(point1, point2));
01481 }
01482
01483 float pointAngleF(PointF point)
01484 {
01485     return atan2f(-point.y, point.x);
01486 }
01487
01488 float pointDotF(PointF point1, PointF point2)
01489 {
01490     return (point1.x * point2.x) + (point1.y * point2.y);
01491 }
01492
01493 float pointCrossF(PointF point1, PointF point2)
01494 {
01495     return (point1.x * point2.y) - (point1.y * point2.x);
01496 }
01497
01498 PointF pointNormalizeF(PointF point)
01499 {
01500     return pointRescaleF(point, 1.0f / pointLengthF(point));
01501 }
01502
01503 PointF pointLerpF(PointF point1, PointF point2, float theta)
01504 {
01505     return (PointF){
01506         .x = lerpFloat(point1.x, point2.x, theta),
01507         .y = lerpFloat(point1.y, point2.y, theta);
01508     }
01509 }
01510 bool pointEqualsF(PointF point1, PointF point2)
01511 {
01512     return nearlyEqualFloats(point1.x, point2.x) && nearlyEqualFloats(point1.y, point2.y);
01513 }
01514
01515 bool pointEmptyF(PointF point)
01516 {
01517     return nearlyZeroFloat(point.x) && nearlyZeroFloat(point.y);
01518 }
01519
01520 PointF pointTransformF(PointF point, Matrix3x2 matrix)
01521 {
01522     return (PointF){
01523         .x = (point.x * matrix.data[0][0]) + (point.y * matrix.data[1][0]) + matrix.data[2][0],

```

```

01524     .y = (point.x * matrix.data[0][1]) + (point.y * matrix.data[1][1]) + matrix.data[2][1]);
01525 }
01526
01527 // Vector constants.
01528
01529 Vector const vectorZero = {.x = 0.0f, .y = 0.0f, .z = 0.0f};
01530 Vector const vectorUnity = {.x = 1.0f, .y = 1.0f, .z = 1.0f};
01531 Vector const vectorAxisX = {.x = 1.0f, .y = 0.0f, .z = 0.0f};
01532 Vector const vectorAxisY = {.x = 0.0f, .y = 1.0f, .z = 0.0f};
01533 Vector const vectorAxisZ = {.x = 0.0f, .y = 0.0f, .z = 1.0f};
01534
01535 // Vector functions.
01536
01537 Vector makeVector(float x, float y, float z)
01538 {
01539     return (Vector){.x = x, .y = y, .z = z};
01540 }
01541
01542 Vector vectorValue(float value)
01543 {
01544     return (Vector){.x = value, .y = value, .z = value};
01545 }
01546
01547 Vector vectorNegate(Vector vector)
01548 {
01549     return (Vector){.x = -vector.x, .y = -vector.y, .z = -vector.z};
01550 }
01551
01552 Vector vectorAdd(Vector vector1, Vector vector2)
01553 {
01554     return (Vector){.x = vector1.x + vector2.x, .y = vector1.y + vector2.y, .z = vector1.z + vector2.z};
01555 }
01556
01557 Vector vectorSubtract(Vector vector1, Vector vector2)
01558 {
01559     return (Vector){.x = vector1.x - vector2.x, .y = vector1.y - vector2.y, .z = vector1.z - vector2.z};
01560 }
01561
01562 Vector vectorMultiply(Vector vector1, Vector vector2)
01563 {
01564     return (Vector){.x = vector1.x * vector2.x, .y = vector1.y * vector2.y, .z = vector1.z * vector2.z};
01565 }
01566
01567 Vector vectorDivide(Vector vector1, Vector vector2)
01568 {
01569     return (Vector){.x = vector1.x / vector2.x, .y = vector1.y / vector2.y, .z = vector1.z / vector2.z};
01570 }
01571
01572 Vector vectorRescale(Vector vector, float theta)
01573 {
01574     return (Vector){.x = vector.x * theta, .y = vector.y * theta, .z = vector.z * theta};
01575 }
01576
01577 float vectorDot(Vector vector1, Vector vector2)
01578 {
01579     return (vector1.x * vector2.x) + (vector1.y * vector2.y) + (vector1.z * vector2.z);
01580 }
01581
01582 Vector vectorCross(Vector vector1, Vector vector2)
01583 {
01584     return (Vector){
01585         .x = (vector1.y * vector2.z) - (vector1.z * vector2.y),
01586         .y = (vector1.z * vector2.x) - (vector1.x * vector2.z),
01587         .z = (vector1.x * vector2.y) - (vector1.y * vector2.x)};
01588 }
01589
01590 float vectorLength(Vector vector)
01591 {
01592     return sqrtf(vectorDot(vector, vector));
01593 }
01594
01595 float vectorDistance(Vector vector1, Vector vector2)
01596 {
01597     return vectorLength(vectorSubtract(vector1, vector2));
01598 }
01599
01600 float vectorAngle(Vector vector1, Vector vector2)
01601 {
01602     float const cosValue = vectorDot(vector1, vector2) / (vectorLength(vector1) *
vectorLength(vector2));
01603     return acosf(saturateFloat(cosValue, -1.0f, 1.0f));
01604 }
01605
01606 Vector vectorNormalize(Vector vector)
01607 {
01608     float const magnitude = vectorLength(vector);
01609     return magnitude > vectorEpsilon ? vectorRescale(vector, 1.0f / magnitude) : vector;

```

```

01610 }
01611
01612 Vector vectorParallel(Vector vector, Vector direction)
01613 {
01614     float const dirAmp = vectorLength(direction);
01615     return vectorRescale(direction, (vectorDot(vector, direction) / (dirAmp * dirAmp)));
01616 }
01617
01618 Vector vectorPerpendicular(Vector vector, Vector direction)
01619 {
01620     return vectorSubtract(vector, vectorParallel(vector, direction));
01621 }
01622
01623 Vector vectorReflect(Vector vector, Vector normal)
01624 {
01625     return vectorSubtract(vector, vectorRescale(normal, vectorDot(vector, normal) * 2.0f));
01626 }
01627
01628 Vector vectorLerp(Vector vector1, Vector vector2, float theta)
01629 {
01630     return (Vector){
01631         .x = lerpFloat(vector1.x, vector2.x, theta),
01632         .y = lerpFloat(vector1.y, vector2.y, theta),
01633         .z = lerpFloat(vector1.z, vector2.z, theta);
01634     }
01635 }
01636 bool vectorEquals(Vector vector1, Vector vector2)
01637 {
01638     return
01639         nearlyEqualFloats(vector1.x, vector2.x) &&
01640         nearlyEqualFloats(vector1.y, vector2.y) &&
01641         nearlyEqualFloats(vector1.z, vector2.z);
01642 }
01643
01644 bool vectorEmpty(Vector vector)
01645 {
01646     return nearlyZeroFloat(vector.x) && nearlyZeroFloat(vector.y) && nearlyZeroFloat(vector.z);
01647 }
01648
01649 Vector vectorTransform(Vector vector, float w, Matrix matrix)
01650 {
01651     return (Vector){
01652         (vector.x * matrix.data[0][0]) + (vector.y * matrix.data[1][0]) + (vector.z * matrix.data[2][0]) +
01653         w * matrix.data[3][0],
01654         (vector.x * matrix.data[0][1]) + (vector.y * matrix.data[1][1]) + (vector.z * matrix.data[2][1]) +
01655         w * matrix.data[3][1],
01656         (vector.x * matrix.data[0][2]) + (vector.y * matrix.data[1][2]) + (vector.z * matrix.data[2][2]) +
01657         w * matrix.data[3][2];
01658     }
01659 }
01660 Vector vectorProject(Vector vector, float w, Matrix matrix)
01661 {
01662     float const wf = vector.x * matrix.data[0][3] + vector.y * matrix.data[1][3] +
01663         vector.z * matrix.data[2][3] + w * matrix.data[3][3];
01664     return vectorRescale(vectorTransform(vector, w, matrix), 1.0f / wf);
01665 }
01666
01667 PointF vectorProjectTarget(Vector vector, PointF targetSize, Matrix matrix)
01668 {
01669     float const wf = vector.x * matrix.data[0][3] + vector.y * matrix.data[1][3] +
01670         vector.z * matrix.data[2][3] + matrix.data[3][3];
01671     PointF temp = (PointF){
01672         .x = vector.x * matrix.data[0][0] + vector.y * matrix.data[1][0] + vector.z * matrix.data[2][0] +
01673         matrix.data[3][0],
01674         .y = vector.x * matrix.data[0][1] + vector.y * matrix.data[1][1] + vector.z * matrix.data[2][1] +
01675         matrix.data[3][1];
01676     };
01677     temp = pointRescaleF(temp, 1.0f / wf);
01678     return (PointF){
01679         .x = (temp.x * 0.5f + 0.5f) * targetSize.x,
01680         .y = (0.5f - temp.y * 0.5f) * targetSize.y;
01681     };
01682 }
01683
01684 // Vector4 constants.
01685
01686 Vector4 const vectorZero3D = {.x = 0.0f, .y = 0.0f, .z = 0.0f, .w = 0.0f};
01687 Vector4 const vectorUnity3D = {.x = 1.0f, .y = 1.0f, .z = 1.0f, .w = 1.0f};
01688 Vector4 const vectorAxisX3D = {.x = 1.0f, .y = 0.0f, .z = 0.0f, .w = 0.0f};
01689 Vector4 const vectorAxisY3D = {.x = 0.0f, .y = 1.0f, .z = 0.0f, .w = 0.0f};
01690 Vector4 const vectorAxisZ3D = {.x = 0.0f, .y = 0.0f, .z = 1.0f, .w = 0.0f};
01691 Vector4 const vectorAxisW3D = {.x = 0.0f, .y = 0.0f, .z = 0.0f, .w = 1.0f};
01692
01693 // Vector4 functions.
01694
01695
01696

```

```

01697 Vector4 makeVector3D(float x, float y, float z, float w)
01698 {
01699     return (Vector4){.x = x, .y = y, .z = z, .w = w};
01700 }
01701
01702 Vector4 vectorFromValue3D(float value)
01703 {
01704     return (Vector4){.x = value, .y = value, .z = value, .w = value};
01705 }
01706
01707 Vector4 vectorNegate3D(Vector4 vector)
01708 {
01709     return (Vector4){.x = -vector.x, .y = -vector.y, .z = -vector.z, .w = -vector.w};
01710 }
01711
01712 Vector4 vectorAdd3D(Vector4 vector1, Vector4 vector2)
01713 {
01714     return (Vector4){.x = vector1.x + vector2.x, .y = vector1.y + vector2.y, .z = vector1.z + vector2.z,
01715         .w = vector1.w + vector2.w};
01716 }
01717
01718 Vector4 vectorSubtract3D(Vector4 vector1, Vector4 vector2)
01719 {
01720     return (Vector4){.x = vector1.x - vector2.x, .y = vector1.y - vector2.y, .z = vector1.z - vector2.z,
01721         .w = vector1.w - vector2.w};
01722 }
01723
01724 Vector4 vectorMultiply3D(Vector4 vector1, Vector4 vector2)
01725 {
01726     return (Vector4){.x = vector1.x * vector2.x, .y = vector1.y * vector2.y, .z = vector1.z * vector2.z,
01727         .w = vector1.w * vector2.w};
01728 }
01729
01730 Vector4 vectorDivide3D(Vector4 vector1, Vector4 vector2)
01731 {
01732     return (Vector4){.x = vector1.x / vector2.x, .y = vector1.y / vector2.y, .z = vector1.z / vector2.z,
01733         .w = vector1.w / vector2.w};
01734 }
01735
01736 Vector4 vectorRescale3D(Vector4 vector, float theta)
01737 {
01738     return (Vector4){.x = vector.x * theta, .y = vector.y * theta, .z = vector.z * theta,
01739         .w = vector.w * theta};
01740 }
01741
01742 float vectorDot3D(Vector4 vector1, Vector4 vector2)
01743 {
01744     return (vector1.x * vector2.x) + (vector1.y * vector2.y) + (vector1.z * vector2.z) +
01745         (vector1.w * vector2.w);
01746 }
01747
01748 float vectorLength3D(Vector4 vector)
01749 {
01750     return sqrtf(vectorDot3D(vector, vector));
01751 }
01752
01753 float vectorDistance3D(Vector4 vector1, Vector4 vector2)
01754 {
01755     return vectorLength3D(vectorSubtract3D(vector1, vector2));
01756 }
01757
01758 Vector4 vectorNormalize3D(Vector4 vector)
01759 {
01760     return vectorMultiply3D(vector, vectorFromValue3D(1.0f / vectorLength3D(vector)));
01761 }
01762
01763 Vector4 vectorLerp3D(Vector4 vector1, Vector4 vector2, float theta)
01764 {
01765     return (Vector4){
01766         .x = lerpFloat(vector1.x, vector2.x, theta),
01767         .y = lerpFloat(vector1.y, vector2.y, theta),
01768         .z = lerpFloat(vector1.z, vector2.z, theta),
01769         .w = lerpFloat(vector1.w, vector2.w, theta)};
01770 }
01771
01772 bool vectorEquals3D(Vector4 vector1, Vector4 vector2)
01773 {
01774     return
01775         nearlyEqualFloats(vector1.x, vector2.x) &&
01776         nearlyEqualFloats(vector1.y, vector2.y) &&
01777         nearlyEqualFloats(vector1.z, vector2.z) &&
01778         nearlyEqualFloats(vector1.w, vector2.w);
01779 }
01780
01781 bool vectorEmpty3D(Vector4 vector)
01782 {
01783     return nearlyZeroFloat(vector.x) && nearlyZeroFloat(vector.y) && nearlyZeroFloat(vector.z) &&

```

```

01784     nearlyZeroFloat(vector.w);
01785 }
01786
01787 Vector4 vectorTransform3D(Vector4 vector, Matrix matrix)
01788 {
01789     return (Vector4){
01790         .x = vector.x * matrix.data[0][0] + vector.y * matrix.data[1][0] +
01791             vector.z * matrix.data[2][0] + vector.w * matrix.data[3][0],
01792         .y = vector.x * matrix.data[0][1] + vector.y * matrix.data[1][1] +
01793             vector.z * matrix.data[2][1] + vector.w * matrix.data[3][1],
01794         .z = vector.x * matrix.data[0][2] + vector.y * matrix.data[1][2] +
01795             vector.z * matrix.data[2][2] + vector.w * matrix.data[3][2],
01796         .w = vector.x * matrix.data[0][3] + vector.y * matrix.data[1][3] +
01797             vector.z * matrix.data[2][3] + vector.w * matrix.data[3][3];
01798 }
01799
01800 // Matrix3 constants.
01801
01802 Matrix3x2 const matrixZero3x2 = {{{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}}};
01803
01804 Matrix3x2 const matrixIdentity3x2 = {{{1.0f, 0.0f}, {0.0f, 1.0f}, {0.0f, 0.0f}}};
01805
01806 // Matrix3 functions.
01807
01808 Matrix3x2 matrixMultiply3x2(Matrix3x2 matrix1, Matrix3x2 matrix2)
01809 {
01810     Matrix3x2 res;
01811     res.data[0][0] = matrix1.data[0][0] * matrix2.data[0][0] + matrix1.data[0][1] * matrix2.data[1][0];
01812     res.data[0][1] = matrix1.data[0][0] * matrix2.data[0][1] + matrix1.data[0][1] * matrix2.data[1][1];
01813     res.data[1][0] = matrix1.data[1][0] * matrix2.data[0][0] + matrix1.data[1][1] * matrix2.data[1][0];
01814     res.data[1][1] = matrix1.data[1][0] * matrix2.data[0][1] + matrix1.data[1][1] * matrix2.data[1][1];
01815     res.data[2][0] = matrix1.data[2][0] * matrix2.data[0][0] + matrix1.data[2][1] * matrix2.data[1][0] +
01816         matrix2.data[2][0];
01817     res.data[2][1] = matrix1.data[2][0] * matrix2.data[0][1] + matrix1.data[2][1] * matrix2.data[1][1] +
01818         matrix2.data[2][1];
01819     return res;
01820 }
01821
01822 Matrix3x2 matrixRescale3x2(Matrix3x2 matrix, float theta)
01823 {
01824     Matrix3x2 res;
01825
01826     for (int_fast8_t j = 0; j < 3; ++j)
01827         for (int_fast8_t i = 0; i < 2; ++i)
01828             res.data[j][i] = matrix.data[j][i] * theta;
01829
01830     return res;
01831 }
01832
01833 float matrixDeterminant3x2(Matrix3x2 matrix)
01834 {
01835     return (matrix.data[0][0] * matrix.data[1][1]) - (matrix.data[1][0] * matrix.data[0][1]);
01836 }
01837
01838 Matrix3x2 matrixInverse3x2(Matrix3x2 matrix)
01839 {
01840     float invDet = 1.0f / matrixDeterminant3x2(matrix);
01841     Matrix3x2 res;
01842
01843     res.data[0][0] = matrix.data[1][1] * invDet;
01844     res.data[0][1] = -matrix.data[0][1] * invDet;
01845     res.data[1][0] = -matrix.data[1][0] * invDet;
01846     res.data[1][1] = matrix.data[0][0] * invDet;
01847     res.data[2][0] = (matrix.data[1][0] * matrix.data[2][1] - matrix.data[2][0] * matrix.data[1][1]) *
01848     invDet;
01849     res.data[2][1] = (matrix.data[2][0] * matrix.data[0][1] - matrix.data[0][0] * matrix.data[2][1]) *
01850     invDet;
01851     return res;
01852 }
01853
01854 Matrix3x2 matrixInverseTranspose3x2(Matrix3x2 matrix)
01855 {
01856     float invDet = 1.0f / matrixDeterminant3x2(matrix);
01857     Matrix3x2 res = {0};
01858
01859     res.data[0][0] = matrix.data[1][1] * invDet;
01860     res.data[1][0] = -matrix.data[0][1] * invDet;
01861     res.data[0][1] = -matrix.data[1][0] * invDet;
01862     res.data[1][1] = matrix.data[0][0] * invDet;
01863
01864     return res;
01865 }
01866
01867 Matrix3x2 matrixScale3x2(PointF scale)
01868 {
01869     Matrix3x2 res = matrixIdentity3x2;

```

```

01869
01870     res.data[0][0] = scale.x;
01871     res.data[1][1] = scale.y;
01872
01873     return res;
01874 }
01875
01876 Matrix3x2 matrixScaleBy3x2(PointF center, PointF scale)
01877 {
01878     Matrix3x2 res = matrixIdentity3x2;
01879
01880     res.data[0][0] = scale.x;
01881     res.data[1][1] = scale.y;
01882     res.data[2][0] = center.x * (1.0f - scale.x);
01883     res.data[2][1] = center.y * (1.0f - scale.y);
01884
01885     return res;
01886 }
01887
01888 Matrix3x2 matrixTranslate3x2(PointF offset)
01889 {
01890     Matrix3x2 res = matrixIdentity3x2;
01891
01892     res.data[2][0] = offset.x;
01893     res.data[2][1] = offset.y;
01894
01895     return res;
01896 }
01897
01898 Matrix3x2 matrixRotate3x2(float angle)
01899 {
01900     Matrix3x2 res = matrixIdentity3x2;
01901
01902     float const angleSin = sinf(angle);
01903     float const angleCos = cosf(angle);
01904
01905     res.data[0][0] = angleCos;
01906     res.data[0][1] = angleSin;
01907     res.data[1][0] = -angleSin;
01908     res.data[1][1] = angleCos;
01909
01910     return res;
01911 }
01912
01913 Matrix3x2 matrixRotateBy3x2(PointF center, float angle)
01914 {
01915     float const angleSin = sinf(angle);
01916     float const angleCos = cosf(angle);
01917
01918     Matrix3x2 res;
01919
01920     res.data[0][0] = angleCos;
01921     res.data[0][1] = angleSin;
01922     res.data[1][0] = -angleSin;
01923     res.data[1][1] = angleCos;
01924     res.data[2][0] = center.x * (1.0f - angleCos) + center.y * angleSin;
01925     res.data[2][1] = center.y * (1.0f - angleCos) - center.x * angleSin;
01926
01927     return res;
01928 }
01929
01930 bool matrixEquals3x2(Matrix3x2 matrix1, Matrix3x2 matrix2)
01931 {
01932     for (int_fast8_t j = 0; j < 3; j++)
01933         for (int_fast8_t i = 0; i < 3; i++)
01934             if (!nearlyEqualFloats(matrix1.data[j][i], matrix2.data[j][i]))
01935                 return false;
01936
01937     return true;
01938 }
01939
01940 bool matrixEmpty3x2(Matrix3x2 matrix)
01941 {
01942     for (int_fast8_t j = 0; j < 3; j++)
01943         for (int_fast8_t i = 0; i < 3; i++)
01944             if (!nearlyZeroFloat(matrix.data[j][i]))
01945                 return false;
01946
01947     return true;
01948 }
01949
01950 // Matrix constants.
01951
01952 Matrix const matrixZero = {{
01953     {0.0f, 0.0f, 0.0f, 0.0f},
01954     {0.0f, 0.0f, 0.0f, 0.0f},
01955     {0.0f, 0.0f, 0.0f, 0.0f},

```

```

01956     {0.0f, 0.0f, 0.0f, 0.0f}}};
01957
01958 Matrix const matrixIdentity = {{
01959     {1.0f, 0.0f, 0.0f, 0.0f},
01960     {0.0f, 1.0f, 0.0f, 0.0f},
01961     {0.0f, 0.0f, 1.0f, 0.0f},
01962     {0.0f, 0.0f, 0.0f, 1.0f}}};
01963
01964 // Matrix forward declarations.
01965
01966 float detSub(float a1, float a2, float a3, float b1, float b2, float b3, float c1, float c2, float
c3);
01967
01968 // Matrix functions.
01969
01970 Matrix3x2 matrixGet3x2(Matrix matrix)
01971 {
01972     Matrix3x2 temp;
01973
01974     for (int_fast8_t j = 0; j < 3; ++j)
01975         for (int_fast8_t i = 0; i < 2; ++i)
01976             temp.data[j][i] = matrix.data[j][i];
01977
01978     return temp;
01979 }
01980
01981 Matrix matrixMultiply(Matrix matrix1, Matrix matrix2)
01982 {
01983     Matrix temp;
01984
01985     for (int_fast8_t j = 0; j < 4; ++j)
01986         for (int_fast8_t i = 0; i < 4; ++i)
01987             temp.data[j][i] =
01988                 matrix1.data[j][0] * matrix2.data[0][i] +
01989                 matrix1.data[j][1] * matrix2.data[1][i] +
01990                 matrix1.data[j][2] * matrix2.data[2][i] +
01991                 matrix1.data[j][3] * matrix2.data[3][i];
01992
01993     return temp;
01994 }
01995
01996 Matrix matrixRescale(Matrix matrix, float theta)
01997 {
01998     Matrix temp;
01999
02000     for (int_fast8_t j = 0; j < 4; ++j)
02001         for (int_fast8_t i = 0; i < 4; ++i)
02002             temp.data[j][i] = matrix.data[j][i] * theta;
02003
02004     return temp;
02005 }
02006
02007 float matrixDeterminant(Matrix matrix)
02008 {
02009     return
02010         matrix.data[0][0] * detSub(
02011             matrix.data[1][1], matrix.data[2][1], matrix.data[3][1],
02012             matrix.data[1][2], matrix.data[2][2], matrix.data[3][2],
02013             matrix.data[1][3], matrix.data[2][3], matrix.data[3][3]) -
02014         matrix.data[0][1] * detSub(
02015             matrix.data[1][0], matrix.data[2][0], matrix.data[3][0],
02016             matrix.data[1][2], matrix.data[2][2], matrix.data[3][2],
02017             matrix.data[1][3], matrix.data[2][3], matrix.data[3][3]) +
02018         matrix.data[0][2] * detSub(
02019             matrix.data[1][0], matrix.data[2][0], matrix.data[3][0],
02020             matrix.data[1][1], matrix.data[2][1], matrix.data[3][1],
02021             matrix.data[1][3], matrix.data[2][3], matrix.data[3][3]) -
02022         matrix.data[0][3] * detSub(
02023             matrix.data[1][0], matrix.data[2][0], matrix.data[3][0],
02024             matrix.data[1][1], matrix.data[2][1], matrix.data[3][1],
02025             matrix.data[1][2], matrix.data[2][2], matrix.data[3][2]);
02026 }
02027
02028 Matrix matrixTranspose(Matrix matrix)
02029 {
02030     Matrix temp;
02031
02032     for (int_fast8_t j = 0; j < 4; ++j)
02033         for (int_fast8_t i = 0; i < 4; ++i)
02034             temp.data[i][j] = matrix.data[j][i];
02035
02036     return temp;
02037 }
02038
02039 Matrix matrixAdjoint(Matrix matrix)
02040 {
02041     Matrix temp;

```



```

02042
02043     temp.data[0][0] = detSub(
02044         matrix.data[1][1], matrix.data[2][1], matrix.data[3][1],
02045         matrix.data[1][2], matrix.data[2][2], matrix.data[3][2],
02046         matrix.data[1][3], matrix.data[2][3], matrix.data[3][3]);
02047
02048     temp.data[1][0] = -detSub(
02049         matrix.data[1][0], matrix.data[2][0], matrix.data[3][0],
02050         matrix.data[1][2], matrix.data[2][2], matrix.data[3][2],
02051         matrix.data[1][3], matrix.data[2][3], matrix.data[3][3]);
02052
02053     temp.data[2][0] = detSub(
02054         matrix.data[1][0], matrix.data[2][0], matrix.data[3][0],
02055         matrix.data[1][1], matrix.data[2][1], matrix.data[3][1],
02056         matrix.data[1][3], matrix.data[2][3], matrix.data[3][3]);
02057
02058     temp.data[3][0] = -detSub(
02059         matrix.data[1][0], matrix.data[2][0], matrix.data[3][0],
02060         matrix.data[1][1], matrix.data[2][1], matrix.data[3][1],
02061         matrix.data[1][2], matrix.data[2][2], matrix.data[3][2]);
02062
02063     temp.data[0][1] = -detSub(
02064         matrix.data[0][1], matrix.data[2][1], matrix.data[3][1],
02065         matrix.data[0][2], matrix.data[2][2], matrix.data[3][2],
02066         matrix.data[0][3], matrix.data[2][3], matrix.data[3][3]);
02067
02068     temp.data[1][1] = detSub(
02069         matrix.data[0][0], matrix.data[2][0], matrix.data[3][0],
02070         matrix.data[0][2], matrix.data[2][2], matrix.data[3][2],
02071         matrix.data[0][3], matrix.data[2][3], matrix.data[3][3]);
02072
02073     temp.data[2][1] = -detSub(
02074         matrix.data[0][0], matrix.data[2][0], matrix.data[3][0],
02075         matrix.data[0][1], matrix.data[2][1], matrix.data[3][1],
02076         matrix.data[0][3], matrix.data[2][3], matrix.data[3][3]);
02077
02078     temp.data[3][1] = detSub(
02079         matrix.data[0][0], matrix.data[2][0], matrix.data[3][0],
02080         matrix.data[0][1], matrix.data[2][1], matrix.data[3][1],
02081         matrix.data[0][2], matrix.data[2][2], matrix.data[3][2]);
02082
02083     temp.data[0][2] = detSub(
02084         matrix.data[0][1], matrix.data[1][1], matrix.data[3][1],
02085         matrix.data[0][2], matrix.data[1][2], matrix.data[3][2],
02086         matrix.data[0][3], matrix.data[1][3], matrix.data[3][3]);
02087
02088     temp.data[1][2] = -detSub(
02089         matrix.data[0][0], matrix.data[1][0], matrix.data[3][0],
02090         matrix.data[0][2], matrix.data[1][2], matrix.data[3][2],
02091         matrix.data[0][3], matrix.data[1][3], matrix.data[3][3]);
02092
02093     temp.data[2][2] = detSub(
02094         matrix.data[0][0], matrix.data[1][0], matrix.data[3][0],
02095         matrix.data[0][1], matrix.data[1][1], matrix.data[3][1],
02096         matrix.data[0][3], matrix.data[1][3], matrix.data[3][3]);
02097
02098     temp.data[3][2] = -detSub(
02099         matrix.data[0][0], matrix.data[1][0], matrix.data[3][0],
02100         matrix.data[0][1], matrix.data[1][1], matrix.data[3][1],
02101         matrix.data[0][2], matrix.data[1][2], matrix.data[3][2]);
02102
02103     temp.data[0][3] = -detSub(
02104         matrix.data[0][1], matrix.data[1][1], matrix.data[2][1],
02105         matrix.data[0][2], matrix.data[1][2], matrix.data[2][2],
02106         matrix.data[0][3], matrix.data[1][3], matrix.data[2][3]);
02107
02108     temp.data[1][3] = detSub(
02109         matrix.data[0][0], matrix.data[1][0], matrix.data[2][0],
02110         matrix.data[0][2], matrix.data[1][2], matrix.data[2][2],
02111         matrix.data[0][3], matrix.data[1][3], matrix.data[2][3]);
02112
02113     temp.data[2][3] = -detSub(
02114         matrix.data[0][0], matrix.data[1][0], matrix.data[2][0],
02115         matrix.data[0][1], matrix.data[1][1], matrix.data[2][1],
02116         matrix.data[0][3], matrix.data[1][3], matrix.data[2][3]);
02117
02118     temp.data[3][3] = detSub(
02119         matrix.data[0][0], matrix.data[1][0], matrix.data[2][0],
02120         matrix.data[0][1], matrix.data[1][1], matrix.data[2][1],
02121         matrix.data[0][2], matrix.data[1][2], matrix.data[2][2]);
02122
02123     return temp;
02124 }
02125
02126 Matrix matrixInverse(Matrix matrix)
02127 {
02128     return matrixRescale(matrixAdjoint(matrix), 1.0f / matrixDeterminant(matrix));

```

```

02129 }
02130
02131 Matrix matrixSub3x3(Matrix matrix)
02132 {
02133     Matrix temp = matrixIdentity;
02134
02135     for (int_fast32_t j = 0; j < 3; ++j)
02136         for (int_fast32_t i = 0; i < 3; ++i)
02137             temp.data[j][i] = matrix.data[j][i];
02138
02139     return temp;
02140 }
02141
02142 Vector matrixEyePosition(Matrix matrix)
02143 {
02144     return (Vector){
02145         .x = -matrix.data[0][0] * matrix.data[3][0] - matrix.data[0][1] * matrix.data[3][1] -
02146             matrix.data[0][2] * matrix.data[3][2],
02147         .y = -matrix.data[1][0] * matrix.data[3][0] - matrix.data[1][1] * matrix.data[3][1] -
02148             matrix.data[1][2] * matrix.data[3][2],
02149         .z = -matrix.data[2][0] * matrix.data[3][0] - matrix.data[2][1] * matrix.data[3][1] -
02150             matrix.data[2][2] * matrix.data[3][2]};
02151 }
02152
02153 Vector matrixWorldPosition(Matrix matrix)
02154 {
02155     return (Vector){.x = matrix.data[3][0], .y = matrix.data[3][1], .z = matrix.data[3][2]};
02156 }
02157
02158 Matrix matrixTranslate(Vector offset)
02159 {
02160     Matrix temp = matrixIdentity;
02161
02162     temp.data[3][0] = offset.x;
02163     temp.data[3][1] = offset.y;
02164     temp.data[3][2] = offset.z;
02165
02166     return temp;
02167 }
02168
02169 Matrix matrixScale(Vector scale)
02170 {
02171     Matrix temp = matrixIdentity;
02172
02173     temp.data[0][0] = scale.x;
02174     temp.data[1][1] = scale.y;
02175     temp.data[2][2] = scale.z;
02176
02177     return temp;
02178 }
02179
02180 Matrix matrixRotateX(float angle)
02181 {
02182     Matrix temp = matrixIdentity;
02183
02184     float const sinAngle = sinf(angle);
02185     float const cosAngle = cosf(angle);
02186
02187     temp.data[1][1] = cosAngle;
02188     temp.data[1][2] = sinAngle;
02189     temp.data[2][1] = -sinAngle;
02190     temp.data[2][2] = cosAngle;
02191
02192     return temp;
02193 }
02194
02195 Matrix matrixRotateY(float angle)
02196 {
02197     Matrix temp = matrixIdentity;
02198
02199     float const sinAngle = sinf(angle);
02200     float const cosAngle = cosf(angle);
02201
02202     temp.data[0][0] = cosAngle;
02203     temp.data[0][2] = -sinAngle;
02204     temp.data[2][0] = sinAngle;
02205     temp.data[2][2] = cosAngle;
02206
02207     return temp;
02208 }
02209
02210 Matrix matrixRotateZ(float angle)
02211 {
02212     Matrix temp = matrixIdentity;
02213
02214     float const sinAngle = sinf(angle);
02215     float const cosAngle = cosf(angle);

```

```

02216
02217     temp.data[0][0] = cosAngle;
02218     temp.data[0][1] = sinAngle;
02219     temp.data[1][0] = -sinAngle;
02220     temp.data[1][1] = cosAngle;
02221
02222     return temp;
02223 }
02224
02225 Matrix matrixRotate(Vector axis, float angle)
02226 {
02227     float const sinAngle = sinf(angle);
02228     float const cosAngle = cosf(angle);
02229     float const invCosAngle = 1.0f - cosAngle;
02230
02231     float const xyMul = axis.x * axis.y * invCosAngle;
02232     float const xzMul = axis.x * axis.z * invCosAngle;
02233     float const yzMul = axis.y * axis.z * invCosAngle;
02234
02235     float const xSin = axis.x * sinAngle;
02236     float const ySin = axis.y * sinAngle;
02237     float const zSin = axis.z * sinAngle;
02238
02239     Matrix temp = matrixIdentity;
02240
02241     temp.data[0][0] = axis.x * axis.x * invCosAngle + cosAngle;
02242     temp.data[0][1] = xyMul + zSin;
02243     temp.data[0][2] = xzMul - ySin;
02244     temp.data[1][0] = xyMul - zSin;
02245     temp.data[1][1] = axis.y * axis.y * invCosAngle + cosAngle;
02246     temp.data[1][2] = yzMul + xSin;
02247     temp.data[2][0] = xzMul + ySin;
02248     temp.data[2][1] = yzMul - xSin;
02249     temp.data[2][2] = axis.z * axis.z * invCosAngle + cosAngle;
02250
02251     return temp;
02252 }
02253
02254 Matrix matrixHeadingPitchBank(Vector angles)
02255 {
02256     float const sinHeading = sinf(angles.y);
02257     float const cosHeading = cosf(angles.y);
02258     float const sinPitch = sinf(angles.x);
02259     float const cosPitch = cosf(angles.x);
02260     float const sinBank = sinf(angles.z);
02261     float const cosBank = cosf(angles.z);
02262
02263     Matrix temp = matrixIdentity;
02264
02265     temp.data[0][0] = cosHeading * cosBank + sinHeading * sinPitch * sinBank;
02266     temp.data[0][1] = sinHeading * sinPitch * cosBank - cosHeading * sinBank;
02267     temp.data[0][2] = sinHeading * cosPitch;
02268     temp.data[1][0] = sinBank * cosPitch;
02269     temp.data[1][1] = cosBank * cosPitch;
02270     temp.data[1][2] = -sinPitch;
02271     temp.data[2][0] = cosHeading * sinPitch * sinBank - sinHeading * cosBank;
02272     temp.data[2][1] = sinBank * sinHeading + cosHeading * sinPitch * cosBank;
02273     temp.data[2][2] = cosHeading * cosPitch;
02274
02275     return temp;
02276 }
02277
02278 Matrix matrixYawPitchRoll(Vector angles)
02279 {
02280     float const sinYaw = sinf(angles.y);
02281     float const cosYaw = cosf(angles.y);
02282     float const sinPitch = sinf(angles.x);
02283     float const cosPitch = cosf(angles.x);
02284     float const sinRoll = sinf(angles.z);
02285     float const cosRoll = cosf(angles.z);
02286
02287     Matrix temp = matrixIdentity;
02288
02289     temp.data[0][0] = cosRoll * cosYaw + sinPitch * sinRoll * sinYaw;
02290     temp.data[0][1] = cosYaw * sinPitch * sinRoll - cosRoll * sinYaw;
02291     temp.data[0][2] = -cosPitch * sinRoll;
02292     temp.data[1][0] = cosPitch * sinYaw;
02293     temp.data[1][1] = cosPitch * cosYaw;
02294     temp.data[1][2] = sinPitch;
02295     temp.data[2][0] = cosYaw * sinRoll - cosRoll * sinPitch * sinYaw;
02296     temp.data[2][1] = -cosRoll * cosYaw * sinPitch - sinRoll * sinYaw;
02297     temp.data[2][2] = cosPitch * cosRoll;
02298
02299     return temp;
02300 }
02301
02302 Matrix matrixReflect(Vector axis)

```

```

02303 {
02304     float const xyMul = -2.0f * axis.x * axis.y;
02305     float const xzMul = -2.0f * axis.x * axis.z;
02306     float const yzMul = -2.0f * axis.y * axis.z;
02307
02308     Matrix temp = matrixIdentity;
02309
02310     temp.data[0][0] = 1.0f - (2.0f * axis.x * axis.x);
02311     temp.data[0][1] = xyMul;
02312     temp.data[0][2] = xzMul;
02313     temp.data[1][0] = xyMul;
02314     temp.data[1][1] = 1.0f - (2.0f * axis.y * axis.y);
02315     temp.data[1][2] = yzMul;
02316     temp.data[2][0] = xzMul;
02317     temp.data[2][1] = yzMul;
02318     temp.data[2][2] = 1.0f - (2.0f * axis.z * axis.z);
02319
02320     return temp;
02321 }
02322
02323 Matrix matrixLookAt(Vector origin, Vector target, Vector ceiling)
02324 {
02325     Vector const zAxis = vectorNormalize(vectorSubtract(target, origin));
02326     Vector const xAxis = vectorNormalize(vectorCross(ceiling, zAxis));
02327     Vector const yAxis = vectorNormalize(vectorCross(zAxis, xAxis));
02328
02329     Matrix temp;
02330
02331     temp.data[0][0] = xAxis.x;
02332     temp.data[0][1] = yAxis.x;
02333     temp.data[0][2] = zAxis.x;
02334     temp.data[0][3] = 0.0f;
02335     temp.data[1][0] = xAxis.y;
02336     temp.data[1][1] = yAxis.y;
02337     temp.data[1][2] = zAxis.y;
02338     temp.data[1][3] = 0.0f;
02339     temp.data[2][0] = xAxis.z;
02340     temp.data[2][1] = yAxis.z;
02341     temp.data[2][2] = zAxis.z;
02342     temp.data[2][3] = 0.0f;
02343     temp.data[3][0] = -vectorDot(xAxis, origin);
02344     temp.data[3][1] = -vectorDot(yAxis, origin);
02345     temp.data[3][2] = -vectorDot(zAxis, origin);
02346     temp.data[3][3] = 1.0f;
02347
02348     return temp;
02349 }
02350
02351 Matrix matrixPerspectiveFOVY(float fieldOfView, float aspectRatio, float minRange, float maxRange,
02352     bool negativeDepthRange)
02353 {
02354     float const yScale = 1.0f / tanf(fieldOfView * 0.5f);
02355     float const xScale = yScale / aspectRatio;
02356
02357     Matrix temp = matrixZero;
02358
02359     temp.data[0][0] = xScale;
02360     temp.data[1][1] = yScale;
02361     temp.data[2][3] = 1.0f;
02362
02363     if (negativeDepthRange)
02364     {
02365         temp.data[2][2] = (minRange + maxRange) / (maxRange - minRange);
02366         temp.data[3][2] = -2.0f * minRange * maxRange / (maxRange - minRange);
02367     }
02368     else
02369     {
02370         temp.data[2][2] = minRange / (minRange - maxRange);
02371         temp.data[3][2] = -maxRange * temp.data[2][2];
02372     }
02373     return temp;
02374 }
02375
02376 Matrix matrixPerspectiveFOVX(float fieldOfView, float aspectRatio, float minRange, float maxRange,
02377     bool negativeDepthRange)
02378 {
02379     float const xScale = 1.0f / tanf(fieldOfView * 0.5f);
02380     float const yScale = xScale / aspectRatio;
02381
02382     Matrix temp = matrixZero;
02383
02384     temp.data[0][0] = xScale;
02385     temp.data[1][1] = yScale;
02386     temp.data[2][3] = 1.0f;
02387
02388     if (negativeDepthRange)
02389     {

```

```

02390     temp.data[2][2] = (minRange + maxRange) / (maxRange - minRange);
02391     temp.data[3][2] = -2.0f * minRange * maxRange / (maxRange - minRange);
02392 }
02393 else
02394 {
02395     temp.data[2][2] = minRange / (minRange - maxRange);
02396     temp.data[3][2] = -maxRange * temp.data[2][2];
02397 }
02398 return temp;
02399 }
02400
02401 Matrix matrixPerspectiveVOL(float width, float height, float minRange, float maxRange)
02402 {
02403     float const rangeDiff = maxRange - minRange;
02404     Matrix temp = matrixZero;
02405
02406     temp.data[0][0] = (2.0f * minRange) / width;
02407     temp.data[1][1] = (2.0f * minRange) / height;
02408     temp.data[2][2] = maxRange / rangeDiff;
02409     temp.data[2][3] = 1.0f;
02410     temp.data[3][2] = -minRange * rangeDiff;
02411
02412     return temp;
02413 }
02414
02415 Matrix matrixPerspectiveBDS(float left, float top, float right, float bottom, float minRange,
02416     float maxRange)
02417 {
02418     Matrix temp = matrixZero;
02419
02420     temp.data[0][0] = (2.0f * minRange) / (right - left);
02421     temp.data[1][1] = (2.0f * minRange) / (top - bottom);
02422     temp.data[2][0] = (left + right) / (left - right);
02423     temp.data[2][1] = (top + bottom) / (bottom - top);
02424     temp.data[2][2] = maxRange / (maxRange - minRange);
02425     temp.data[2][3] = 1.0f;
02426     temp.data[3][2] = minRange * maxRange / (minRange - maxRange);
02427
02428     return temp;
02429 }
02430
02431 Matrix matrixOrthogonalVOL(float width, float height, float minRange, float maxRange)
02432 {
02433     Matrix temp = matrixZero;
02434
02435     temp.data[0][0] = 2.0f / width;
02436     temp.data[1][1] = 2.0f / height;
02437     temp.data[2][2] = 1.0f / (minRange - maxRange);
02438     temp.data[3][2] = -maxRange / (minRange - maxRange);
02439     temp.data[3][3] = 1.0f;
02440
02441     return temp;
02442 }
02443
02444 Matrix matrixOrthogonalBDS(float left, float top, float right, float bottom, float minRange, float
02445     maxRange)
02446 {
02447     Matrix temp = matrixZero;
02448
02449     temp.data[0][0] = 2.0f / (right - left);
02450     temp.data[1][1] = 2.0f / (top - bottom);
02451     temp.data[2][2] = 1.0f / (minRange - maxRange);
02452     temp.data[3][0] = (left + right) / (left - right);
02453     temp.data[3][1] = (top + bottom) / (bottom - top);
02454     temp.data[3][2] = -maxRange / (minRange - maxRange);
02455     temp.data[3][3] = 1.0f;
02456
02457     return temp;
02458 }
02459 bool matrixEquals(Matrix matrix1, Matrix matrix2)
02460 {
02461     for (int_fast8_t j = 0; j < 4; j++)
02462         for (int_fast8_t i = 0; i < 4; i++)
02463             if (!nearlyEqualFloats(matrix1.data[j][i], matrix2.data[j][i]))
02464                 return false;
02465
02466     return true;
02467 }
02468
02469 bool matrixEmpty(Matrix matrix)
02470 {
02471     for (int_fast8_t j = 0; j < 4; j++)
02472         for (int_fast8_t i = 0; i < 4; i++)
02473             if (!nearlyZeroFloat(matrix.data[j][i]))
02474                 return false;
02475

```

```

02476     return true;
02477 }
02478
02479 float detSub(float a1, float a2, float a3, float b1, float b2, float b3, float c1, float c2, float c3)
02480 {
02481     return a1 * (b2 * c3 - b3 * c2) - b1 * (a2 * c3 - a3 * c2) + c1 * (a2 * b3 - a3 * b2);
02482 }
02483
02484 // Quaternion constants and functions.
02485
02486 Quaternion const quaternionIdentity = {.x = 0.0f, .y = 0.0f, .z = 0.0f, .w = 1.0f};
02487
02488 Quaternion quaternionMultiply(Quaternion quat1, Quaternion quat2)
02489 {
02490     return (Quaternion){
02491         .x = quat2.w * quat1.x + quat2.x * quat1.w + quat2.z * quat1.y - quat2.y * quat1.z,
02492         .y = quat2.w * quat1.y + quat2.y * quat1.w + quat2.x * quat1.z - quat2.z * quat1.x,
02493         .z = quat2.w * quat1.z + quat2.z * quat1.w + quat2.y * quat1.x - quat2.x * quat1.y,
02494         .w = quat2.w * quat1.w - quat2.x * quat1.x - quat2.y * quat1.y - quat2.z * quat1.z;
02495     }
02496
02497 float quaternionDot(Quaternion quat1, Quaternion quat2)
02498 {
02499     return (quat1.x * quat2.x) + (quat1.y * quat2.y) + (quat1.z * quat2.z) + (quat1.w * quat2.w);
02500 }
02501
02502 float quaternionLength(Quaternion quat)
02503 {
02504     return sqrtf(quaternionDot(quat, quat));
02505 }
02506
02507 float quaternionAngle(Quaternion quat)
02508 {
02509     return acosf(quat.w) * 2.0f;
02510 }
02511
02512 Vector quaternionAxis(Quaternion quat)
02513 {
02514     float const temp1 = 1.0f - quat.w * quat.w;
02515     if (temp1 < vectorEpsilon)
02516         return vectorAxisY;
02517
02518     float const temp2 = 1.0f / sqrtf(temp1);
02519
02520     return (Vector){.x = quat.x * temp2, .y = quat.y * temp2, .z = quat.z * temp2};
02521 }
02522
02523 Quaternion quaternionNormalize(Quaternion quat)
02524 {
02525     float const amp = quaternionLength(quat);
02526     if (!nearlyZeroFloat(amp))
02527     {
02528         float const invAmp = 1.0f / amp;
02529         return (Quaternion){
02530             .x = quat.x * invAmp, .y = quat.y * invAmp, .z = quat.z * invAmp, .w = quat.w * invAmp;
02531         }
02532     }
02533     return quat;
02534 }
02535
02536 Quaternion quaternionConjugate(Quaternion quat)
02537 {
02538     return (Quaternion){.x = -quat.x, .y = -quat.y, .z = -quat.z, .w = quat.w};
02539 }
02540
02541 Quaternion quaternionExponentiate(Quaternion quat, float exponent)
02542 {
02543     if (nearlyEqualFloats(quat.w, 1.0f))
02544         return quat;
02545
02546     float const alpha = acosf(quat.w);
02547     float const newAlpha = alpha * exponent;
02548     float const compMult = sinf(newAlpha) / sinf(alpha);
02549
02550     return (Quaternion){
02551         .x = quat.x * compMult, .y = quat.y * compMult, .z = quat.z * compMult, .w = cosf(newAlpha);
02552     }
02553
02554 Quaternion quaternionSlerp(Quaternion quat1, Quaternion quat2, float theta)
02555 {
02556     if (theta <= 0.0f)
02557         return quat1;
02558     else if (theta >= 1.0f)
02559         return quat2;
02560
02561     float cosOmega = quaternionDot(quat1, quat2);
02562     Quaternion tempQuat = quat2;

```

```

02563
02564     if (cosOmega < 0.0f)
02565     {
02566         tempQuat.x = -tempQuat.x;
02567         tempQuat.y = -tempQuat.y;
02568         tempQuat.z = -tempQuat.z;
02569         tempQuat.w = -tempQuat.w;
02570         cosOmega = -cosOmega;
02571     }
02572
02573     float kappa1, kappa2;
02574
02575     if (cosOmega > 1.0f - vectorEpsilon)
02576     {
02577         kappa1 = 1.0f - theta;
02578         kappa2 = theta;
02579     }
02580     else
02581     {
02582         float const sinOmega = sqrtf(1.0f - cosOmega * cosOmega);
02583         float const omega = atan2f(sinOmega, cosOmega);
02584         float const oneOverSinOmega = 1.0f / sinOmega;
02585
02586         kappa1 = sinf((1.0f - theta) * omega) * oneOverSinOmega;
02587         kappa2 = sinf(theta * omega) * oneOverSinOmega;
02588     }
02589
02590     return (Quaternion){
02591         .x = kappa1 * quat1.x + kappa2 * tempQuat.x,
02592         .y = kappa1 * quat1.y + kappa2 * tempQuat.y,
02593         .z = kappa1 * quat1.z + kappa2 * tempQuat.z,
02594         .w = kappa1 * quat1.w + kappa2 * tempQuat.w};
02595 }
02596
02597 Matrix quaternionMatrix(Quaternion quat)
02598 {
02599     Matrix temp = matrixIdentity;
02600
02601     temp.data[0][0] = 1.0f - 2.0f * quat.y * quat.y - 2.0f * quat.z * quat.z;
02602     temp.data[0][1] = 2.0f * quat.x * quat.y + 2.0f * quat.w * quat.z;
02603     temp.data[0][2] = 2.0f * quat.x * quat.z - 2.0f * quat.w * quat.y;
02604     temp.data[1][0] = 2.0f * quat.x * quat.y - 2.0f * quat.w * quat.z;
02605     temp.data[1][1] = 1.0f - 2.0f * quat.x * quat.x - 2.0f * quat.z * quat.z;
02606     temp.data[1][2] = 2.0f * quat.y * quat.z + 2.0f * quat.w * quat.x;
02607     temp.data[2][0] = 2.0f * quat.x * quat.z + 2.0f * quat.w * quat.y;
02608     temp.data[2][1] = 2.0f * quat.y * quat.z - 2.0f * quat.w * quat.x;
02609     temp.data[2][2] = 1.0f - 2.0f * quat.x * quat.x - 2.0f * quat.y * quat.y;
02610
02611     return temp;
02612 }
02613
02614 Quaternion matrixQuaternion(Matrix matrix)
02615 {
02616     Quaternion const temp = {
02617         matrix.data[0][0] - matrix.data[1][1] - matrix.data[2][2],
02618         matrix.data[1][1] - matrix.data[0][0] - matrix.data[2][2],
02619         matrix.data[2][2] - matrix.data[0][0] - matrix.data[1][1],
02620         matrix.data[0][0] + matrix.data[1][1] + matrix.data[2][2]};
02621
02622     int_fast8_t index = 0;
02623     float maxValue = temp.w;
02624
02625     if (temp.x > maxValue)
02626     {
02627         maxValue = temp.x;
02628         index = 1;
02629     }
02630     else if (temp.y > maxValue)
02631     {
02632         maxValue = temp.y;
02633         index = 2;
02634     }
02635     else if (temp.z > maxValue)
02636     {
02637         maxValue = temp.z;
02638         index = 3;
02639     }
02640
02641     float const highValue = sqrtf(maxValue + 1.0f) * 0.5f;
02642     float const multValue = 0.25f / highValue;
02643
02644     Quaternion res;
02645
02646     switch (index)
02647     {
02648     case 1:
02649         res.x = highValue;

```

```

02650     res.y = (matrix.data[0][1] + matrix.data[1][0]) * multValue;
02651     res.z = (matrix.data[2][0] + matrix.data[0][2]) * multValue;
02652     res.w = (matrix.data[1][2] - matrix.data[2][1]) * multValue;
02653     break;
02654 case 2:
02655     res.x = (matrix.data[0][1] + matrix.data[1][0]) * multValue;
02656     res.y = highValue;
02657     res.z = (matrix.data[1][2] + matrix.data[2][1]) * multValue;
02658     res.w = (matrix.data[2][0] - matrix.data[0][2]) * multValue;
02659     break;
02660 case 3:
02661     res.x = (matrix.data[2][0] + matrix.data[0][2]) * multValue;
02662     res.y = (matrix.data[1][2] + matrix.data[2][1]) * multValue;
02663     res.z = highValue;
02664     res.w = (matrix.data[0][1] - matrix.data[1][0]) * multValue;
02665     break;
02666 default:
02667     res.x = (matrix.data[1][2] - matrix.data[2][1]) * multValue;
02668     res.y = (matrix.data[2][0] - matrix.data[0][2]) * multValue;
02669     res.z = (matrix.data[0][1] - matrix.data[1][0]) * multValue;
02670     res.w = highValue;
02671     break;
02672 }
02673 return res;
02674 }
02675
02676 Quaternion quaternionRotateX(float angle)
02677 {
02678     float const newAngle = angle * 0.5f;
02679     return (Quaternion){.x = sinf(newAngle), .y = 0.0f, .z = 0.0f, .w = cosf(newAngle)};
02680 }
02681
02682 Quaternion quaternionRotateY(float angle)
02683 {
02684     float const newAngle = angle * 0.5f;
02685     return (Quaternion){.x = 0.0f, .y = sinf(newAngle), .z = 0.0f, .w = cosf(newAngle)};
02686 }
02687
02688 Quaternion quaternionRotateZ(float angle)
02689 {
02690     float const newAngle = angle * 0.5f;
02691     return (Quaternion){.x = 0.0f, .y = 0.0f, .z = sinf(newAngle), .w = cosf(newAngle)};
02692 }
02693
02694 Quaternion quaternionRotate(Vector axis, float angle)
02695 {
02696     float const amp = vectorLength(axis);
02697     if (!nearlyZeroFloat(amp))
02698     {
02699         float const newAngle = angle * 0.5f;
02700         float const sinDivAmp = sinf(newAngle) / amp;
02701
02702         return (Quaternion){
02703             .x = axis.x * sinDivAmp, .y = axis.y * sinDivAmp, .z = axis.z * sinDivAmp, .w = cosf(newAngle)};
02704     }
02705     else
02706         return quaternionIdentity;
02707 }
02708
02709 Quaternion quaternionRotateObjectToInertial(Vector angles)
02710 {
02711     float const sinHeading = sinf(angles.y);
02712     float const cosHeading = cosf(angles.y);
02713     float const sinPitch = sinf(angles.x);
02714     float const cosPitch = cosf(angles.x);
02715     float const sinBank = sinf(angles.z);
02716     float const cosBank = cosf(angles.z);
02717
02718     return (Quaternion){
02719         .x = cosHeading * sinPitch * cosBank + sinHeading * cosPitch * sinBank,
02720         .y = -cosHeading * sinPitch * sinBank + sinHeading * cosPitch * cosBank,
02721         .z = -sinHeading * sinPitch * cosBank + cosHeading * cosPitch * sinBank,
02722         .w = cosHeading * cosPitch * cosBank + sinHeading * sinPitch * sinBank};
02723 }
02724
02725 Quaternion quaternionRotateInertialToObject(Vector angles)
02726 {
02727     float const sinHeading = sinf(angles.y);
02728     float const cosHeading = cosf(angles.y);
02729     float const sinPitch = sinf(angles.x);
02730     float const cosPitch = cosf(angles.x);
02731     float const sinBank = sinf(angles.z);
02732     float const cosBank = cosf(angles.z);
02733
02734     return (Quaternion){
02735         .x = -cosHeading * sinPitch * cosBank - sinHeading * cosPitch * sinBank,
02736         .y = cosHeading * sinPitch * sinBank - sinHeading * cosPitch * cosBank,

```



```

02737     .z = sinHeading * sinPitch * cosBank - cosHeading * cosPitch * sinBank,
02738     .w = cosHeading * cosPitch * cosBank + sinHeading * sinPitch * sinBank};
02739 }
02740
02741 // Rect constants and functions.
02742
02743 Rect const rectZero = {.left = 0, .top = 0, .width = 0, .height = 0};
02744
02745 Rect makeRect(int32_t left, int32_t top, int32_t width, int32_t height)
02746 {
02747     return (Rect){.left = left, .top = top, .width = width, .height = height};
02748 }
02749
02750 Rect makeBounds(int32_t left, int32_t top, int32_t right, int32_t bottom)
02751 {
02752     return (Rect){.left = left, .top = top, .width = right - left, .height = bottom - top};
02753 }
02754
02755 Point rectGetSize(Rect rect)
02756 {
02757     return (Point){.x = rect.width, .y = rect.height};
02758 }
02759
02760 Rect rectSetSize(Rect rect, Point size)
02761 {
02762     rect.width = size.x;
02763     rect.height = size.y;
02764     return rect;
02765 }
02766
02767 int32_t rectGetRight(Rect rect)
02768 {
02769     return rect.left + rect.width;
02770 }
02771
02772 Rect rectSetRight(Rect rect, int32_t right)
02773 {
02774     rect.width = right - rect.left;
02775     return rect;
02776 }
02777
02778 int32_t rectGetBottom(Rect rect)
02779 {
02780     return rect.top + rect.height;
02781 }
02782
02783 Rect rectSetBottom(Rect rect, int32_t bottom)
02784 {
02785     rect.height = bottom - rect.top;
02786     return rect;
02787 }
02788
02789 Point rectGetTopLeft(Rect rect)
02790 {
02791     return (Point){.x = rect.left, .y = rect.top};
02792 }
02793
02794 Rect rectSetTopLeft(Rect rect, Point topLeft)
02795 {
02796     rect.left = topLeft.x;
02797     rect.top = topLeft.y;
02798     return rect;
02799 }
02800
02801 Point rectGetBottomRight(Rect rect)
02802 {
02803     return (Point){.x = rect.left + rect.width, .y = rect.top + rect.height};
02804 }
02805
02806 Rect rectSetBottomRight(Rect rect, Point bottomRight)
02807 {
02808     rect.width = bottomRight.x - rect.left;
02809     rect.height = bottomRight.y - rect.top;
02810     return rect;
02811 }
02812
02813 Point rectGetTopRight(Rect rect)
02814 {
02815     return (Point){.x = rectGetRight(rect), .y = rect.top};
02816 }
02817
02818 Point rectGetBottomLeft(Rect rect)
02819 {
02820     return (Point){rect.left, rectGetBottom(rect)};
02821 }
02822
02823 bool rectEquals(Rect rect1, Rect rect2)

```

```

02824 {
02825     return rect1.left == rect2.left && rect1.top == rect2.top && rect1.width == rect2.width &&
02826         rect1.height == rect2.height;
02827 }
02828
02829 bool rectEmpty(Rect rect)
02830 {
02831     return rect.width <= 0 || rect.height <= 0;
02832 }
02833
02834 bool pointInRect(Point point, Rect rect)
02835 {
02836     return point.x >= rect.left && point.x < rectGetRight(rect) && point.y >= rect.top &&
02837         point.y < rectGetBottom(rect);
02838 }
02839
02840 bool rectInRect(Rect rect, Rect largerRect)
02841 {
02842     return largerRect.left >= rect.left && rectGetRight(largerRect) <= rectGetRight(rect) &&
02843         largerRect.top >= rect.top && rectGetBottom(largerRect) <= rectGetBottom(rect);
02844 }
02845
02846 bool rectOverlap(Rect rect1, Rect rect2)
02847 {
02848     return rect2.left < rectGetRight(rect1) && rectGetRight(rect2) > rect1.left &&
02849         rect2.top < rectGetBottom(rect1) && rectGetBottom(rect2) > rect1.top;
02850 }
02851
02852 Rect rectIntersect(Rect rect1, Rect rect2)
02853 {
02854     if (rectOverlap(rect1, rect2))
02855     {
02856         return makeBounds(
02857             MAX(rect1.left, rect2.left),
02858             MAX(rect1.top, rect2.top),
02859             MIN(rectGetRight(rect1), rectGetRight(rect2)),
02860             MIN(rectGetBottom(rect1), rectGetBottom(rect2)));
02861     }
02862     else
02863         return rectZero;
02864 }
02865
02866 Rect rectJoin(Rect rect1, Rect rect2)
02867 {
02868     return makeBounds(
02869         MIN(rect1.left, rect2.left),
02870         MIN(rect1.top, rect2.top),
02871         MAX(rectGetRight(rect1), rectGetRight(rect2)),
02872         MAX(rectGetBottom(rect1), rectGetBottom(rect2)));
02873 }
02874
02875 Rect rectOffset(Rect rect, Point delta)
02876 {
02877     return (Rect){
02878         .left = rect.left + delta.x, .top = rect.top + delta.y, .width = rect.width, .height =
02879         rect.height};
02880 }
02881
02882 Rect rectInflate(Rect rect, Point delta)
02883 {
02884     return (Rect){.left = rect.left - delta.x, .top = rect.top - delta.y,
02885         .width = rect.width + delta.x * 2, .height = rect.height + delta.y * 2};
02886 }
02887 // RectF constants and functions.
02888
02889 RectF const rectZeroF = {.left = 0.0f, .top = 0.0f, .width = 0.0f, .height = 0.0f};
02890
02891 RectF makeRectF(float left, float top, float width, float height)
02892 {
02893     return (RectF){.left = left, .top = top, .width = width, .height = height};
02894 }
02895
02896 RectF makeBoundsF(float left, float top, float right, float bottom)
02897 {
02898     return (RectF){.left = left, .top = top, .width = right - left, .height = bottom - top};
02899 }
02900
02901 RectF rectF(Rect rect)
02902 {
02903     return (RectF){.left = (float)rect.left, .top = (float)rect.top,
02904         .width = (float)rect.width, .height = (float)rect.height};
02905 }
02906
02907 Rect rectToIntRectF(RectF rect)
02908 {
02909     return (Rect){.left = (int32_t)roundf(rect.left), .top = (int32_t)roundf(rect.top),

```

```

02910     .width = (int32_t)roundf(rect.width), .height = (int32_t)roundf(rect.height));
02911 }
02912
02913 PointF rectGetSizeF(RectF rect)
02914 {
02915     return (PointF){.x = rect.width, .y = rect.height};
02916 }
02917
02918 RectF rectSetSizeF(RectF rect, PointF size)
02919 {
02920     rect.width = size.x;
02921     rect.height = size.y;
02922     return rect;
02923 }
02924
02925 float rectGetRightF(RectF rect)
02926 {
02927     return rect.left + rect.width;
02928 }
02929
02930 RectF rectSetRightF(RectF rect, float right)
02931 {
02932     rect.width = right - rect.left;
02933     return rect;
02934 }
02935
02936 float rectGetBottomF(RectF rect)
02937 {
02938     return rect.top + rect.height;
02939 }
02940
02941 RectF rectSetBottomF(RectF rect, float bottom)
02942 {
02943     rect.height = bottom - rect.top;
02944     return rect;
02945 }
02946
02947 PointF rectGetTopLeftF(RectF rect)
02948 {
02949     return (PointF){.x = rect.left, .y = rect.top};
02950 }
02951
02952 RectF rectSetTopLeftF(RectF rect, PointF topLeft)
02953 {
02954     rect.left = topLeft.x;
02955     rect.top = topLeft.y;
02956     return rect;
02957 }
02958
02959 PointF rectGetBottomRightF(RectF rect)
02960 {
02961     return (PointF){.x = rect.left + rect.width, .y = rect.top + rect.height};
02962 }
02963
02964 RectF rectSetBottomRightF(RectF rect, PointF bottomRight)
02965 {
02966     rect.width = bottomRight.x - rect.left;
02967     rect.height = bottomRight.y - rect.top;
02968     return rect;
02969 }
02970
02971 PointF rectGetTopRightF(RectF rect)
02972 {
02973     return (PointF){.x = rectGetRightF(rect), .y = rect.top};
02974 }
02975
02976 PointF rectGetBottomLeftF(RectF rect)
02977 {
02978     return (PointF){rect.left, rectGetBottomF(rect)};
02979 }
02980
02981 bool rectEqualsF(RectF rect1, RectF rect2)
02982 {
02983     return nearlyEqualFloats(rect1.left, rect2.left) && nearlyEqualFloats(rect1.top, rect2.top) &&
02984         nearlyEqualFloats(rect1.width, rect2.width) && nearlyEqualFloats(rect1.height, rect2.height);
02985 }
02986
02987 bool rectEmptyF(RectF rect)
02988 {
02989     return rect.width < vectorEpsilon || rect.height < vectorEpsilon;
02990 }
02991
02992 bool pointInRectF(PointF point, RectF rect)
02993 {
02994     return point.x >= rect.left && point.x < rectGetRightF(rect) && point.y >= rect.top &&
02995         point.y < rectGetBottomF(rect);
02996 }

```

```

02997
02998 bool rectInRectF(RectF rect, RectF largerRect)
02999 {
03000     return largerRect.left >= rect.left && rectGetRightF(largerRect) <= rectGetRightF(rect) &&
03001         largerRect.top >= rect.top && rectGetBottomF(largerRect) <= rectGetBottomF(rect);
03002 }
03003
03004 bool rectOverlapF(RectF rect1, RectF rect2)
03005 {
03006     return rect2.left < rectGetRightF(rect1) && rectGetRightF(rect2) > rect1.left &&
03007         rect2.top < rectGetBottomF(rect1) && rectGetBottomF(rect2) > rect1.top;
03008 }
03009
03010 RectF rectIntersectF(RectF rect1, RectF rect2)
03011 {
03012     if (rectOverlapF(rect1, rect2))
03013     {
03014         return makeBoundsF(
03015             MAX(rect1.left, rect2.left),
03016             MAX(rect1.top, rect2.top),
03017             MIN(rectGetRightF(rect1), rectGetRightF(rect2)),
03018             MIN(rectGetBottomF(rect1), rectGetBottomF(rect2)));
03019     }
03020     else
03021         return rectZeroF;
03022 }
03023
03024 RectF rectJoinF(RectF rect1, RectF rect2)
03025 {
03026     return makeBoundsF(
03027         MIN(rect1.left, rect2.left),
03028         MIN(rect1.top, rect2.top),
03029         MAX(rectGetRightF(rect1), rectGetRightF(rect2)),
03030         MAX(rectGetBottomF(rect1), rectGetBottomF(rect2)));
03031 }
03032
03033 RectF rectOffsetF(RectF rect, PointF delta)
03034 {
03035     return (RectF){
03036         .left = rect.left + delta.x, .top = rect.top + delta.y, .width = rect.width, .height =
03037         rect.height};
03038 }
03039
03040 RectF rectInflateF(RectF rect, PointF delta)
03041 {
03042     return (RectF){.left = rect.left - delta.x, .top = rect.top - delta.y,
03043         .width = rect.width + delta.x * 2.0f, .height = rect.height + delta.y * 2.0f};
03044 }
03045 // Quad constants and functions.
03046
03047 Quad const quadZero = {{0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}, {0.0f, 0.0f}};
03048 Quad const quadUnity = {{0.0f, 0.0f}, {1.0f, 0.0f}, {1.0f, 1.0f}, {0.0f, 1.0f}};
03049
03050 Quad makeQuadWith(PointF topLeft, PointF topRight, PointF bottomRight, PointF bottomLeft)
03051 {
03052     return (Quad){
03053         .topLeft = topLeft, .topRight = topRight, .bottomRight = bottomRight, .bottomLeft = bottomLeft};
03054 }
03055
03056 Quad makeQuad(float left, float top, float width, float height)
03057 {
03058     return (Quad){
03059         .topLeft = (PointF){.x = left, .y = top},
03060         .topRight = (PointF){.x = left + width, .y = top},
03061         .bottomRight = (PointF){.x = left + width, .y = top + height},
03062         .bottomLeft = (PointF){.x = left, .y = top + height}};
03063 }
03064
03065 Quad makeQuadScaled(float left, float top, float width, float height, float scale, bool centered)
03066 {
03067     if (nearlyEqualFloats(scale, 1.0f))
03068         return makeQuad(left, top, width, height);
03069
03070     if (centered)
03071     {
03072         float const newWidth = width * scale;
03073         float const newHeight = height * scale;
03074
03075         return makeQuad(
03076             left + (width - newWidth) * 0.5f, top + (height - newHeight) * 0.5f, newWidth, newHeight);
03077     }
03078     else
03079         return makeQuad(left, top, width * scale, height * scale);
03080 }
03081
03082 Quad makeQuadRotatedAt(PointF origin, PointF size, PointF center, float angle, float scale)

```

```

03083 {
03084     float const sinAngle = sinf(angle);
03085     float const cosAngle = cosf(angle);
03086
03087     bool const scaled = !nearlyEqualFloats(scale, 1.0f);
03088
03089     Quad rect = makeQuad(-center.x, -center.y, size.x, size.y);
03090     PointF* destPos = &rect.topLeft;
03091
03092     for (int_fast8_t i = 0; i < 4; ++i)
03093     {
03094         if (scaled)
03095             *destPos = pointRescaleF(*destPos, scale);
03096
03097         PointF const newPos = makePointF(
03098             destPos->x * cosAngle - destPos->y * sinAngle,
03099             destPos->y * cosAngle + destPos->x * sinAngle);
03100
03101         *destPos = pointAddF(newPos, origin);
03102         destPos++;
03103     }
03104     return rect;
03105 }
03106
03107 Quad makeQuadRotated(PointF origin, PointF size, float angle, float scale)
03108 {
03109     return makeQuadRotatedAt(origin, size, pointRescaleF(size, 0.5f), angle, scale);
03110 }
03111
03112 Quad makeQuadRotatedTL(PointF topLeft, PointF size, PointF center, float angle, float scale)
03113 {
03114     return quadOffset(makeQuadRotatedAt(topLeft, size, center, angle, scale), center);
03115 }
03116
03117 Quad makeQuadSkewedHoriz(PointF origin, PointF size, float angle)
03118 {
03119     float const offset = cosf(angle) * size.y * 0.5f;
03120
03121     Quad quad = makeQuad(origin.x - size.x * 0.5f, origin.y - size.y * 0.5f, size.x, size.y);
03122     quad.topLeft.x -= offset;
03123     quad.topRight.x -= offset;
03124     quad.bottomRight.x += offset;
03125     quad.bottomLeft.x += offset;
03126
03127     return quad;
03128 }
03129
03130 Quad makeQuadSkewedVert(PointF origin, PointF size, float angle)
03131 {
03132     float const offset = cosf(angle) * size.x * 0.5f;
03133
03134     Quad quad = makeQuad(origin.x - size.x * 0.5f, origin.y - size.y * 0.5f, size.x, size.y);
03135     quad.topLeft.y -= offset;
03136     quad.bottomLeft.y -= offset;
03137     quad.topRight.y += offset;
03138     quad.bottomRight.y += offset;
03139
03140     return quad;
03141 }
03142
03143 Quad quadFromRect(Rect rect)
03144 {
03145     return (Quad){
03146         .topLeft = pointF(rectGetTopLeft(rect)),
03147         .topRight = pointF(rectGetTopRight(rect)),
03148         .bottomRight = pointF(rectGetBottomRight(rect)),
03149         .bottomLeft = pointF(rectGetBottomLeft(rect))};
03150 }
03151
03152 Quad quadFromRectF(RectF rect)
03153 {
03154     return (Quad){
03155         .topLeft = rectGetTopLeftF(rect),
03156         .topRight = rectGetTopRightF(rect),
03157         .bottomRight = rectGetBottomRightF(rect),
03158         .bottomLeft = rectGetBottomLeftF(rect)};
03159 }
03160
03161 Quad quadScale(Quad quad, float scale, bool centered)
03162 {
03163     if (nearlyEqualFloats(scale, 1.0f))
03164         return quad;
03165
03166     if (centered)
03167     {
03168         PointF const center = pointRescaleF(pointAddF(pointAddF(quad.topLeft, quad.topRight),
03169             pointAddF(quad.bottomRight, quad.bottomLeft)), 0.25f);

```

```

03170
03171     return (Quad){
03172         .topLeft = pointLerpF(center, quad.topLeft, scale),
03173         .topRight = pointLerpF(center, quad.topRight, scale),
03174         .bottomRight = pointLerpF(center, quad.bottomRight, scale),
03175         .bottomLeft = pointLerpF(center, quad.bottomLeft, scale);
03176     }
03177     else
03178     return (Quad){
03179         .topLeft = pointRescaleF(quad.topLeft, scale),
03180         .topRight = pointRescaleF(quad.topRight, scale),
03181         .bottomRight = pointRescaleF(quad.bottomRight, scale),
03182         .bottomLeft = pointRescaleF(quad.bottomLeft, scale);
03183     }
03184
03185 Quad quadMirror(Quad quad)
03186 {
03187     return (Quad){.topLeft = quad.topRight, .topRight = quad.topLeft,
03188         .bottomRight = quad.bottomLeft, .bottomLeft = quad.bottomRight};
03189 }
03190
03191 Quad quadFlip(Quad quad)
03192 {
03193     return (Quad){.topLeft = quad.bottomLeft, .topRight = quad.bottomRight,
03194         .bottomRight = quad.topRight, .bottomLeft = quad.topLeft};
03195 }
03196
03197 Quad quadTransform(Quad quad, Matrix3x2 matrix)
03198 {
03199     return (Quad){
03200         .topLeft = pointTransformF(quad.topLeft, matrix),
03201         .topRight = pointTransformF(quad.topRight, matrix),
03202         .bottomRight = pointTransformF(quad.bottomRight, matrix),
03203         .bottomLeft = pointTransformF(quad.bottomLeft, matrix);
03204 }
03205
03206 Quad quadOffset(Quad quad, PointF delta)
03207 {
03208     return (Quad){
03209         .topLeft = pointAddF(quad.topLeft, delta),
03210         .topRight = pointAddF(quad.topRight, delta),
03211         .bottomRight = pointAddF(quad.bottomRight, delta),
03212         .bottomLeft = pointAddF(quad.bottomLeft, delta);
03213 }
03214
03215 // Service functions.
03216
03217 bool nearlyEqualFloats(float left, float right)
03218 {
03219     return
03220         left == right ||
03221         fabsf(left - right) <= fmaxf(fabsf(left), fabsf(right)) * vectorEpsilon;
03222 }
03223
03224 bool nearlyZeroFloat(float value)
03225 {
03226     return fabsf(value) <= vectorEpsilon;
03227 }
03228
03229 float saturateFloat(float value, float min, float max)
03230 {
03231     float const tempValue = value > max ? max : value;
03232     return tempValue < min ? min : tempValue;
03233 }
03234
03235 float lerpFloat(float value1, float value2, float theta)
03236 {
03237     return value1 + (value2 - value1) * theta;
03238 }
03239
03240 #endif

```

Index

- access
 - ComputeBindTextureFormat, [45](#)
- ActorCamera_t
 - pxt, [17](#)
- ActorCameraCommand
 - Afterwarp.h, [163](#)
- ActorCameraCreate
 - Afterwarp.h, [163](#)
- ActorCameraDestroy
 - Afterwarp.h, [163](#)
- ActorCameraGetCeiling
 - Afterwarp.h, [163](#)
- ActorCameraGetConstraints
 - Afterwarp.h, [163](#)
- ActorCameraGetDistance
 - Afterwarp.h, [163](#)
- ActorCameraGetForward
 - Afterwarp.h, [164](#)
- ActorCameraGetPosition
 - Afterwarp.h, [164](#)
- ActorCameraGetQuaternion
 - Afterwarp.h, [164](#)
- ActorCameraGetRight
 - Afterwarp.h, [164](#)
- ActorCameraGetRotation
 - Afterwarp.h, [164](#)
- ActorCameraGetView
 - Afterwarp.h, [164](#)
- ActorCameraSetConstraints
 - Afterwarp.h, [165](#)
- ActorCameraSetDistance
 - Afterwarp.h, [165](#)
- ActorCameraSetPosition
 - Afterwarp.h, [165](#)
- ActorCameraSetQuaternion
 - Afterwarp.h, [165](#)
- ActorCameraSetRotation
 - Afterwarp.h, [165](#)
- ActorCameraZoom
 - Afterwarp.h, [165](#)
- ActorCameraZoomOrtho
 - Afterwarp.h, [166](#)
- AddColors
 - Afterwarp.h, [166](#)
- address
 - SamplerState, [89](#)
- addressU
 - CanvasSamplerState, [42](#)
- addressV
 - CanvasSamplerState, [42](#)
- Afterwarp Framework, [1](#)
- Afterwarp.h, [121](#)
 - ActorCameraCommand, [163](#)
 - ActorCameraCreate, [163](#)
 - ActorCameraDestroy, [163](#)
 - ActorCameraGetCeiling, [163](#)
 - ActorCameraGetConstraints, [163](#)
 - ActorCameraGetDistance, [163](#)
 - ActorCameraGetForward, [164](#)
 - ActorCameraGetPosition, [164](#)
 - ActorCameraGetQuaternion, [164](#)
 - ActorCameraGetRight, [164](#)
 - ActorCameraGetRotation, [164](#)
 - ActorCameraGetView, [164](#)
 - ActorCameraSetConstraints, [165](#)
 - ActorCameraSetDistance, [165](#)
 - ActorCameraSetPosition, [165](#)
 - ActorCameraSetQuaternion, [165](#)
 - ActorCameraSetRotation, [165](#)
 - ActorCameraZoom, [165](#)
 - ActorCameraZoomOrtho, [166](#)
 - AddColors, [166](#)
 - ApplicationCaptureMouseInput, [166](#)
 - ApplicationConvertPortableKey, [166](#)
 - ApplicationCreate, [166](#)
 - ApplicationDestroy, [167](#)
 - ApplicationExecute, [167](#)
 - ApplicationFileChooserDialog, [167](#)
 - ApplicationGetClientRect, [167](#)
 - ApplicationGetCursor, [167](#)
 - ApplicationGetExecutablePath, [167](#)
 - ApplicationGetIconTitle, [168](#)
 - ApplicationGetMinimalSize, [168](#)
 - ApplicationGetTitle, [168](#)
 - ApplicationGetWindowHandle, [168](#)
 - ApplicationGetWindowRect, [168](#)
 - ApplicationGetWindowScale, [169](#)
 - ApplicationGetWindowState, [169](#)
 - ApplicationInvalidate, [169](#)
 - ApplicationMouseInputCaptured, [169](#)
 - ApplicationReadTextFromClipboard, [169](#)
 - ApplicationReleaseMouseInput, [169](#)
 - ApplicationSetClientSize, [170](#)
 - ApplicationSetCursor, [170](#)
 - ApplicationSetEvents, [170](#)
 - ApplicationSetIcons, [170](#)
 - ApplicationSetIconsFromFiles, [170](#)
 - ApplicationSetIconTitle, [170](#)

- ApplicationSetMinimalSize, 171
- ApplicationSetTitle, 171
- ApplicationSetWindowRect, 171
- ApplicationSetWindowState, 171
- ApplicationTerminate, 171
- ApplicationTranslateVirtualKey, 171
- ApplicationWriteTextToClipboard, 172
- AverageColors, 172
- AverageFourColors, 172
- AverageSixColors, 172
- BiasTransform, 172
- BlendColors, 173
- BlendFourColors, 173
- BufferCopy, 173
- BufferCreate, 173
- BufferDestroy, 173
- BufferGetAccessType, 174
- BufferGetDataType, 174
- BufferGetDevice, 174
- BufferGetFormat, 174
- BufferGetPitch, 174
- BufferGetPlatformHandle, 174
- BufferGetSize, 174
- BufferRetrieve, 175
- BufferUpdate, 175
- CanvasArc, 175
- CanvasArcGrad, 175
- CanvasBegin, 176
- CanvasBufferClear, 176
- CanvasBufferClearAndShrink, 176
- CanvasBufferCreate, 176
- CanvasBufferDestroy, 176
- CanvasBufferGetColors, 176
- CanvasBufferGetExtents, 177
- CanvasBufferGetIndexCount, 177
- CanvasBufferGetIndices, 177
- CanvasBufferGetVertexCount, 177
- CanvasBufferGetVertices, 177
- CanvasBufferSetIndexCount, 177
- CanvasBufferSetVertexCount, 177
- CanvasCreate, 178
- CanvasDestroy, 178
- CanvasDrawBuffer, 178
- CanvasEllipse, 178
- CanvasEnd, 178
- CanvasFillRect, 178
- CanvasFillRoundRect, 179
- CanvasFillRoundRectBottom, 179
- CanvasFillRoundRectTop, 179
- CanvasFillRoundRectTopInverse, 179
- CanvasFlush, 180
- CanvasFrameRect, 180
- CanvasFrameRoundRect, 180
- CanvasGetAttributes, 180
- CanvasGetBatchCount, 180
- CanvasGetClipRect, 181
- CanvasGetContextState, 181
- CanvasGetDevice, 181
- CanvasGetSamplerState, 181
- CanvasGetSignedDistanceField, 181
- CanvasGetTransform, 181
- CanvasGetViewProjection, 182
- CanvasGetWorld, 182
- CanvasHexagon, 182
- CanvasHexagonGrad, 182
- CanvasHighlight, 182
- CanvasLine, 183
- CanvasLineCircle, 183
- CanvasLineEllipse, 183
- CanvasLineHexagon, 183
- CanvasLineHexagonGrad, 184
- CanvasLineQuad, 184
- CanvasLines, 184
- CanvasLineTriangle, 184
- CanvasPixel, 185
- CanvasPixels, 185
- CanvasQuad, 185
- CanvasQuadImage, 185
- CanvasRectWithHole, 185
- CanvasReset, 186
- CanvasResetSamplerState, 186
- CanvasRibbon, 186
- CanvasRibbonGrad, 186
- CanvasRibbonTri, 186
- CanvasSetAttributes, 187
- CanvasSetClipRect, 187
- CanvasSetContextState, 187
- CanvasSetSamplerState, 187
- CanvasSetSignedDistanceField, 187
- CanvasSetTransform, 188
- CanvasSetViewProjection, 188
- CanvasSetWorld, 188
- CanvasTape, 188
- CanvasTapeGrad, 188
- CanvasTapeTri, 189
- CanvasTexturedQuad, 189
- CanvasTexturedQuadRegion, 189
- CanvasTexturedRoundRect, 189
- CanvasTexturedRoundRectRegion, 190
- CanvasTexturedTriangle, 190
- CanvasTexturedTriangleRegion, 190
- CanvasTexturedTriangles, 191
- CanvasThickLine, 191
- CanvasThickLineCircle, 191
- CanvasThickLineEllipse, 191
- CanvasThickLineHexagon, 192
- CanvasThickLineHexagonGrad, 192
- CanvasThickLineQuad, 192
- CanvasThickLineTriangle, 192
- CanvasTriangle, 193
- CanvasTriangles, 193
- CatmullRom, 193
- ColorDitheringCreate, 193
- ColorDitheringDestroy, 194
- ColorDitheringExecute, 194
- ColorDitheringExecuteTo, 194

- ColorDitheringGetDevice, [194](#)
- ColorDitheringGetFormat, [194](#)
- ColorDitheringSetFormat, [194](#)
- ColorToGray, [195](#)
- ColorToGray16, [195](#)
- ColorToGrayF, [195](#)
- ComposeColors, [195](#)
- ComputeProgramBegin, [195](#)
- ComputeProgramBindBuffer, [195](#)
- ComputeProgramBindTexture, [196](#)
- ComputeProgramCommit, [196](#)
- ComputeProgramCreate, [196](#)
- ComputeProgramDestroy, [196](#)
- ComputeProgramDispatch, [196](#)
- ComputeProgramEnd, [197](#)
- ComputeProgramGetDevice, [197](#)
- ComputeProgramResetBindings, [197](#)
- ComputeProgramUnbindBuffer, [197](#)
- ComputeProgramUnbindTexture, [197](#)
- Cubic, [197](#)
- DeviceClear, [198](#)
- DeviceCreate, [198](#)
- DeviceCreateWrapped, [198](#)
- DeviceDestroy, [198](#)
- DeviceGetAttributes, [198](#)
- DeviceGetBehavior, [199](#)
- DeviceGetLegacyBits, [199](#)
- DeviceGetPlatform, [199](#)
- DeviceGetPlatformDevice, [199](#)
- DeviceGetRenderingState, [199](#)
- DeviceGetScissor, [199](#)
- DeviceGetTechFeatureVersion, [199](#)
- DeviceGetTechnology, [200](#)
- DeviceGetTechVersion, [200](#)
- DeviceGetViewport, [200](#)
- DeviceMemoryBarrier, [200](#)
- DeviceResetCache, [200](#)
- DeviceSetRenderingState, [200](#)
- DeviceSetScissor, [201](#)
- DeviceSetViewport, [201](#)
- DisplaceRB, [201](#)
- GainTransform, [201](#)
- GaussianBlurCreate, [201](#)
- GaussianBlurDestroy, [201](#)
- GaussianBlurGetChroma, [202](#)
- GaussianBlurGetDevice, [202](#)
- GaussianBlurGetFixedSamples, [202](#)
- GaussianBlurGetHardwareFiltering, [202](#)
- GaussianBlurGetSamples, [202](#)
- GaussianBlurGetSigma, [202](#)
- GaussianBlurSetChroma, [202](#)
- GaussianBlurSetHardwareFiltering, [203](#)
- GaussianBlurSetSamples, [203](#)
- GaussianBlurSetSigma, [203](#)
- GaussianBlurUpdate, [203](#)
- GaussianBlurUpdateAt, [203](#)
- GetColorAlpha, [204](#)
- GetColorAlphaF, [204](#)
- GetSystemTicks, [204](#)
- GrapherArrow, [204](#)
- GrapherBegin, [204](#)
- GrapherBoundingBox, [205](#)
- GrapherCreate, [205](#)
- GrapherDestroy, [205](#)
- GrapherDottedLine, [205](#)
- GrapherEnd, [205](#)
- GrapherFlush, [206](#)
- GrapherGetBatchCount, [206](#)
- GrapherGetDevice, [206](#)
- GrapherGetTransform, [206](#)
- GrapherLine, [206](#)
- GrapherLines, [206](#)
- GrapherPoint, [207](#)
- GrapherPoints, [207](#)
- GrapherReset, [207](#)
- GrapherSetTransform, [207](#)
- Hermite, [207](#)
- ImageAtlasClearRegions, [208](#)
- ImageAtlasClearTextures, [208](#)
- ImageAtlasCreate, [208](#)
- ImageAtlasCreateRegion, [208](#)
- ImageAtlasCreateTexture, [208](#)
- ImageAtlasDestroy, [209](#)
- ImageAtlasGetDevice, [209](#)
- ImageAtlasMakeRegions, [209](#)
- ImageAtlasPackRegion, [209](#)
- ImageAtlasPackSurface, [209](#)
- ImageAtlasRegion, [210](#)
- ImageAtlasRegionCount, [210](#)
- ImageAtlasRemoveRegion, [210](#)
- ImageAtlasRemoveTexture, [210](#)
- ImageAtlasTexture, [210](#)
- ImageAtlasTextureCount, [210](#)
- InvertColor, [211](#)
- KawaseBlurCreate, [211](#)
- KawaseBlurDestroy, [211](#)
- KawaseBlurGetDevice, [211](#)
- KawaseBlurGetOffset, [211](#)
- KawaseBlurGetPasses, [211](#)
- KawaseBlurSetOffset, [211](#)
- KawaseBlurSetPasses, [212](#)
- KawaseBlurSinglePass, [212](#)
- KawaseBlurUpdate, [212](#)
- Lerp, [212](#)
- LibraryGetVersion, [212](#)
- LibrarySerialCode, [213](#)
- MakeColor, [213](#)
- MakeColorAlpha, [213](#)
- MakeColorAlphaF, [213](#)
- MakeColorF, [213](#)
- MakeColorGray, [213](#)
- MakeColorGrayF, [214](#)
- MakeColorRGB, [214](#)
- MakeColorRGBF, [214](#)
- MakeColorWithGray, [214](#)
- MakeColorWithGrayF, [214](#)

- MeshBoundsTagOffset, 215
- MeshBoundsToMatrixModel, 215
- MeshBoundsToMatrixVolume, 215
- MeshBoundsToMatrixVolumeTag, 215
- MeshBufferCalculateBounds, 215
- MeshBufferCalculateFlatNormals, 216
- MeshBufferCalculateNormals, 216
- MeshBufferCalculateNormalsWeld, 216
- MeshBufferCentralize, 216
- MeshBufferClear, 216
- MeshBufferCombine, 217
- MeshBufferConeSimple, 217
- MeshBufferCreate, 217
- MeshBufferCreateModel, 217
- MeshBufferCube, 217
- MeshBufferCubeMinimal, 218
- MeshBufferCubeRound, 218
- MeshBufferCylinder, 218
- MeshBufferDestroy, 219
- MeshBufferDisc, 219
- MeshBufferEliminateUnusedVertices, 219
- MeshBufferExtrusion, 219
- MeshBufferFrustumVolume, 220
- MeshBufferGeosphere, 220
- MeshBufferGetIndexCount, 220
- MeshBufferGetIndices, 220
- MeshBufferGetTransform, 220
- MeshBufferGetVertexCount, 221
- MeshBufferGetVertices, 221
- MeshBufferInvertIndexOrder, 221
- MeshBufferInvertNormals, 221
- MeshBufferJoinDuplicateVertices, 221
- MeshBufferLoadFromFile, 221
- MeshBufferLoadFromFileEx, 222
- MeshBufferPlane, 222
- MeshBufferSaveToFile, 222
- MeshBufferSaveToFileEx, 223
- MeshBufferSetIndexCount, 223
- MeshBufferSetTransform, 223
- MeshBufferSetVertexCount, 223
- MeshBufferSuperEllipse, 223
- MeshBufferSupertoroid, 224
- MeshBufferTorus, 224
- MeshBufferTorusKnot, 225
- MeshBufferTransferIndexElements, 225
- MeshBufferTransferIndices, 225
- MeshBufferTransferVertexElements, 226
- MeshBufferTransferVertices, 226
- MeshBufferTransformVertices, 226
- MeshBufferVoxelize, 226
- MeshMetaTagGetBounds, 227
- MeshMetaTagGetName, 227
- MeshMetaTagGetParent, 227
- MeshMetaTagGetPortionCount, 227
- MeshMetaTagGetType, 227
- MeshMetaTagPortionAdd, 227
- MeshMetaTagPortionErase, 228
- MeshMetaTagPortionGet, 228
- MeshMetaTagPortionsClear, 228
- MeshMetaTagPortionsCopy, 228
- MeshMetaTagsAdd, 228
- MeshMetaTagsClear, 228
- MeshMetaTagsCopy, 229
- MeshMetaTagsCount, 229
- MeshMetaTagsCreate, 229
- MeshMetaTagsDestroy, 229
- MeshMetaTagsErase, 229
- MeshMetaTagsGetByIndex, 229
- MeshMetaTagsGetByName, 230
- MeshMetaTagsTakeAway, 230
- MeshModelCreate, 230
- MeshModelDestroy, 230
- MeshModelDismantle, 230
- MeshModelDraw, 230
- MeshModelDrawInstances, 231
- MeshModelGetChannel, 231
- MeshModelGetIndexBuffer, 231
- MeshModelGetIndexCount, 231
- MeshModelGetVertexBuffer, 231
- MeshModelGetVertexCount, 232
- MeshModelRenderable, 232
- MeshModelTakeAway, 232
- MeshVoxelComputeParameters, 232
- MeshVoxelCreate, 232
- MeshVoxelDestroy, 232
- MeshVoxelExtents, 233
- MeshVoxelIntersect, 233
- MeshVoxelLoadFromFile, 233
- MeshVoxelLoadFromFileInMemory, 233
- MeshVoxelSaveToFile, 233
- MeshVoxelTakeAway, 234
- MeshVoxelVisualize, 234
- MultiplyColors, 234
- ObjectMaterialsAdd, 234
- ObjectMaterialsClear, 234
- ObjectMaterialsCreate, 234
- ObjectMaterialsDestroy, 235
- ObjectMaterialsErase, 235
- ObjectMaterialsGetCount, 235
- ObjectMaterialsGetElement, 235
- ObjectModelConnectLatches, 235
- ObjectModelConnectLatchesByName, 235
- ObjectModelGetAABB, 236
- ObjectModelGetAlignments, 236
- ObjectModelGetAttributes, 236
- ObjectModelGetChild, 236
- ObjectModelGetChildCount, 236
- ObjectModelGetColor, 236
- ObjectModelGetConnectedLatches, 237
- ObjectModelGetDepthBias, 237
- ObjectModelGetDescription, 237
- ObjectModelGetHighlight, 237
- ObjectModelGetID, 237
- ObjectModelGetLatchTransform, 237
- ObjectModelGetLatchTransformByName, 238
- ObjectModelGetLatchWaypointCouple, 238

- ObjectModelGetLatchWaypointCoupleMatrix, [238](#)
- ObjectModelGetLayers, [238](#)
- ObjectModelGetMaterial, [238](#)
- ObjectModelGetMesh, [239](#)
- ObjectModelGetMeshName, [239](#)
- ObjectModelGetName, [239](#)
- ObjectModelGetNext, [239](#)
- ObjectModelGetOrderIndex, [239](#)
- ObjectModelGetOwner, [239](#)
- ObjectModelGetParent, [240](#)
- ObjectModelGetPayload, [240](#)
- ObjectModelGetPosition, [240](#)
- ObjectModelGetSize, [240](#)
- ObjectModelGetTransform, [240](#)
- ObjectModelGetVoxel, [240](#)
- ObjectModelGetWaypointDistance, [241](#)
- ObjectModelInvalidate, [241](#)
- ObjectModelsAdd, [241](#)
- ObjectModelsAddWithID, [241](#)
- ObjectModelsClear, [241](#)
- ObjectModelsClearViews, [241](#)
- ObjectModelsCreate, [242](#)
- ObjectModelsCreateView, [242](#)
- ObjectModelsDestroy, [242](#)
- ObjectModelsErase, [242](#)
- ObjectModelsEraseView, [242](#)
- ObjectModelSetAlignments, [242](#)
- ObjectModelSetAttributes, [243](#)
- ObjectModelSetColor, [243](#)
- ObjectModelSetDepthBias, [243](#)
- ObjectModelSetDescription, [243](#)
- ObjectModelSetHighlight, [243](#)
- ObjectModelSetLayers, [243](#)
- ObjectModelSetMaterial, [244](#)
- ObjectModelSetMesh, [244](#)
- ObjectModelSetMeshName, [244](#)
- ObjectModelSetName, [244](#)
- ObjectModelSetOrderIndex, [244](#)
- ObjectModelSetParent, [244](#)
- ObjectModelSetSize, [245](#)
- ObjectModelSetTransform, [245](#)
- ObjectModelSetVoxel, [245](#)
- ObjectModelsGetFirst, [245](#)
- ObjectModelsGetMeshes, [245](#)
- ObjectModelsGetObjectByID, [245](#)
- ObjectModelsGetObjectByName, [246](#)
- ObjectModelsGetObjectCount, [246](#)
- ObjectModelsPayload, [246](#)
- ObjectModelViewAutoDraw, [246](#)
- ObjectModelViewGetIntersectedObjects, [246](#)
- ObjectModelViewGetIntersectedRays, [246](#)
- ObjectModelViewGetLayers, [247](#)
- ObjectModelViewGetObject, [247](#)
- ObjectModelViewGetObjectCount, [247](#)
- ObjectModelViewGetObjectsNotCulled, [247](#)
- ObjectModelViewGetOwner, [247](#)
- ObjectModelViewGetProjection, [247](#)
- ObjectModelViewGetView, [248](#)
- ObjectModelViewGetViewProjection, [248](#)
- ObjectModelViewInvalidate, [248](#)
- ObjectModelViewSelect, [248](#)
- ObjectModelViewSelectAny, [248](#)
- ObjectModelViewSetLayers, [248](#)
- ObjectModelViewSetProjection, [249](#)
- ObjectModelViewSetView, [249](#)
- ObjectModelViewSort, [249](#)
- ObjectModelViewSortWith, [249](#)
- ObjectModelViewUpdate, [249](#)
- ObjectModelViewUpdateNeeded, [249](#)
- OceanSimulationCreate, [250](#)
- OceanSimulationDestroy, [250](#)
- OceanSimulationGetAttributes, [250](#)
- OceanSimulationGetDevice, [250](#)
- OceanSimulationGetMaterial, [250](#)
- OceanSimulationGetPlane, [250](#)
- OceanSimulationGetProjection, [250](#)
- OceanSimulationGetSamplerShadow, [251](#)
- OceanSimulationGetScale, [251](#)
- OceanSimulationGetSections, [251](#)
- OceanSimulationGetView, [251](#)
- OceanSimulationGetViewDistance, [251](#)
- OceanSimulationGetWavesParameters, [251](#)
- OceanSimulationGetWavesTexture, [252](#)
- OceanSimulationRender, [252](#)
- OceanSimulationSetAttributes, [252](#)
- OceanSimulationSetMaterial, [252](#)
- OceanSimulationSetPlane, [252](#)
- OceanSimulationSetProjection, [252](#)
- OceanSimulationSetScale, [253](#)
- OceanSimulationSetSections, [253](#)
- OceanSimulationSetView, [253](#)
- OceanSimulationSetViewDistance, [253](#)
- OceanSimulationSetWavesParameters, [253](#)
- OceanSimulationUpdate, [253](#)
- PathBrokerCreate, [254](#)
- PathBrokerDestroy, [254](#)
- PathBrokerFill, [254](#)
- PathBrokerReset, [254](#)
- PathBrokerStroke, [254](#)
- PixelFormatBits, [254](#)
- PixelFormatConvert, [255](#)
- PixelFormatConvertArray, [255](#)
- PixelFormatValid, [255](#)
- PremultiplyAlpha, [255](#)
- ProgramBegin, [255](#)
- ProgramBind, [255](#)
- ProgramCommit, [256](#)
- ProgramCreate, [256](#)
- ProgramDestroy, [256](#)
- ProgramDraw, [256](#)
- ProgramDrawIndexed, [257](#)
- ProgramDrawInstances, [257](#)
- ProgramDrawInstancesIndexed, [257](#)
- ProgramEnd, [257](#)
- ProgramGetDevice, [257](#)
- ProgramResetBindings, [258](#)

- ProgramResetCache, [258](#)
- ProgramSetPatchVertices, [258](#)
- ProgramUnbind, [258](#)
- ProgramUpdateByIndex, [258](#)
- ProgramUpdateByName, [258](#)
- RandomSequenceGenerate, [259](#)
- RandomSequenceGenerate64, [259](#)
- RandomSequenceGenerateDouble, [259](#)
- RandomSequenceGenerateFloat, [259](#)
- RandomSequenceGenerateGaussian, [259](#)
- RandomSequenceGenerateRanged, [259](#)
- RandomSequenceInit, [260](#)
- RandomSequenceInitBySeed, [260](#)
- RayCreate, [260](#)
- RayIntersectCubeVolume, [260](#)
- RayIntersectPlane, [260](#)
- RayIntersectTriangle, [261](#)
- SamplerBind, [261](#)
- SamplerCreate, [261](#)
- SamplerDestroy, [261](#)
- SamplerGetDevice, [261](#)
- SamplerGetState, [262](#)
- SamplerSetState, [262](#)
- SamplerUnbind, [262](#)
- SceneBegin, [262](#)
- SceneCreateDepthsNormals, [262](#)
- SceneCreateModeling, [262](#)
- SceneDestroy, [263](#)
- SceneEnd, [263](#)
- SceneGetActiveVertexElements, [263](#)
- SceneGetAttributes, [263](#)
- SceneGetDevice, [263](#)
- SceneGetInstancesCount, [263](#)
- SceneGetLights, [263](#)
- SceneGetMaterial, [264](#)
- SceneGetParallaxMappingParameters, [264](#)
- SceneGetProgram, [264](#)
- SceneGetProjection, [264](#)
- SceneGetSampler, [264](#)
- SceneGetShadowCastingAtlas, [264](#)
- SceneGetTexture, [265](#)
- SceneGetTextureCabinet, [265](#)
- SceneGetToneMappingCoefficients, [265](#)
- SceneGetVertexElementCount, [265](#)
- SceneGetVertexElements, [265](#)
- SceneGetView, [265](#)
- SceneGetWorld, [266](#)
- SceneInstances, [266](#)
- SceneLightsAdd, [266](#)
- SceneLightsClear, [266](#)
- SceneLightsCreate, [266](#)
- SceneLightsDestroy, [266](#)
- SceneLightsErase, [267](#)
- SceneLightsExecute, [267](#)
- SceneLightsGetClusters, [267](#)
- SceneLightsGetClusterSize, [267](#)
- SceneLightsGetCount, [267](#)
- SceneLightsGetCullingMode, [267](#)
- SceneLightsGetDepthSlices, [268](#)
- SceneLightsGetDevice, [268](#)
- SceneLightsGetElement, [268](#)
- SceneLightsGetIndices, [268](#)
- SceneLightsGetViewSize, [268](#)
- SceneLightsRenderDebug, [268](#)
- SceneLightsSetClusterSize, [269](#)
- SceneLightsSetCullingMode, [269](#)
- SceneLightsSetDepthSlices, [269](#)
- SceneLightsSetViewSize, [269](#)
- SceneMeshAutoDraw, [269](#)
- SceneMeshAutoDrawSliced, [270](#)
- SceneMeshesAdd, [270](#)
- SceneMeshesAddFromBuffer, [270](#)
- SceneMeshesAddFromFile, [271](#)
- SceneMeshesClear, [271](#)
- SceneMeshesCreate, [271](#)
- SceneMeshesDestroy, [271](#)
- SceneMeshesErase, [272](#)
- SceneMeshesGetCount, [272](#)
- SceneMeshesGetDevice, [272](#)
- SceneMeshesGetMeshByIndex, [272](#)
- SceneMeshesGetMeshByName, [272](#)
- SceneMeshesPayload, [272](#)
- SceneMeshesSlice, [273](#)
- SceneMeshGetBounds, [273](#)
- SceneMeshGetLatches, [273](#)
- SceneMeshGetMaterials, [273](#)
- SceneMeshGetModel, [273](#)
- SceneMeshGetName, [273](#)
- SceneMeshGetPayload, [274](#)
- SceneMeshGetScale, [274](#)
- SceneMeshGetSize, [274](#)
- SceneMeshGetSlice, [274](#)
- SceneMeshGetTags, [274](#)
- SceneMeshGetVertexElementsIndex, [274](#)
- SceneMeshGetVoxel, [275](#)
- SceneMeshLatchesAdd, [275](#)
- SceneMeshLatchesClear, [275](#)
- SceneMeshLatchesCopy, [275](#)
- SceneMeshLatchesCreate, [275](#)
- SceneMeshLatchesDestroy, [275](#)
- SceneMeshLatchesErase, [276](#)
- SceneMeshLatchesGetCount, [276](#)
- SceneMeshLatchesGetLatch, [276](#)
- SceneMeshLatchesGetLatchIndex, [276](#)
- SceneMeshLatchesGetWaypointCouple, [276](#)
- SceneMeshLatchesGetWaypointDistance, [276](#)
- SceneMeshLatchesInvalidateWaypoints, [277](#)
- SceneMeshLatchesLoadFromFile, [277](#)
- SceneMeshLatchesLoadFromFileInMemory, [277](#)
- SceneMeshLatchesSaveToFile, [277](#)
- SceneMeshLatchesSetLatch, [277](#)
- SceneMeshLatchesTakeAway, [277](#)
- SceneMeshMaterialAddRange, [278](#)
- SceneMeshMaterialClearRanges, [278](#)
- SceneMeshMaterialCommit, [278](#)
- SceneMeshMaterialCopy, [278](#)

- SceneMeshMaterialGetName, [278](#)
- SceneMeshMaterialGetRange, [278](#)
- SceneMeshMaterialGetRangeCount, [279](#)
- SceneMeshMaterialGetShading, [279](#)
- SceneMeshMaterialGetTexture, [279](#)
- SceneMeshMaterialReleaseTextures, [279](#)
- SceneMeshMaterialsAdd, [279](#)
- SceneMeshMaterialsClear, [279](#)
- SceneMeshMaterialsCommit, [280](#)
- SceneMeshMaterialsCopy, [280](#)
- SceneMeshMaterialsCreate, [280](#)
- SceneMeshMaterialsDestroy, [280](#)
- SceneMeshMaterialsErase, [280](#)
- SceneMeshMaterialSetRange, [280](#)
- SceneMeshMaterialSetShading, [281](#)
- SceneMeshMaterialSetTexture, [281](#)
- SceneMeshMaterialsGetCount, [281](#)
- SceneMeshMaterialsGetMaterial, [281](#)
- SceneMeshMaterialsGetName, [281](#)
- SceneMeshMaterialsSetName, [281](#)
- SceneMeshMaterialsTakeAway, [282](#)
- SceneMeshMaterialsTexturing, [282](#)
- SceneMeshSetSlice, [282](#)
- SceneMeshSetVertexElementsIndex, [282](#)
- ScenePrepare, [282](#)
- SceneRendering, [282](#)
- SceneResetCache, [283](#)
- SceneSetActiveVertexElements, [283](#)
- SceneSetAttributes, [283](#)
- SceneSetLights, [283](#)
- SceneSetMaterial, [283](#)
- SceneSetParallaxMappingParameters, [283](#)
- SceneSetProjection, [284](#)
- SceneSetShadowCastingAtlas, [284](#)
- SceneSetTexture, [284](#)
- SceneSetTextureCabinet, [284](#)
- SceneSetToneMappingCoefficients, [284](#)
- SceneSetVertexElements, [284](#)
- SceneSetVertexElementsFromTextModeller, [285](#)
- SceneSetView, [285](#)
- SceneSetWorld, [285](#)
- SelectionHighlightBegin, [285](#)
- SelectionHighlightClear, [285](#)
- SelectionHighlightCreate, [285](#)
- SelectionHighlightDestroy, [286](#)
- SelectionHighlightEnd, [286](#)
- SelectionHighlightFilter, [286](#)
- SelectionHighlightGetDepthStencil, [286](#)
- SelectionHighlightGetDevice, [286](#)
- SelectionHighlightGetGrayscale, [286](#)
- SelectionHighlightGetParameters, [287](#)
- SelectionHighlightGetSamples, [287](#)
- SelectionHighlightGetSize, [287](#)
- SelectionHighlightGetTexture, [287](#)
- SelectionHighlightGetTextureCoordinates, [287](#)
- SelectionHighlightRendering, [287](#)
- SelectionHighlightSetParameters, [288](#)
- SelectionHighlightSetSize, [288](#)
- ShadowCasterBegin, [288](#)
- ShadowCasterClear, [288](#)
- ShadowCasterEnd, [288](#)
- ShadowCasterFilter, [288](#)
- ShadowCasterGetAtlas, [288](#)
- ShadowCasterGetDevice, [289](#)
- ShadowCasterGetPosition, [289](#)
- ShadowCasterGetSize, [289](#)
- ShadowCasterGetTexture, [289](#)
- ShadowCasterGetViewProjection, [289](#)
- ShadowCasterRendering, [289](#)
- ShadowCasterSetViewProjection, [290](#)
- ShadowCastingAtlasAdd, [290](#)
- ShadowCastingAtlasBorderFill, [290](#)
- ShadowCastingAtlasClear, [290](#)
- ShadowCastingAtlasCreate, [290](#)
- ShadowCastingAtlasDestroy, [290](#)
- ShadowCastingAtlasErase, [291](#)
- ShadowCastingAtlasGetCaster, [291](#)
- ShadowCastingAtlasGetCount, [291](#)
- ShadowCastingAtlasGetDevice, [291](#)
- ShadowCastingAtlasGetPadding, [291](#)
- ShadowCastingAtlasGetParameters, [291](#)
- ShadowCastingAtlasGetSamples, [292](#)
- ShadowCastingAtlasGetSize, [292](#)
- ShadowCastingAtlasGetTechnique, [292](#)
- ShadowCastingAtlasGetTexture, [292](#)
- ShadowCastingAtlasSetParameters, [292](#)
- SineAccelerate, [292](#)
- SineCycle, [293](#)
- SineDecelerate, [293](#)
- SineTransform, [293](#)
- SineTwoCycle, [293](#)
- SmootherStep, [293](#)
- SmoothStep, [293](#)
- SpatialFogCreate, [294](#)
- SpatialFogDestroy, [294](#)
- SpatialFogExecute, [294](#)
- SpatialFogExecuteGlassy, [294](#)
- SpatialFogGetDepthDistance, [294](#)
- SpatialFogGetDevice, [294](#)
- SpatialFogGetFormula, [295](#)
- SpatialFogGetParameters, [295](#)
- SpatialFogGetProjection, [295](#)
- SpatialFogGetView, [295](#)
- SpatialFogSetDepthDistance, [295](#)
- SpatialFogSetFormula, [295](#)
- SpatialFogSetParameters, [296](#)
- SpatialFogSetProjection, [296](#)
- SpatialFogSetView, [296](#)
- SubtractColors, [296](#)
- SurfaceClear, [296](#)
- SurfaceClearWith, [296](#)
- SurfaceConvertFormat, [297](#)
- SurfaceCopyFrom, [297](#)
- SurfaceCopyRect, [297](#)
- SurfaceCreate, [297](#)
- SurfaceCreateFromFile, [297](#)

- SurfaceCreateFromFileInMemory, 298
- SurfaceDestroy, 298
- SurfaceEmpty, 298
- SurfaceFlip, 298
- SurfaceGetBits, 298
- SurfaceGetByteSize, 299
- SurfaceGetBytesPerPixel, 299
- SurfaceGetFormat, 299
- SurfaceGetHeight, 299
- SurfaceGetPitch, 299
- SurfaceGetPixel, 299
- SurfaceGetPixelBilinear, 300
- SurfaceGetPixelPtr, 300
- SurfaceGetPremultipliedAlpha, 300
- SurfaceGetScanline, 300
- SurfaceGetWidth, 300
- SurfaceHasAlphaChannel, 300
- SurfaceInvert, 301
- SurfaceMakeSignedDistanceField, 301
- SurfaceMirror, 301
- SurfacePremultiplyAlpha, 301
- SurfaceResetAlpha, 301
- SurfaceResize, 302
- SurfaceSaveToFile, 302
- SurfaceSaveToFileInMemory, 302
- SurfaceSetPixel, 302
- SurfaceSetPremultipliedAlpha, 302
- SurfaceShrinkFrom, 303
- SurfaceStretchRect, 303
- SurfaceStretchRectBilinear, 303
- SurfaceUnpremultiplyAlpha, 303
- SwapChainBegin, 303
- SwapChainCreate, 304
- SwapChainDestroy, 304
- SwapChainEnd, 304
- SwapChainGetDepthStencil, 304
- SwapChainGetDevice, 304
- SwapChainGetFormat, 304
- SwapChainGetHeight, 305
- SwapChainGetMultisamples, 305
- SwapChainGetVSync, 305
- SwapChainGetWidth, 305
- SwapChainGetWindowHandle, 305
- SwapChainResize, 305
- TextModellerClear, 305
- TextModellerCopyToMeshBuffer, 306
- TextModellerCreate, 306
- TextModellerDestroy, 306
- TextModellerDraw, 306
- TextModellerDrawCurved, 306
- TextModellerDrawCurvedToCanvas, 307
- TextModellerDrawCurvedToCanvasBuffer, 307
- TextModellerDrawDepthCurved, 307
- TextModellerDrawToCanvas, 308
- TextModellerDrawToCanvasBuffer, 308
- TextModellerExtent, 308
- TextModellerExtentByShape, 308
- TextModellerGetDevice, 309
- TextModellerGetFontProvider, 309
- TextModellerGetTransform, 309
- TextModellerPrepare, 309
- TextModellerRects, 309
- TextModellerRender, 309
- TextModellerReset, 310
- TextModellerSetFontParameters, 310
- TextModellerSetFontProvider, 310
- TextModellerSetTransform, 310
- TextRendererCreate, 310
- TextRendererDestroy, 310
- TextRendererDraw, 311
- TextRendererDrawAligned, 311
- TextRendererDrawAlignedByPixels, 311
- TextRendererDrawCentered, 311
- TextRendererDrawCenteredByPixels, 312
- TextRendererExtent, 312
- TextRendererExtentByPixels, 312
- TextRendererGetCanvas, 312
- TextRendererGetFontParameters, 312
- TextRendererRects, 313
- TextRendererSetFontParameters, 313
- TextureAttach, 313
- TextureBegin, 313
- TextureBind, 313
- TextureCabinetBegin, 314
- TextureCabinetClear, 314
- TextureCabinetCreate, 314
- TextureCabinetDestroy, 314
- TextureCabinetEnd, 314
- TextureCabinetFilter, 314
- TextureCabinetGetAmbientOcclusionParameters, 315
- TextureCabinetGetAttributes, 315
- TextureCabinetGetDepthStencil, 315
- TextureCabinetGetDevice, 315
- TextureCabinetGetFidelity, 315
- TextureCabinetGetFinalTexture, 315
- TextureCabinetGetSamples, 316
- TextureCabinetGetSize, 316
- TextureCabinetGetTexture, 316
- TextureCabinetGetToneMappingBloom, 316
- TextureCabinetPresent, 316
- TextureCabinetRendering, 316
- TextureCabinetResolve, 317
- TextureCabinetRetrieveGlassyMetrics, 317
- TextureCabinetSetAmbientOcclusionParameters, 317
- TextureCabinetSetAttributes, 317
- TextureCabinetSetSize, 317
- TextureCabinetSetToneMappingBloom, 317
- TextureClear, 318
- TextureClearWith, 318
- TextureCopy, 318
- TextureCopyFromSurface, 318
- TextureCopyToSurface, 318
- TextureCreate, 319
- TextureCreateFromFile, 319

- TextureCreateFromFileInMemory, 319
- TextureCreateFromFileNormalsAndOcclusion, 319
- TextureCreateFromFileParallax, 320
- TextureCreateFromSurface, 320
- TextureDestroy, 320
- TextureDetach, 320
- TextureEnd, 320
- TextureGenerateMipMaps, 320
- TextureGetDevice, 321
- TextureGetParameters, 321
- TextureGetPlatformHandle, 321
- TextureLoadFromFile, 321
- TextureResetCache, 321
- TextureRetrieve, 321
- TextureSaveToFile, 322
- TextureUnbind, 322
- TextureUpdate, 322
- TimerCreate, 322
- TimerDestroy, 322
- TimerExtractTokens, 323
- TimerGetFrameRate, 323
- TimerGetLatency, 323
- TimerGetSkippedTimeSlices, 323
- TimerGetSpeed, 323
- TimerGetTimeSlice, 323
- TimerNextSlice, 324
- TimerReset, 324
- TimerSetSpeed, 324
- TimerTrimSkippedTimeSlices, 324
- TimerUpdate, 324
- TimerUpdateNextSlice, 324
- UnpremultiplyAlpha, 325
- VertexElementsEstimatePitch, 325
- VolumeCalculateNearFarPlanes, 325
- VolumeCalculateVisibleFrame, 325
- WidgetAcceptKey, 325
- WidgetAcceptMouse, 325
- WidgetAccomodate, 326
- WidgetBringToFront, 326
- WidgetCreate, 326
- WidgetDestroy, 326
- WidgetFindAt, 326
- WidgetGetChildByIndex, 326
- WidgetGetChildCount, 327
- WidgetGetChildIndex, 327
- WidgetGetExternalEvent, 327
- WidgetGetManager, 327
- WidgetGetParent, 327
- WidgetGetPayload, 327
- WidgetGetProperty, 328
- WidgetGetPropertyAsString, 328
- WidgetGetPropertyCount, 328
- WidgetInvalidate, 328
- WidgetInvokeEvent, 328
- WidgetLocalToScreen, 328
- WidgetManagerBatchCount, 329
- WidgetManagerCreate, 329
- WidgetManagerGetApplication, 329
- WidgetManagerGetAttributes, 329
- WidgetManagerGetFormat, 329
- WidgetManagerGetMultisamples, 329
- WidgetManagerGetTextRenderer, 330
- WidgetManagerGetTexture, 330
- WidgetManagerGetWidgetByName, 330
- WidgetManagerGetWidgetByPayload, 330
- WidgetManagerPresent, 330
- WidgetPropertyIdentify, 330
- WidgetScreenToLocal, 331
- WidgetSendToBack, 331
- WidgetSetExternalEvent, 331
- WidgetSetParent, 331
- WidgetSetProperty, 331
- WidgetUpdate, 332
- Afterwarrior.Structs.h, 363
- Afterwarrior.Types.h, 382
 - ALPHA_FORMAT_REQUEST_DONT_CARE, 405
 - ALPHA_FORMAT_REQUEST_NON_PREMULTIPLIED, 405
 - ALPHA_FORMAT_REQUEST_PREMULTIPLIED, 405
 - AlphaFormatRequest, 394
 - APP_CURSOR_ARROW, 430
 - APP_CURSOR_ARROW_WAIT, 430
 - APP_CURSOR_BLANK, 430
 - APP_CURSOR_CELL, 431
 - APP_CURSOR_CROSSHAIR, 431
 - APP_CURSOR_DRAG, 431
 - APP_CURSOR_DRAGGING, 431
 - APP_CURSOR_HELP, 431
 - APP_CURSOR_LINK_SELECT, 431
 - APP_CURSOR_MOVE, 431
 - APP_CURSOR_NOT_ALLOWED, 431
 - APP_CURSOR_RESIZE1, 431
 - APP_CURSOR_RESIZE2, 431
 - APP_CURSOR_RESIZE_BOTTOM, 431
 - APP_CURSOR_RESIZE_BOTTOM_LEFT, 431
 - APP_CURSOR_RESIZE_BOTTOM_RIGHT, 431
 - APP_CURSOR_RESIZE_HORIZ, 431
 - APP_CURSOR_RESIZE_LEFT, 431
 - APP_CURSOR_RESIZE_RIGHT, 431
 - APP_CURSOR_RESIZE_TOP, 431
 - APP_CURSOR_RESIZE_TOP_LEFT, 431
 - APP_CURSOR_RESIZE_TOP_RIGHT, 431
 - APP_CURSOR_RESIZE_VERT, 431
 - APP_CURSOR_TEXT_SELECT, 430
 - APP_CURSOR_TEXT_SELECT_VERTICAL, 430
 - APP_CURSOR_UNDEFINED, 431
 - APP_CURSOR_WAIT, 430
 - AppCursor, 394
 - APPLICATION_EVENT_ACTIVATE, 430
 - APPLICATION_EVENT_APPEARANCE, 430
 - APPLICATION_EVENT_DEACTIVATE, 430
 - APPLICATION_EVENT_MINIMIZE, 430
 - APPLICATION_EVENT_RESTORE, 430
 - APPLICATION_WINDOW_STATE_FULLSCREEN, 429

- APPLICATION_WINDOW_STATE_MAXIMIZED, 429
- APPLICATION_WINDOW_STATE_MINIMIZED, 429
- APPLICATION_WINDOW_STATE_WINDOWED, 429
- ApplicationEvent, 395
- ApplicationWindowState, 395
- AUTO_DRAW_OPTION_MATERIAL_IGNORE, 426
- AUTO_DRAW_OPTION_MATERIAL_SKIP_OPAQUE, 426
- AUTO_DRAW_OPTION_MATERIAL_SKIP_TRANSPARENT, 426
- AUTO_DRAW_OPTION_MATERIAL_TRANSPARENCY, 426
- AUTO_DRAW_OPTION_OBJECT_DISABLE_INSTANCING, 426
- AUTO_DRAW_OPTION_OBJECT_HIGHLIGHTED, 426
- AUTO_DRAW_OPTION_OBJECT_SKIP_HAVING_MATERIAL, 426
- AUTO_DRAW_OPTION_OBJECT_SKIP_NON_TRANSPARENT, 426
- AUTO_DRAW_OPTION_OBJECT_SKIP_NOT_HAVING_MATERIAL, 426
- AUTO_DRAW_OPTION_OBJECT_SKIP_TRANSPARENT, 426
- AUTO_DRAW_OPTION_POST_UNBIND, 426
- AUTO_DRAW_OPTION_RESET_TEXTURES, 426
- BLEND_FACTOR_CONSTANT_ALPHA, 410
- BLEND_FACTOR_CONSTANT_COLOR, 410
- BLEND_FACTOR_DEST_ALPHA, 410
- BLEND_FACTOR_DEST_COLOR, 410
- BLEND_FACTOR_INV_CONSTANT_ALPHA, 410
- BLEND_FACTOR_INV_CONSTANT_COLOR, 410
- BLEND_FACTOR_INV_DEST_ALPHA, 410
- BLEND_FACTOR_INV_DEST_COLOR, 410
- BLEND_FACTOR_INV_SOURCE_ALPHA, 410
- BLEND_FACTOR_INV_SOURCE_ALPHA_1, 410
- BLEND_FACTOR_INV_SOURCE_COLOR, 410
- BLEND_FACTOR_INV_SOURCE_COLOR_1, 410
- BLEND_FACTOR_ONE, 410
- BLEND_FACTOR_SOURCE_ALPHA, 410
- BLEND_FACTOR_SOURCE_ALPHA_1, 410
- BLEND_FACTOR_SOURCE_ALPHA_SAT, 410
- BLEND_FACTOR_SOURCE_COLOR, 410
- BLEND_FACTOR_SOURCE_COLOR_1, 410
- BLEND_FACTOR_ZERO, 410
- BLEND_OP_ADD, 410
- BLEND_OP_INV_SUBTRACT, 410
- BLEND_OP_MAXIMUM, 410
- BLEND_OP_MINIMUM, 410
- BLEND_OP_SUBTRACT, 410
- BlendFactor, 395
- BLENDING_EFFECT_ADD, 415
- BLENDING_EFFECT_COPY, 415
- BLENDING_EFFECT_CUSTOM, 415
- BLENDING_EFFECT_INVERSE_MULTIPLY, 415
- BLENDING_EFFECT_MULTIPLY, 415
- BLENDING_EFFECT_NORMAL, 415
- BLENDING_EFFECT_SHADOW, 415
- BLENDING_EFFECT_SOURCE_COLOR, 415
- BLENDING_EFFECT_SOURCE_COLOR_ADD, 415
- BLENDING_EFFECT_SUBTRACT, 415
- BLENDING_EFFECT_UNDEFINED, 415
- BlendingEffect, 395
- BlendOp, 395
- BUFFER_ACCESS_TYPE_COMPUTE, 411
- BUFFER_ACCESS_TYPE_DEFAULT, 411
- BUFFER_ACCESS_TYPE_DYNAMIC, 411
- BUFFER_ACCESS_TYPE_STAGING, 411
- BUFFER_ACCESS_TYPE_STATIC, 411
- BUFFER_DATA_TYPE_CONSTANT, 411
- BUFFER_DATA_TYPE_INDEX, 411
- BUFFER_DATA_TYPE_RW_STRUCTURED, 411
- BUFFER_DATA_TYPE_RW_TYPED, 411
- BUFFER_DATA_TYPE_STRUCTURED, 411
- BUFFER_DATA_TYPE_TYPED, 411
- BUFFER_DATA_TYPE_VERTEX, 411
- BufferDataAccessType, 395
- BufferDataType, 395
- CAMERA_COMMAND_DRAGGING, 429
- CAMERA_COMMAND_MOVEMENT, 429
- CAMERA_COMMAND_ROTATION, 429
- CAMERA_COMMAND_STOP, 429
- CAMERA_COMMAND_UPDATE, 429
- CameraCommand, 395
- CANVAS_ATTRIBUTE_COLOR_ADJUST, 416
- CANVAS_ATTRIBUTE_CUBIC, 416
- CANVAS_ATTRIBUTE_OUTLINE_SDF, 416
- CANVAS_ATTRIBUTE_PROJECTED, 416
- CANVAS_ATTRIBUTE_SDF, 416
- CANVAS_ATTRIBUTE_TRANSFORM, 416
- CANVAS_CONTEXT_STATE_FLAT_SCENE, 416
- CANVAS_CONTEXT_STATE_FLAT_TEXT, 416
- CANVAS_CONTEXT_STATE_PROJECTED, 416
- CANVAS_CONTEXT_STATE_UNDEFINED, 416
- CanvasContextState, 396
- CLUSTERS_CULLING_MODE_PERFORMANCE, 421
- CLUSTERS_CULLING_MODE_QUALITY, 421
- ClustersCullingMode, 396
- Color, 396
- COLOR_DITHERING_FORMAT_R5G6B5, 420
- COLOR_DITHERING_FORMAT_RG3B2, 420
- COLOR_DITHERING_FORMAT_RGBA, 420
- COLOR_DITHERING_FORMAT_RGB6, 420
- COLOR_DITHERING_FORMAT_RGB8, 420
- ColorDitheringFormat, 396
- ColorPair, 396
- ColorRect, 396
- COMPARISON_FUNC_ALWAYS, 409
- COMPARISON_FUNC_EQUAL, 409

- COMPARISON_FUNC_GREATER, 409
- COMPARISON_FUNC_GREATER_EQUAL, 409
- COMPARISON_FUNC_LESS, 409
- COMPARISON_FUNC_LESS_EQUAL, 409
- COMPARISON_FUNC_NEVER, 409
- COMPARISON_FUNC_NOT_EQUAL, 409
- ComparisonFunc, 396
- COMPUTE_TEXTURE_ACCESS_READ, 413
- COMPUTE_TEXTURE_ACCESS_READ_WRITE, 413
- COMPUTE_TEXTURE_ACCESS_WRITE, 413
- ComputeTextureAccess, 397
- DEPTH_FOG_DISTANCE_EYE_RELATIVE, 421
- DEPTH_FOG_DISTANCE_Z_BASED, 421
- DepthFogDistance, 397
- DEVICE_ATTRIBUTE_DEBUG, 406
- DEVICE_ATTRIBUTE_LEGACY, 406
- DEVICE_ATTRIBUTE_LIMITED_EXTENSIONS, 406
- DEVICE_ATTRIBUTE_SOFTWARE, 406
- DEVICE_BARRIER_ATOMIC_COUNTER, 407
- DEVICE_BARRIER_BUFFER_UPDATE, 407
- DEVICE_BARRIER_DRAWABLES, 407
- DEVICE_BARRIER_SHADER_BUFFER, 406
- DEVICE_BARRIER_SHADER_IMAGE_ACCESS, 407
- DEVICE_BARRIER_STRUCTURED, 407
- DEVICE_BARRIER_TEXTURE_FETCH, 407
- DEVICE_BARRIER_TEXTURE_UPDATE, 407
- DEVICE_BEHAVIOR_COMPUTE, 406
- DEVICE_BEHAVIOR_DEPTH_CLEAR_BAD_PRECISION, 406
- DEVICE_BEHAVIOR_DEPTH_CLIP_NEGATIVE, 406
- DEVICE_BEHAVIOR_FORCE_BUFFER_UNBIND, 406
- DEVICE_BEHAVIOR_PER_SAMPLE_SHADING, 406
- DEVICE_BEHAVIOR_POST_DEPTH_COVERAGE, 406
- DEVICE_BEHAVIOR_SIGNED_NORM_INT_FORMAT_BUFFER, 406
- DEVICE_BEHAVIOR_TESSELLATION, 406
- DEVICE_BEHAVIOR_VARIABLE_RATE_REFRESH, 406
- DEVICE_CLEAR_LAYER_COLOR, 405
- DEVICE_CLEAR_LAYER_DEPTH, 405
- DEVICE_CLEAR_LAYER_STENCIL, 405
- DEVICE_PLATFORM_ANDROID, 408
- DEVICE_PLATFORM_IOS, 408
- DEVICE_PLATFORM_LINUX, 408
- DEVICE_PLATFORM_OSX, 408
- DEVICE_PLATFORM_UNIX, 408
- DEVICE_PLATFORM_UNKNOWN, 408
- DEVICE_PLATFORM_WINDOWS, 408
- DEVICE_STATE_ALPHA_TO_COVERAGE, 408
- DEVICE_STATE_BLEND_ENABLE, 408
- DEVICE_STATE_COLOR_WRITE, 408
- DEVICE_STATE_CUBE_MAP_SEAMLESS, 408
- DEVICE_STATE_CULL_CLOCKWISE, 408
- DEVICE_STATE_DEPTH_CLIP, 408
- DEVICE_STATE_DEPTH_TEST, 408
- DEVICE_STATE_DEPTH_WRITE, 408
- DEVICE_STATE_LINE_ANTIALIAS, 408
- DEVICE_STATE_MULTISAMPLING, 408
- DEVICE_STATE_PER_SAMPLE_SHADING, 408
- DEVICE_STATE_SCISSOR_CLIP, 408
- DEVICE_STATE_STENCIL_TEST, 408
- DEVICE_STATE_WIREFRAME, 408
- DEVICE_TECHNOLOGY_DIRECT3D, 407
- DEVICE_TECHNOLOGY_METAL, 407
- DEVICE_TECHNOLOGY_OPENGL, 407
- DEVICE_TECHNOLOGY_OPENGL_ES, 407
- DEVICE_TECHNOLOGY_PROPRIETARY, 407
- DEVICE_TECHNOLOGY_SOFTWARE, 407
- DEVICE_TECHNOLOGY_UNKNOWN, 407
- DEVICE_TECHNOLOGY_VULKAN, 407
- DEVICE_TECHNOLOGY_WEBGL, 407
- DevicePlatform, 397
- DeviceTechnology, 397
- ELEMENT_FORMAT_BYTE, 412
- ELEMENT_FORMAT_DOUBLE, 412
- ELEMENT_FORMAT_FLOAT, 412
- ELEMENT_FORMAT_FLOAT_11_11_10, 412
- ELEMENT_FORMAT_HALF_FLOAT, 412
- ELEMENT_FORMAT_INT, 412
- ELEMENT_FORMAT_SHORT, 412
- ELEMENT_FORMAT_UNDEFINED, 412
- ELEMENT_FORMAT_UNSIGNED_BYTE, 412
- ELEMENT_FORMAT_UNSIGNED_INT, 412
- ELEMENT_FORMAT_UNSIGNED_SHORT, 412
- ElementFormat, 397
- FILE_CHOOSER_DIALOG_DIRECTORY, 431
- FILE_CHOOSER_DIALOG_OPEN_FILE, 431
- FILE_CHOOSER_DIALOG_SAVE_FILE, 431
- FileChooserDialog, 397
- FloatColor, 397
- FloatColorRGB, 398
- FOG_FORMULA_EXPONENTIAL, 420
- FOG_FORMULA_GROUND, 420
- FOG_FORMULA_LINEAR, 420
- FogFormula, 398
- FONT_ATTRIBUTE_STRIKE_OUT, 419
- FONT_ATTRIBUTE_UNDERLINE, 419
- FONT_BORDER_HEAVY, 419
- FONT_BORDER_NONE, 419
- FONT_BORDER_NORMAL, 419
- FONT_BORDER_SEMI_HEAVY, 419
- FONT_SLANT_ITALIC, 419
- FONT_SLANT_NONE, 419
- FONT_SLANT_OBLIQUE, 419
- FONT_STRETCH_CONDENSED, 418
- FONT_STRETCH_EXPANDED, 418
- FONT_STRETCH_EXTRA_CONDENSED, 418
- FONT_STRETCH_EXTRA_EXPANDED, 418
- FONT_STRETCH_NORMAL, 418

- FONT_STRETCH_SEMI_CONDENSED, [418](#)
- FONT_STRETCH_SEMI_EXPANDED, [418](#)
- FONT_STRETCH_ULTRA_CONDENSED, [418](#)
- FONT_STRETCH_ULTRA_EXPANDED, [418](#)
- FONT_WEIGHT_BOLD, [418](#)
- FONT_WEIGHT_EXTRA_BOLD, [418](#)
- FONT_WEIGHT_EXTRA_HEAVY, [418](#)
- FONT_WEIGHT_EXTRA_LIGHT, [418](#)
- FONT_WEIGHT_HEAVY, [418](#)
- FONT_WEIGHT_LIGHT, [418](#)
- FONT_WEIGHT_MEDIUM, [418](#)
- FONT_WEIGHT_NORMAL, [418](#)
- FONT_WEIGHT_SEMI_BOLD, [418](#)
- FONT_WEIGHT_SEMI_LIGHT, [418](#)
- FONT_WEIGHT_THIN, [418](#)
- FontAttributes, [398](#)
- FontBorder, [398](#)
- FontSlant, [398](#)
- FontStretch, [398](#)
- FontWeight, [398](#)
- IMAGE_REGION_FLIP, [417](#)
- IMAGE_REGION_MIRROR, [417](#)
- IMAGE_REGION_ROTATE, [417](#)
- Key, [399](#), [435](#)
- KEY_ADD, [437](#)
- KEY_ALT_LEFT, [436](#)
- KEY_ALT_RIGHT, [436](#)
- KEY_APPS, [437](#)
- KEY_BACKSPACE, [436](#)
- KEY_CANCEL, [436](#)
- KEY_CAPSLOCK, [436](#)
- KEY_CLEAR, [436](#)
- KEY_CTRL_LEFT, [436](#)
- KEY_CTRL_RIGHT, [436](#)
- KEY_DECIMAL, [437](#)
- KEY_DELETE, [436](#)
- KEY_DIVIDE, [437](#)
- KEY_DOWN, [436](#)
- KEY_END, [435](#)
- KEY_ESCAPE, [436](#)
- KEY_EVENT_PRESSED, [430](#)
- KEY_EVENT_RELEASED, [430](#)
- KEY_EVENT_TYPING, [430](#)
- KEY_EXECUTE, [437](#)
- KEY_F1, [436](#)
- KEY_F10, [436](#)
- KEY_F11, [436](#)
- KEY_F12, [436](#)
- KEY_F2, [436](#)
- KEY_F3, [436](#)
- KEY_F4, [436](#)
- KEY_F5, [436](#)
- KEY_F6, [436](#)
- KEY_F7, [436](#)
- KEY_F8, [436](#)
- KEY_F9, [436](#)
- KEY_HELP, [437](#)
- KEY_HOME, [435](#)
- KEY_INSERT, [436](#)
- KEY_LEFT, [436](#)
- KEY_LINEFEED, [436](#)
- KEY_MEDIA_NEXT_TRACK, [437](#)
- KEY_MEDIA_PLAY_PAUSE, [437](#)
- KEY_MEDIA_PREV_TRACK, [437](#)
- KEY_MEDIA_STOP, [437](#)
- KEY_MULTIPLY, [437](#)
- KEY_NULL, [435](#)
- KEY_NUM_LOCK, [436](#)
- KEY_NUMPAD_0, [436](#)
- KEY_NUMPAD_1, [436](#)
- KEY_NUMPAD_2, [436](#)
- KEY_NUMPAD_3, [436](#)
- KEY_NUMPAD_4, [436](#)
- KEY_NUMPAD_5, [436](#)
- KEY_NUMPAD_6, [436](#)
- KEY_NUMPAD_7, [436](#)
- KEY_NUMPAD_8, [436](#)
- KEY_NUMPAD_9, [437](#)
- KEY_PAGE_DOWN, [436](#)
- KEY_PAGE_UP, [436](#)
- KEY_PAUSE, [436](#)
- KEY_PRINT, [437](#)
- KEY_PRINT_SCREEN, [435](#)
- KEY_RETURN, [436](#)
- KEY_RIGHT, [436](#)
- KEY_SCROLL_LOCK, [436](#)
- KEY_SELECT, [437](#)
- KEY_SEPARATOR, [437](#)
- KEY_SHIFT_LEFT, [436](#)
- KEY_SHIFT_RIGHT, [436](#)
- KEY_SLEEP, [437](#)
- KEY_SPACE, [436](#)
- KEY_SUBTRACT, [437](#)
- KEY_SUPER_LEFT, [436](#)
- KEY_SUPER_RIGHT, [436](#)
- KEY_SYS_REQ, [436](#)
- KEY_TAB, [436](#)
- KEY_UP, [436](#)
- KEY_VOLUME_DOWN, [437](#)
- KEY_VOLUME_MUTE, [437](#)
- KEY_VOLUME_UP, [437](#)
- KeyEvent, [399](#)
- LEGACY_BIT_DEPTH_FORMAT_RAWZ, [405](#)
- LEGACY_BIT_DIFFERENT_BITDEPTH_MRT, [405](#)
- LEGACY_BIT_MISSING_DEPTH_TEXTURE, [405](#)
- LINE_CAPS_BUTT, [417](#)
- LINE_CAPS_ROUND, [417](#)
- LINE_CAPS_SQUARE, [417](#)
- LineCaps, [399](#)
- Margins, [399](#)
- Matrix, [399](#)
- Matrix3x2, [399](#)
- MESH_ALIGN_NEGATIVE, [427](#)
- MESH_ALIGN_ORIGIN, [427](#)
- MESH_ALIGN_POSITIVE, [427](#)

- MESH_ALIGN_UNALIGNED, [427](#)
- MESH_LOADING_OPTION_EXPORT_COLORS, [428](#)
- MESH_LOADING_OPTION_EXPORT_NORMALS, [428](#)
- MESH_LOADING_OPTION_EXPORT_TANGENTS, [428](#)
- MESH_LOADING_OPTION_EXPORT_TEXTURE_COORDINATES, [428](#)
- MESH_LOADING_OPTION_EXPORT_WEIGHTS, [428](#)
- MESH_LOADING_OPTION_MATERIAL_TEXTURE_MISSING, [428](#)
- MESH_LOADING_OPTION_MATERIAL_VERTEX_COLORS, [428](#)
- MESH_LOADING_OPTION_MATERIAL_VERTEX_COLORS_TEXTURE, [428](#)
- MESH_LOADING_OPTION_STRIP_GEOMETRY_NAMES, [428](#)
- MESH_META_TAG_TYPE_GEOMETRY, [425](#)
- MESH_META_TAG_TYPE_INDETERMINATE, [425](#)
- MESH_META_TAG_TYPE_OBJECT, [425](#)
- MeshAlign, [399](#)
- MODEL_TRANSFORM_GLOBAL, [427](#)
- MODEL_TRANSFORM_GLOBAL_MODEL, [427](#)
- MODEL_TRANSFORM_GLOBAL_VOLUME, [427](#)
- MODEL_TRANSFORM_LOCAL, [426](#)
- MODEL_TRANSFORM_LOCAL_MODEL, [426](#)
- MODEL_TRANSFORM_LOCAL_VOLUME, [426](#)
- ModelTransform, [399](#)
- MOUSE_BUTTON_LEFT, [430](#)
- MOUSE_BUTTON_MIDDLE, [430](#)
- MOUSE_BUTTON_NONE, [429](#)
- MOUSE_BUTTON_RIGHT, [430](#)
- MOUSE_EVENT_MOUSE_DOWN, [429](#)
- MOUSE_EVENT_MOUSE_ENTER, [429](#)
- MOUSE_EVENT_MOUSE_LEAVE, [429](#)
- MOUSE_EVENT_MOUSE_MOVE, [429](#)
- MOUSE_EVENT_MOUSE_UP, [429](#)
- MOUSE_EVENT_WHEEL_DOWN, [429](#)
- MOUSE_EVENT_WHEEL_UP, [429](#)
- MouseButton, [400](#)
- MouseEvent, [400](#)
- OBJECT_MODEL_ATTRIBUTE_DISABLE_REALIGN, [428](#)
- OBJECT_MODEL_ATTRIBUTE_HIERARCHYLESS, [428](#)
- OBJECT_MODEL_ATTRIBUTE_NON_VIEWABLE, [428](#)
- OBJECT_MODEL_ATTRIBUTE_SELECTABLE, [428](#)
- OBJECT_MODEL_ATTRIBUTE_TRANSPARENT, [428](#)
- OBJECT_MODEL_ATTRIBUTE_VISIBLE, [427](#)
- OBJECT_MODEL_COMPARE_DEPTH, [427](#)
- OBJECT_MODEL_COMPARE_DEPTH_REVERSE, [427](#)
- OBJECT_MODEL_COMPARE_MESHES, [427](#)
- OBJECT_MODEL_COMPARE_OBJECTS, [427](#)
- OBJECT_MODEL_COMPARE_ORDERED, [427](#)
- ObjectModelViewCompare, [400](#)
- PATH_JOINT_BEVEL, [417](#)
- PATH_JOINT_MITER, [417](#)
- PATH_JOINT_MITER_BEVEL, [417](#)
- PATH_JOINT_NONE, [417](#)
- PATH_JOINT_ROUND, [417](#)
- PATH_JOINT_SIMPLE, [417](#)
- PathJoint, [400](#)
- PIXEL_FORMAT_A8, [435](#)
- PIXEL_FORMAT_B5G6R5, [435](#)
- PIXEL_FORMAT_BGR10A2, [434](#)
- PIXEL_FORMAT_BGR10X2, [434](#)
- PIXEL_FORMAT_BGR5A1, [435](#)
- PIXEL_FORMAT_BGR5X1, [435](#)
- PIXEL_FORMAT_BGR8, [435](#)
- PIXEL_FORMAT_BGRA4, [435](#)
- PIXEL_FORMAT_BGRA8, [434](#)
- PIXEL_FORMAT_BGRA8_SRGB, [434](#)
- PIXEL_FORMAT_BGRX4, [435](#)
- PIXEL_FORMAT_BGRX8, [434](#)
- PIXEL_FORMAT_D16, [435](#)
- PIXEL_FORMAT_D24S8, [435](#)
- PIXEL_FORMAT_D32F, [435](#)
- PIXEL_FORMAT_D32S8F, [435](#)
- PIXEL_FORMAT_I8, [435](#)
- PIXEL_FORMAT_L16, [435](#)
- PIXEL_FORMAT_L8, [435](#)
- PIXEL_FORMAT_LA4, [435](#)
- PIXEL_FORMAT_LA8, [435](#)
- PIXEL_FORMAT_R16, [433](#)
- PIXEL_FORMAT_R16F, [434](#)
- PIXEL_FORMAT_R16I, [434](#)
- PIXEL_FORMAT_R16S, [433](#)
- PIXEL_FORMAT_R16U, [434](#)
- PIXEL_FORMAT_R32F, [434](#)
- PIXEL_FORMAT_R32I, [434](#)
- PIXEL_FORMAT_R32U, [434](#)
- PIXEL_FORMAT_R8, [433](#)
- PIXEL_FORMAT_R8I, [434](#)
- PIXEL_FORMAT_R8S, [433](#)
- PIXEL_FORMAT_R8U, [434](#)
- PIXEL_FORMAT_R_BC, [435](#)
- PIXEL_FORMAT_RG11B10F, [434](#)
- PIXEL_FORMAT_RG16, [433](#)
- PIXEL_FORMAT_RG16F, [434](#)
- PIXEL_FORMAT_RG16I, [434](#)
- PIXEL_FORMAT_RG16S, [433](#)
- PIXEL_FORMAT_RG16U, [434](#)
- PIXEL_FORMAT_RG32F, [434](#)
- PIXEL_FORMAT_RG32I, [434](#)
- PIXEL_FORMAT_RG32U, [434](#)
- PIXEL_FORMAT_RG8, [433](#)
- PIXEL_FORMAT_RG8I, [434](#)
- PIXEL_FORMAT_RG8S, [433](#)
- PIXEL_FORMAT_RG8U, [434](#)
- PIXEL_FORMAT_RG_BC, [435](#)

- PIXEL_FORMAT_RGB10A2, [434](#)
- PIXEL_FORMAT_RGB10A2U, [434](#)
- PIXEL_FORMAT_RGB8, [435](#)
- PIXEL_FORMAT_RGB9E5F, [434](#)
- PIXEL_FORMAT_RGB_BC, [435](#)
- PIXEL_FORMAT_RGB_BC_SRGB, [435](#)
- PIXEL_FORMAT_RGBA16, [433](#)
- PIXEL_FORMAT_RGBA16F, [434](#)
- PIXEL_FORMAT_RGBA16I, [434](#)
- PIXEL_FORMAT_RGBA16S, [433](#)
- PIXEL_FORMAT_RGBA16U, [434](#)
- PIXEL_FORMAT_RGBA32F, [434](#)
- PIXEL_FORMAT_RGBA32I, [434](#)
- PIXEL_FORMAT_RGBA32U, [434](#)
- PIXEL_FORMAT_RGBA8, [433](#)
- PIXEL_FORMAT_RGBA8_SRGB, [434](#)
- PIXEL_FORMAT_RGBA8I, [434](#)
- PIXEL_FORMAT_RGBA8S, [433](#)
- PIXEL_FORMAT_RGBA8U, [434](#)
- PIXEL_FORMAT_RGBA_BC, [435](#)
- PIXEL_FORMAT_RGBA_BC_SRGB, [435](#)
- PIXEL_FORMAT_RGBX8, [434](#)
- PIXEL_FORMAT_UNKNOWN, [433](#)
- PixelFormat, [400](#)
- Point, [400](#)
- POINT_SHAPE_CROSS, [418](#)
- POINT_SHAPE_ROUND, [418](#)
- POINT_SHAPE_SQUARE, [418](#)
- POINT_SHAPE_STAR, [418](#)
- POINT_SHAPE_TRIANGLE, [418](#)
- PointF, [400](#)
- PointShape, [400](#)
- PRIMITIVE_TOPOLOGY_LINE_STRIP, [411](#)
- PRIMITIVE_TOPOLOGY_LINE_STRIP_ADJACENCY, [412](#)
- PRIMITIVE_TOPOLOGY_LINES, [411](#)
- PRIMITIVE_TOPOLOGY_LINES_ADJACENCY, [412](#)
- PRIMITIVE_TOPOLOGY_POINTS, [411](#)
- PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP, [412](#)
- PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_ADJACENCY, [412](#)
- PRIMITIVE_TOPOLOGY_TRIANGLES, [412](#)
- PRIMITIVE_TOPOLOGY_TRIANGLES_ADJACENCY, [412](#)
- PRIMITIVE_TOPOLOGY_UNKNOWN, [411](#)
- PrimitiveTopology, [401](#)
- Quad, [401](#)
- Quaternion, [401](#)
- Rect, [401](#)
- RectF, [401](#)
- SCENE_ATTRIBUTE_COLORING, [424](#)
- SCENE_ATTRIBUTE_DEPTH_PREPASS, [424](#)
- SCENE_ATTRIBUTE_GLASSY, [424](#)
- SCENE_ATTRIBUTE_INSTANCING, [424](#)
- SCENE_ATTRIBUTE_LINEAR_DEPTHS, [424](#)
- SCENE_ATTRIBUTE_MODELING, [424](#)
- SCENE_ATTRIBUTE_NORMALS, [424](#)
- SCENE_ATTRIBUTE_SHADOWS_CUBIC, [424](#)
- SCENE_ATTRIBUTE_SINGLE_PASS, [424](#)
- SCENE_ATTRIBUTE_TEXTUREING_CUBIC, [424](#)
- SCENE_MESH_LATCH_TYPE_JOINT, [425](#)
- SCENE_MESH_LATCH_TYPE_WAYPOINT, [425](#)
- SCENE_MESH_TEXTURE_DIFFUSE, [425](#)
- SCENE_MESH_TEXTURE_NORMALS, [425](#)
- SCENE_MESH_TEXTURE_PARALLAX, [425](#)
- SCENE_MESH_VERTEX_ELEMENTS_INDEX_UNDEFINED, [425](#)
- SCENE_SAMPLER_TYPE_ALBEDO, [424](#)
- SCENE_SAMPLER_TYPE_NORMAL_MAP, [424](#)
- SCENE_SAMPLER_TYPE_PARALLAX_MAP, [424](#)
- SCENE_SAMPLER_TYPE_SHADOW_MAP, [424](#)
- SCENE_TEXTURE_TYPE_ALBEDO, [424](#)
- SCENE_TEXTURE_TYPE_NORMAL_MAP, [424](#)
- SCENE_TEXTURE_TYPE_PARALLAX_MAP, [424](#)
- SceneSamplerType, [401](#)
- SceneTextureType, [401](#)
- SELECTION_HIGHLIGHT_TEXTURE_HIGHLIGHT, [420](#)
- SELECTION_HIGHLIGHT_TEXTURE_HIGHLIGHT_MASK, [420](#)
- SelectionHighlightTextureType, [401](#)
- SHADER_ELEMENT_CONSTANT_BUFFER, [413](#)
- SHADER_ELEMENT_RW_STRUCTURED_BUFFER, [413](#)
- SHADER_ELEMENT_RW_Typed_BUFFER, [413](#)
- SHADER_ELEMENT_STRUCTURED_BUFFER, [413](#)
- SHADER_ELEMENT_TEXTURE, [413](#)
- SHADER_ELEMENT_Typed_BUFFER, [413](#)
- SHADER_ELEMENT_UNDEFINED, [413](#)
- SHADER_INDEX_UNDEFINED, [413](#)
- SHADER_TYPE_COMPUTE, [412](#)
- SHADER_TYPE_DOMAIN, [412](#)
- SHADER_TYPE_FRAGMENT, [412](#)
- SHADER_TYPE_GEOMETRY, [412](#)
- SHADER_TYPE_HULL, [412](#)
- SHADER_TYPE_UNDEFINED, [412](#)
- SHADER_TYPE_VERTEX, [412](#)
- ShaderElement, [402](#)
- ShaderType, [402](#)
- STENCIL_OP_DECREMENT, [409](#)
- STENCIL_OP_DECREMENT_WARP, [409](#)
- STENCIL_OP_INCREMENT, [409](#)
- STENCIL_OP_INCREMENT_WARP, [409](#)
- STENCIL_OP_INVERT, [409](#)
- STENCIL_OP_KEEP, [409](#)
- STENCIL_OP_REPLACE, [409](#)
- STENCIL_OP_ZERO, [409](#)
- StencilOp, [402](#)
- SUPER_SAMPLE_SDF_NO_SUPER_SAMPLE, [416](#)
- SUPER_SAMPLE_SDF_SUPER_SAMPLE_16X, [416](#)
- SUPER_SAMPLE_SDF_SUPER_SAMPLE_4X, [416](#)

- SuperSampleSDF, [402](#)
- TECHNIQUE_LIGHTING_COOK_TORRANCE, [421](#)
- TECHNIQUE_LIGHTING_MINNAERT, [421](#)
- TECHNIQUE_LIGHTING_OREN_NAYER, [421](#)
- TECHNIQUE_LIGHTING_PHONG, [421](#)
- TECHNIQUE_SHADOWS_ESM, [421](#)
- TECHNIQUE_SHADOWS_ESM_WARP, [421](#)
- TECHNIQUE_SHADOWS_EVSM, [421](#)
- TECHNIQUE_SHADOWS_NONE, [421](#)
- TechniqueLighting, [402](#)
- TechniqueShadows, [402](#)
- TEXT_ALIGNMENT_END, [419](#)
- TEXT_ALIGNMENT_MIDDLE, [419](#)
- TEXT_ALIGNMENT_START, [419](#)
- TextAlignment, [402](#)
- TEXTURE_ADDRESS_BORDER, [414](#)
- TEXTURE_ADDRESS_CLAMP, [414](#)
- TEXTURE_ADDRESS_MIRROR, [414](#)
- TEXTURE_ADDRESS_MIRROR_ONCE, [414](#)
- TEXTURE_ADDRESS_WRAP, [414](#)
- TEXTURE_ATTRIBUTE_DRAWABLE, [414](#)
- TEXTURE_ATTRIBUTE_DYNAMIC, [414](#)
- TEXTURE_ATTRIBUTE_MIPMAPPING, [414](#)
- TEXTURE_ATTRIBUTE_PREMULTIPLIED_ALPHA, [415](#)
- TEXTURE_CABINET_ATTRIBUTE_AMBIENT_OCCLUSION, [423](#)
- TEXTURE_CABINET_ATTRIBUTE_AMBIENT_OCCLUSION_HD, [423](#)
- TEXTURE_CABINET_ATTRIBUTE_BLOOM, [423](#)
- TEXTURE_CABINET_ATTRIBUTE_BLOOM_CUBIC, [423](#)
- TEXTURE_CABINET_ATTRIBUTE_BLOOM_DOWNSCALE, [423](#)
- TEXTURE_CABINET_ATTRIBUTE_GLASSY, [423](#)
- TEXTURE_CABINET_ATTRIBUTE_GLASSY_FAST, [423](#)
- TEXTURE_CABINET_ATTRIBUTE_GLASSY_FOG, [423](#)
- TEXTURE_CABINET_ATTRIBUTE_GLASSY_FROSTED, [423](#)
- TEXTURE_CABINET_ATTRIBUTE_LINEAR Depths, [423](#)
- TEXTURE_CABINET_FILTER_TYPE_BLOOM, [423](#)
- TEXTURE_CABINET_FILTER_TYPE_GLASSY, [423](#)
- TEXTURE_CABINET_FILTER_TYPE_GLASSY_FOG, [423](#)
- TEXTURE_CABINET_FILTER_TYPE_OCCLUSION, [423](#)
- TEXTURE_CABINET_PASS_COLOR, [422](#)
- TEXTURE_CABINET_PASS_DEPTH, [422](#)
- TEXTURE_CABINET_PASS_GLASSY, [422](#)
- TEXTURE_CABINET_TYPE_COLOR, [422](#)
- TEXTURE_CABINET_TYPE_COLOR_HDR, [422](#)
- TEXTURE_CABINET_TYPE_DEPTH, [422](#)
- TEXTURE_CABINET_TYPE_LINEAR_Depths, [422](#)
- TEXTURE_CABINET_TYPE_NORMALS, [422](#)
- TEXTURE_COMPUTE, [415](#)
- TEXTURE_FIDELITY_EXTREME, [422](#)
- TEXTURE_FIDELITY_HIGH, [422](#)
- TEXTURE_FIDELITY_LOW, [422](#)
- TEXTURE_FIDELITY_MEDIUM, [422](#)
- TEXTURE_FILTER_ANISOTROPIC, [414](#)
- TEXTURE_FILTER_LINEAR, [414](#)
- TEXTURE_FILTER_NEAREST, [414](#)
- TEXTURE_FILTER_NONE, [414](#)
- TEXTURE_SCRATCH, [415](#)
- TEXTURE_TYPE_CUBE_MAP, [415](#)
- TEXTURE_TYPE_SURFACE, [415](#)
- TEXTURE_TYPE_VOLUME, [415](#)
- TextureAddress, [402](#)
- TextureCabinetFilterType, [403](#)
- TextureCabinetPass, [403](#)
- TextureCabinetType, [403](#)
- TextureFidelity, [403](#)
- TextureFilter, [403](#)
- TextureType, [403](#)
- TRIANGLE_FACE_BACK, [409](#)
- TRIANGLE_FACE_BOTH, [409](#)
- TRIANGLE_FACE_FRONT, [409](#)
- TRIANGLE_FACE_NONE, [409](#)
- TriangleFace, [403](#)
- Vec4, [403](#)
- Vector4, [404](#)
- VERTEX_CHANNEL_INDEX_BUFFER, [413](#)
- VERTEX_CHANNEL_NO_BUFFERS, [413](#)
- VERTEX_CHANNEL_NORMALIZED, [413](#)
- WIDGET_ALIGNMENT_BOTTOM, [432](#)
- WIDGET_ALIGNMENT_CLIENT, [431](#)
- WIDGET_ALIGNMENT_LEFT, [431](#)
- WIDGET_ALIGNMENT_NONE, [431](#)
- WIDGET_ALIGNMENT_RIGHT, [432](#)
- WIDGET_ALIGNMENT_TOP, [432](#)
- WIDGET_MANAGER_ATTRIBUTE_COMPOSITION, [437](#)
- WIDGET_MANAGER_ATTRIBUTE_DESIGN, [437](#)
- WIDGET_MANAGER_TEXTURE_TYPE_AUXILIARY, [433](#)
- WIDGET_MANAGER_TEXTURE_TYPE_CANVAS, [433](#)
- WIDGET_MANAGER_TEXTURE_TYPE_SCREEN, [433](#)
- WIDGET_PROPERTY_ATTRIBUTE_ASSIGNED, [433](#)
- WIDGET_PROPERTY_BEHAVIOR_EVENT, [432](#)
- WIDGET_PROPERTY_BEHAVIOR_INDIRECT, [432](#)
- WIDGET_PROPERTY_BEHAVIOR_NORMAL, [432](#)
- WIDGET_PROPERTY_BEHAVIOR_STATIC, [432](#)
- WIDGET_PROPERTY_BEHAVIOR_VOLATILE, [432](#)

- WIDGET_PROPERTY_TYPE_BOOLEAN, [432](#)
- WIDGET_PROPERTY_TYPE_COLOR, [432](#)
- WIDGET_PROPERTY_TYPE_COLOR_PAIR, [432](#)
- WIDGET_PROPERTY_TYPE_COLOR_RECT, [432](#)
- WIDGET_PROPERTY_TYPE_ENUMERATION, [432](#)
- WIDGET_PROPERTY_TYPE_FLOAT, [432](#)
- WIDGET_PROPERTY_TYPE_FONT, [432](#)
- WIDGET_PROPERTY_TYPE_INT64, [432](#)
- WIDGET_PROPERTY_TYPE_INTEGER, [432](#)
- WIDGET_PROPERTY_TYPE_MARGINS, [432](#)
- WIDGET_PROPERTY_TYPE_NONE, [432](#)
- WIDGET_PROPERTY_TYPE_POINT, [432](#)
- WIDGET_PROPERTY_TYPE_REAL, [432](#)
- WIDGET_PROPERTY_TYPE_RECT, [432](#)
- WIDGET_PROPERTY_TYPE_STRING, [432](#)
- WIDGET_PROPERTY_TYPE_VECTOR, [432](#)
- WidgetAlignment, [404](#)
- WidgetManagerAttribute, [437](#)
- WidgetManagerTextureType, [404](#)
- WidgetPropertyBehavior, [404](#)
- WidgetPropertyType, [404](#)
- Afterwarp.Vectors.h, [456](#)
- AFTERWARP_VECTORS_H, [467](#)
- AFTERWARP_VECTORS_INTERFACE, [467](#)
- colorBlack, [507](#)
- colorPair, [468](#)
- colorPairBlack, [507](#)
- colorPairTraslucentBlack, [507](#)
- colorPairTraslucentWhite, [507](#)
- colorPairWhite, [507](#)
- colorRectBlack, [507](#)
- colorRectTraslucentBlack, [507](#)
- colorRectTraslucentWhite, [507](#)
- colorRectWhite, [508](#)
- colorTraslucentBlack, [508](#)
- colorTraslucentWhite, [508](#)
- colorWhite, [508](#)
- floatColor, [468](#)
- floatColorAdd, [468](#)
- floatColorAddRGB, [468](#)
- floatColorAverage, [468](#)
- floatColorAverageRGB, [469](#)
- floatColorBlack, [508](#)
- floatColorBlackRGB, [508](#)
- floatColorDivide, [469](#)
- floatColorDivideRGB, [469](#)
- floatColorEmpty, [469](#)
- floatColorEmptyRGB, [469](#)
- floatColorEquals, [469](#)
- floatColorEqualsRGB, [470](#)
- floatColorFromValue, [470](#)
- floatColorFromValueRGB, [470](#)
- floatColorGray, [470](#)
- floatColorGrayRGB, [470](#)
- floatColorInverse, [470](#)
- floatColorInverseRGB, [471](#)
- floatColorLerp, [471](#)
- floatColorLerpRGB, [471](#)
- floatColorMultiply, [471](#)
- floatColorMultiplyRGB, [471](#)
- floatColorNegate, [471](#)
- floatColorNegateRGB, [472](#)
- floatColorPremultiply, [472](#)
- floatColorRGB, [472](#)
- floatColorSaturate, [472](#)
- floatColorSaturateRGB, [472](#)
- floatColorSubtract, [472](#)
- floatColorSubtractRGB, [472](#)
- floatColorToRGB, [473](#)
- floatColorTraslucentBlack, [508](#)
- floatColorTraslucentWhite, [508](#)
- floatColorUnpremultiply, [473](#)
- floatColorWhite, [509](#)
- floatColorWhiteRGB, [509](#)
- floatToColor, [473](#)
- floatToColorRGB, [473](#)
- fontEffectDefault, [509](#)
- makeBounds, [473](#)
- makeBoundsF, [473](#)
- makeColorPair, [474](#)
- makeFloatColor, [474](#)
- makeFloatColorRGB, [474](#)
- makePoint, [474](#)
- makePointF, [474](#)
- makeQuad, [474](#)
- makeQuadRotated, [475](#)
- makeQuadRotatedAt, [475](#)
- makeQuadRotatedTL, [475](#)
- makeQuadScaled, [475](#)
- makeQuadSkewedHoriz, [476](#)
- makeQuadSkewedVert, [476](#)
- makeQuadWith, [476](#)
- makeRect, [476](#)
- makeRectF, [476](#)
- makeVector, [477](#)
- makeVector3D, [477](#)
- matrixAdjoint, [477](#)
- matrixDeterminant, [477](#)
- matrixDeterminant3x2, [477](#)
- matrixEmpty, [477](#)
- matrixEmpty3x2, [478](#)
- matrixEquals, [478](#)
- matrixEquals3x2, [478](#)
- matrixEyePosition, [478](#)
- matrixGet3x2, [478](#)
- matrixHeadingPitchBank, [478](#)
- matrixIdentity, [509](#)
- matrixIdentity3x2, [509](#)
- matrixInverse, [479](#)
- matrixInverse3x2, [479](#)
- matrixInverseTranspose3x2, [479](#)
- matrixLookAt, [479](#)
- matrixMultiply, [479](#)
- matrixMultiply3x2, [479](#)

matrixOrthogonalBDS, [480](#)
matrixOrthogonalVOL, [480](#)
matrixPerspectiveBDS, [480](#)
matrixPerspectiveFOVX, [480](#)
matrixPerspectiveFOVY, [480](#)
matrixPerspectiveVOL, [481](#)
matrixQuaternion, [481](#)
matrixReflect, [481](#)
matrixRescale, [481](#)
matrixRescale3x2, [481](#)
matrixRotate, [482](#)
matrixRotate3x2, [482](#)
matrixRotateBy3x2, [482](#)
matrixRotateX, [482](#)
matrixRotateY, [482](#)
matrixRotateZ, [482](#)
matrixScale, [483](#)
matrixScale3x2, [483](#)
matrixScaleBy3x2, [483](#)
matrixSub3x3, [483](#)
matrixTranslate, [483](#)
matrixTranslate3x2, [483](#)
matrixTranspose, [483](#)
matrixWorldPosition, [484](#)
matrixYawPitchRoll, [484](#)
matrixZero, [509](#)
matrixZero3x2, [509](#)
MAX, [468](#)
MIN, [468](#)
pointAdd, [484](#)
pointAddF, [484](#)
pointAngle, [484](#)
pointAngleF, [484](#)
pointAxisX, [509](#)
pointAxisXF, [510](#)
pointAxisY, [510](#)
pointAxisYF, [510](#)
pointCross, [485](#)
pointCrossF, [485](#)
pointDistance, [485](#)
pointDistanceF, [485](#)
pointDivide, [485](#)
pointDivideF, [485](#)
pointDot, [486](#)
pointDotF, [486](#)
pointEmpty, [486](#)
pointEmptyF, [486](#)
pointEquals, [486](#)
pointEqualsF, [486](#)
pointF, [487](#)
pointInRect, [487](#)
pointInRectF, [487](#)
pointLength, [487](#)
pointLengthF, [487](#)
pointLerp, [487](#)
pointLerpF, [488](#)
pointMultiply, [488](#)
pointMultiplyF, [488](#)
pointNegate, [488](#)
pointNegateF, [488](#)
pointNormalizeF, [488](#)
pointRescale, [489](#)
pointRescaleF, [489](#)
pointSubtract, [489](#)
pointSubtractF, [489](#)
pointTransformF, [489](#)
pointUnity, [510](#)
pointUnityF, [510](#)
pointValue, [489](#)
pointValueF, [490](#)
pointZero, [510](#)
pointZeroF, [510](#)
quadFlip, [490](#)
quadFromRect, [490](#)
quadFromRectF, [490](#)
quadMirror, [490](#)
quadOffset, [490](#)
quadScale, [491](#)
quadTransform, [491](#)
quadUnity, [510](#)
quadZero, [511](#)
quaternionAngle, [491](#)
quaternionAxis, [491](#)
quaternionConjugate, [491](#)
quaternionDot, [491](#)
quaternionExponentiate, [492](#)
quaternionIdentity, [511](#)
quaternionLength, [492](#)
quaternionMatrix, [492](#)
quaternionMultiply, [492](#)
quaternionNormalize, [492](#)
quaternionRotate, [492](#)
quaternionRotateInertialToObject, [493](#)
quaternionRotateObjectToInertial, [493](#)
quaternionRotateX, [493](#)
quaternionRotateY, [493](#)
quaternionRotateZ, [493](#)
quaternionSlerp, [493](#)
rectEmpty, [494](#)
rectEmptyF, [494](#)
rectEquals, [494](#)
rectEqualsF, [494](#)
rectF, [494](#)
rectGetBottom, [494](#)
rectGetBottomF, [495](#)
rectGetBottomLeft, [495](#)
rectGetBottomLeftF, [495](#)
rectGetBottomRight, [495](#)
rectGetBottomRightF, [495](#)
rectGetRight, [495](#)
rectGetRightF, [495](#)
rectGetSize, [496](#)
rectGetSizeF, [496](#)
rectGetTopLeft, [496](#)
rectGetTopLeftF, [496](#)
rectGetTopRight, [496](#)

- rectGetTopRightF, [496](#)
- rectInflate, [496](#)
- rectInflateF, [497](#)
- rectInRect, [497](#)
- rectInRectF, [497](#)
- rectIntersect, [497](#)
- rectIntersectF, [497](#)
- rectJoin, [497](#)
- rectJoinF, [498](#)
- rectOffset, [498](#)
- rectOffsetF, [498](#)
- rectOverlap, [498](#)
- rectOverlapF, [498](#)
- rectSetBottom, [498](#)
- rectSetBottomF, [499](#)
- rectSetBottomRight, [499](#)
- rectSetBottomRightF, [499](#)
- rectSetRight, [499](#)
- rectSetRightF, [499](#)
- rectSetSize, [499](#)
- rectSetSizeF, [500](#)
- rectSetTopLeft, [500](#)
- rectSetTopLeftF, [500](#)
- rectToIntRectF, [500](#)
- rectZero, [511](#)
- rectZeroF, [511](#)
- vectorAdd, [500](#)
- vectorAdd3D, [500](#)
- vectorAngle, [501](#)
- vectorAxisW3D, [511](#)
- vectorAxisX, [511](#)
- vectorAxisX3D, [511](#)
- vectorAxisY, [511](#)
- vectorAxisY3D, [512](#)
- vectorAxisZ, [512](#)
- vectorAxisZ3D, [512](#)
- vectorCross, [501](#)
- vectorDistance, [501](#)
- vectorDistance3D, [501](#)
- vectorDivide, [501](#)
- vectorDivide3D, [501](#)
- vectorDot, [502](#)
- vectorDot3D, [502](#)
- vectorEmpty, [502](#)
- vectorEmpty3D, [502](#)
- vectorEpsilon, [512](#)
- vectorEquals, [502](#)
- vectorEquals3D, [502](#)
- vectorFromValue3D, [503](#)
- vectorLength, [503](#)
- vectorLength3D, [503](#)
- vectorLerp, [503](#)
- vectorLerp3D, [503](#)
- vectorMultiply, [503](#)
- vectorMultiply3D, [504](#)
- vectorNegate, [504](#)
- vectorNegate3D, [504](#)
- vectorNormalize, [504](#)
- vectorNormalize3D, [504](#)
- vectorParallel, [504](#)
- vectorPerpendicular, [505](#)
- vectorProject, [505](#)
- vectorProjectTarget, [505](#)
- vectorReflect, [505](#)
- vectorRescale, [505](#)
- vectorRescale3D, [505](#)
- vectorSubtract, [506](#)
- vectorSubtract3D, [506](#)
- vectorTransform, [506](#)
- vectorTransform3D, [506](#)
- vectorUnity, [512](#)
- vectorUnity3D, [512](#)
- vectorValue, [506](#)
- vectorZero, [512](#)
- vectorZero3D, [513](#)
- AFTERWARP_VECTORS_H
 - Afterwarp.Vectors.h, [467](#)
- AFTERWARP_VECTORS_INTERFACE
 - Afterwarp.Vectors.h, [467](#)
- albedoColor
 - ObjectMaterial, [66](#)
 - OceanMaterial, [69](#)
 - SceneLight, [91](#)
- alpha
 - FloatColor, [47](#)
- ALPHA_FORMAT_REQUEST_DONT_CARE
 - Afterwarp.Types.h, [405](#)
- ALPHA_FORMAT_REQUEST_NON_PREMULTIPLIED
 - Afterwarp.Types.h, [405](#)
- ALPHA_FORMAT_REQUEST_PREMULTIPLIED
 - Afterwarp.Types.h, [405](#)
- AlphaFormatRequest
 - Afterwarp.Types.h, [394](#)
- ambient
 - SceneMeshMaterialShading, [97](#)
- ambientColor
 - ObjectMaterial, [66](#)
 - OceanMaterial, [69](#)
 - SceneLight, [91](#)
- AmbientOcclusionParameters, [31](#)
 - attenuation, [32](#)
 - blurFallOff, [32](#)
 - blurSigma, [32](#)
 - contrast, [32](#)
 - depthBias, [32](#)
 - kernelBias, [32](#)
 - pvt, [17](#)
 - radius, [32](#)
 - samples, [32](#)
 - strength, [33](#)
- angle
 - SceneLight, [91](#)
- angleCutoff
 - SceneLight, [92](#)
- anisotropy
 - SamplerState, [89](#)

- APP_CURSOR_ARROW
 - Afterwarp.Types.h, [430](#)
- APP_CURSOR_ARROW_WAIT
 - Afterwarp.Types.h, [430](#)
- APP_CURSOR_BLANK
 - Afterwarp.Types.h, [430](#)
- APP_CURSOR_CELL
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_CROSSHAIR
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_DRAG
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_DRAGGING
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_HELP
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_LINK_SELECT
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_MOVE
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_NOT_ALLOWED
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE1
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE2
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_BOTTOM
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_BOTTOM_LEFT
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_BOTTOM_RIGHT
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_HORIZ
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_LEFT
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_RIGHT
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_TOP
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_TOP_LEFT
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_TOP_RIGHT
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_RESIZE_VERT
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_TEXT_SELECT
 - Afterwarp.Types.h, [430](#)
- APP_CURSOR_TEXT_SELECT_VERTICAL
 - Afterwarp.Types.h, [430](#)
- APP_CURSOR_UNDEFINED
 - Afterwarp.Types.h, [431](#)
- APP_CURSOR_WAIT
 - Afterwarp.Types.h, [430](#)
- AppCursor
 - Afterwarp.Types.h, [394](#)
- APPLICATION_EVENT_ACTIVATE
 - Afterwarp.Types.h, [430](#)
- APPLICATION_EVENT_APPEARANCE
 - Afterwarp.Types.h, [430](#)
- APPLICATION_EVENT_DEACTIVATE
 - Afterwarp.Types.h, [430](#)
- APPLICATION_EVENT_MINIMIZE
 - Afterwarp.Types.h, [430](#)
- APPLICATION_EVENT_RESTORE
 - Afterwarp.Types.h, [430](#)
- Application_t
 - pxt, [17](#)
- APPLICATION_WINDOW_STATE_FULLSCREEN
 - Afterwarp.Types.h, [429](#)
- APPLICATION_WINDOW_STATE_MAXIMIZED
 - Afterwarp.Types.h, [429](#)
- APPLICATION_WINDOW_STATE_MINIMIZED
 - Afterwarp.Types.h, [429](#)
- APPLICATION_WINDOW_STATE_WINDOWED
 - Afterwarp.Types.h, [429](#)
- ApplicationCaptureMouseInput
 - Afterwarp.h, [166](#)
- ApplicationConfiguration, [33](#)
 - handleIconBig, [34](#)
 - handleIconSmall, [34](#)
 - handleInstance, [34](#)
 - nx, [35](#)
 - parameterCount, [35](#)
 - parameterStrings, [35](#)
 - position, [35](#)
 - pxt, [17](#)
 - renderingType, [35](#)
 - size, [35](#)
 - startup, [35](#)
 - state, [35](#)
 - win, [36](#)
 - windowClassName, [36](#)
- ApplicationConvertPortableKey
 - Afterwarp.h, [166](#)
- ApplicationCreate
 - Afterwarp.h, [166](#)
- ApplicationDestroy
 - Afterwarp.h, [167](#)
- ApplicationEvent
 - Afterwarp.Types.h, [395](#)
- ApplicationEvents, [36](#)
 - eventCreate, [37](#)
 - eventDestroy, [37](#)
 - eventHook, [37](#)
 - eventIdle, [37](#)
 - eventKey, [38](#)
 - eventMouse, [38](#)
 - eventRender, [38](#)
 - eventResize, [38](#)
 - eventStatus, [38](#)
 - pxt, [17](#)
- ApplicationExecute
 - Afterwarp.h, [167](#)
- ApplicationFileChooserDialog
 - Afterwarp.h, [167](#)

- ApplicationGetClientRect
 - Afterwarp.h, [167](#)
- ApplicationGetCursor
 - Afterwarp.h, [167](#)
- ApplicationGetExecutablePath
 - Afterwarp.h, [167](#)
- ApplicationGetIconTitle
 - Afterwarp.h, [168](#)
- ApplicationGetMinimalSize
 - Afterwarp.h, [168](#)
- ApplicationGetTitle
 - Afterwarp.h, [168](#)
- ApplicationGetWindowHandle
 - Afterwarp.h, [168](#)
- ApplicationGetWindowRect
 - Afterwarp.h, [168](#)
- ApplicationGetWindowScale
 - Afterwarp.h, [169](#)
- ApplicationGetWindowState
 - Afterwarp.h, [169](#)
- ApplicationInvalidate
 - Afterwarp.h, [169](#)
- ApplicationMouseInputCaptured
 - Afterwarp.h, [169](#)
- ApplicationReadTextFromClipboard
 - Afterwarp.h, [169](#)
- ApplicationReleaseMouseInput
 - Afterwarp.h, [169](#)
- ApplicationSetClientSize
 - Afterwarp.h, [170](#)
- ApplicationSetCursor
 - Afterwarp.h, [170](#)
- ApplicationSetEvents
 - Afterwarp.h, [170](#)
- ApplicationSetIcons
 - Afterwarp.h, [170](#)
- ApplicationSetIconsFromFiles
 - Afterwarp.h, [170](#)
- ApplicationSetIconTitle
 - Afterwarp.h, [170](#)
- ApplicationSetMinimalSize
 - Afterwarp.h, [171](#)
- ApplicationSetTitle
 - Afterwarp.h, [171](#)
- ApplicationSetWindowRect
 - Afterwarp.h, [171](#)
- ApplicationSetWindowState
 - Afterwarp.h, [171](#)
- ApplicationTerminate
 - Afterwarp.h, [171](#)
- ApplicationTranslateVirtualKey
 - Afterwarp.h, [171](#)
- ApplicationWindowState
 - Afterwarp.Types.h, [395](#)
- ApplicationWriteTextToClipboard
 - Afterwarp.h, [172](#)
- attenuation
 - AmbientOcclusionParameters, [32](#)
 - attenuationEnd
 - SceneLight, [92](#)
 - attenuationStart
 - SceneLight, [92](#)
 - attributes
 - FontParameters, [55](#)
 - TextureParameters, [107](#)
 - WidgetProperty, [115](#)
 - AUTO_DRAW_OPTION_MATERIAL_IGNORE
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_MATERIAL_SKIP_OPAQUE
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_MATERIAL_SKIP_TRANSPARENT
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_MATERIAL_TRANSPARENCY
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_OBJECT_DISABLE_INSTANCING
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_OBJECT_HIGHLIGHTED
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_OBJECT_SKIP_HAVING_MATERIALS
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_OBJECT_SKIP_NON_TRANSPARENT
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_OBJECT_SKIP_NOT_HAVING_MATERIALS
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_OBJECT_SKIP_TRANSPARENT
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_POST_UNBIND
 - Afterwarp.Types.h, [426](#)
 - AUTO_DRAW_OPTION_RESET_TEXTURES
 - Afterwarp.Types.h, [426](#)
 - AverageColors
 - Afterwarp.h, [172](#)
 - AverageFourColors
 - Afterwarp.h, [172](#)
 - AverageSixColors
 - Afterwarp.h, [172](#)
 - behavior
 - WidgetProperty, [115](#)
 - bias
 - FogParameters, [49](#)
 - MeshAligns, [61](#)
 - ShadowParameters, [100](#)
 - biasLOD
 - SamplerState, [89](#)
 - BiasTransform
 - Afterwarp.h, [172](#)
 - bitmask
 - ObjectMaterial, [66](#)
 - OceanMaterial, [69](#)
 - SceneLight, [92](#)
 - bleedSigma
 - ShadowParameters, [100](#)
 - Blend, [38](#)
 - dest, [39](#)
 - op, [39](#)
 - source, [39](#)

- BLEND_FACTOR_CONSTANT_ALPHA
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_CONSTANT_COLOR
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_DEST_ALPHA
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_DEST_COLOR
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_INV_CONSTANT_ALPHA
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_INV_CONSTANT_COLOR
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_INV_DEST_ALPHA
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_INV_DEST_COLOR
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_INV_SOURCE_ALPHA
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_INV_SOURCE_ALPHA_1
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_INV_SOURCE_COLOR
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_INV_SOURCE_COLOR_1
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_ONE
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_SOURCE_ALPHA
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_SOURCE_ALPHA_1
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_SOURCE_ALPHA_SAT
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_SOURCE_COLOR
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_SOURCE_COLOR_1
 - Afterwarp.Types.h, [410](#)
- BLEND_FACTOR_ZERO
 - Afterwarp.Types.h, [410](#)
- BLEND_OP_ADD
 - Afterwarp.Types.h, [410](#)
- BLEND_OP_INV_SUBTRACT
 - Afterwarp.Types.h, [410](#)
- BLEND_OP_MAXIMUM
 - Afterwarp.Types.h, [410](#)
- BLEND_OP_MINIMUM
 - Afterwarp.Types.h, [410](#)
- BLEND_OP_SUBTRACT
 - Afterwarp.Types.h, [410](#)
- blendAlpha
 - RenderingState, [86](#)
- blendColor
 - RenderingState, [86](#)
- BlendColors
 - Afterwarp.h, [173](#)
- blendConstant
 - RenderingState, [86](#)
- BlendFactor
 - Afterwarp.Types.h, [395](#)
- BlendFourColors
 - Afterwarp.h, [173](#)
- BLENDING_EFFECT_ADD
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_COPY
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_CUSTOM
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_INVERSE_MULTIPLY
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_MULTIPLY
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_NORMAL
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_SHADOW
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_SOURCE_COLOR
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_SOURCE_COLOR_ADD
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_SUBTRACT
 - Afterwarp.Types.h, [415](#)
- BLENDING_EFFECT_UNDEFINED
 - Afterwarp.Types.h, [415](#)
- BlendingEffect
 - Afterwarp.Types.h, [395](#)
- BlendOp
 - Afterwarp.Types.h, [395](#)
- bloom
 - SceneMeshMaterialShading, [97](#)
- bloomBlurSamples
 - ToneMappingBloom, [109](#)
- bloomBlurSigma
 - ToneMappingBloom, [109](#)
- bloomCoefficients
 - ToneMappingBloom, [109](#)
- bloomColorShift
 - ToneMappingBloom, [109](#)
- bloomGamma
 - ToneMappingBloom, [110](#)
- bloomThreshold
 - ToneMappingBloom, [110](#)
- blue
 - FloatColor, [47](#)
 - FloatColorRGB, [48](#)
- blurFallOff
 - AmbientOcclusionParameters, [32](#)
- blurOffset
 - SelectionHighlightParameters, [99](#)
- blurPasses
 - SelectionHighlightParameters, [99](#)
- blurSamples
 - ShadowParameters, [100](#)
- blurSigma
 - AmbientOcclusionParameters, [32](#)
 - ShadowParameters, [100](#)
- BoolFunc
 - pxt, [17](#)

- borderBrightness
 - FontEffect, [51](#)
- borderColor
 - CanvasSamplerState, [42](#)
 - SamplerState, [89](#)
- borderOpacity
 - FontEffect, [51](#)
- borderThickness
 - FontEffect, [52](#)
- borderType
 - FontEffect, [52](#)
- bottom
 - Margins, [58](#)
- bottomLeft
 - ColorRect, [44](#)
 - Quad, [80](#)
- bottomRight
 - ColorRect, [44](#)
 - Quad, [80](#)
- boundsMax
 - MeshMetaTagPortion, [64](#)
- boundsMin
 - MeshMetaTagPortion, [64](#)
- BUFFER_ACCESS_TYPE_COMPUTE
 - Afterwarp.Types.h, [411](#)
- BUFFER_ACCESS_TYPE_DEFAULT
 - Afterwarp.Types.h, [411](#)
- BUFFER_ACCESS_TYPE_DYNAMIC
 - Afterwarp.Types.h, [411](#)
- BUFFER_ACCESS_TYPE_STAGING
 - Afterwarp.Types.h, [411](#)
- BUFFER_ACCESS_TYPE_STATIC
 - Afterwarp.Types.h, [411](#)
- BUFFER_DATA_TYPE_CONSTANT
 - Afterwarp.Types.h, [411](#)
- BUFFER_DATA_TYPE_INDEX
 - Afterwarp.Types.h, [411](#)
- BUFFER_DATA_TYPE_RW_STRUCTURED
 - Afterwarp.Types.h, [411](#)
- BUFFER_DATA_TYPE_RW_TYPED
 - Afterwarp.Types.h, [411](#)
- BUFFER_DATA_TYPE_STRUCTURED
 - Afterwarp.Types.h, [411](#)
- BUFFER_DATA_TYPE_TYPED
 - Afterwarp.Types.h, [411](#)
- BUFFER_DATA_TYPE_VERTEX
 - Afterwarp.Types.h, [411](#)
- Buffer_t
 - pxt, [17](#)
- BufferAccessType
 - Afterwarp.Types.h, [395](#)
- BufferCopy
 - Afterwarp.h, [173](#)
- BufferCreate
 - Afterwarp.h, [173](#)
- BufferDataType
 - Afterwarp.Types.h, [395](#)
- BufferDestroy
 - Afterwarp.h, [173](#)
- BufferGetAccessType
 - Afterwarp.h, [174](#)
- BufferGetDataType
 - Afterwarp.h, [174](#)
- BufferGetDevice
 - Afterwarp.h, [174](#)
- BufferGetFormat
 - Afterwarp.h, [174](#)
- BufferGetPitch
 - Afterwarp.h, [174](#)
- BufferGetPlatformHandle
 - Afterwarp.h, [174](#)
- BufferGetSize
 - Afterwarp.h, [174](#)
- BufferRetrieve
 - Afterwarp.h, [175](#)
- BufferUpdate
 - Afterwarp.h, [175](#)
- CAMERA_COMMAND_DRAGGING
 - Afterwarp.Types.h, [429](#)
- CAMERA_COMMAND_MOVEMENT
 - Afterwarp.Types.h, [429](#)
- CAMERA_COMMAND_ROTATION
 - Afterwarp.Types.h, [429](#)
- CAMERA_COMMAND_STOP
 - Afterwarp.Types.h, [429](#)
- CAMERA_COMMAND_UPDATE
 - Afterwarp.Types.h, [429](#)
- CameraCommand
 - Afterwarp.Types.h, [395](#)
- CameraConstraints, [40](#)
 - positionMax, [40](#)
 - positionMin, [40](#)
 - pxt, [17](#)
 - rotationMax, [41](#)
 - rotationMin, [41](#)
- CANVAS_ATTRIBUTE_COLOR_ADJUST
 - Afterwarp.Types.h, [416](#)
- CANVAS_ATTRIBUTE_CUBIC
 - Afterwarp.Types.h, [416](#)
- CANVAS_ATTRIBUTE_OUTLINE_SDF
 - Afterwarp.Types.h, [416](#)
- CANVAS_ATTRIBUTE_PROJECTED
 - Afterwarp.Types.h, [416](#)
- CANVAS_ATTRIBUTE_SDF
 - Afterwarp.Types.h, [416](#)
- CANVAS_ATTRIBUTE_TRANSFORM
 - Afterwarp.Types.h, [416](#)
- CANVAS_CONTEXT_STATE_FLAT_SCENE
 - Afterwarp.Types.h, [416](#)
- CANVAS_CONTEXT_STATE_FLAT_TEXT
 - Afterwarp.Types.h, [416](#)
- CANVAS_CONTEXT_STATE_PROJECTED
 - Afterwarp.Types.h, [416](#)
- CANVAS_CONTEXT_STATE_UNDEFINED
 - Afterwarp.Types.h, [416](#)
- Canvas_t

- pxt, [18](#)
- CanvasArc
 - Afterwarp.h, [175](#)
- CanvasArcGrad
 - Afterwarp.h, [175](#)
- CanvasAttributes
 - pxt, [18](#)
- CanvasBegin
 - Afterwarp.h, [176](#)
- CanvasBufferClear
 - Afterwarp.h, [176](#)
- CanvasBufferClearAndShrink
 - Afterwarp.h, [176](#)
- CanvasBufferCreate
 - Afterwarp.h, [176](#)
- CanvasBufferDestroy
 - Afterwarp.h, [176](#)
- CanvasBufferGetColors
 - Afterwarp.h, [176](#)
- CanvasBufferGetExtents
 - Afterwarp.h, [177](#)
- CanvasBufferGetIndexCount
 - Afterwarp.h, [177](#)
- CanvasBufferGetIndices
 - Afterwarp.h, [177](#)
- CanvasBufferGetVertexCount
 - Afterwarp.h, [177](#)
- CanvasBufferGetVertices
 - Afterwarp.h, [177](#)
- CanvasBufferSetIndexCount
 - Afterwarp.h, [177](#)
- CanvasBufferSetVertexCount
 - Afterwarp.h, [177](#)
- CanvasContextState
 - Afterwarp.Types.h, [396](#)
- CanvasCreate
 - Afterwarp.h, [178](#)
- CanvasDestroy
 - Afterwarp.h, [178](#)
- CanvasDrawBuffer
 - Afterwarp.h, [178](#)
- CanvasEllipse
 - Afterwarp.h, [178](#)
- CanvasEnd
 - Afterwarp.h, [178](#)
- CanvasFillRect
 - Afterwarp.h, [178](#)
- CanvasFillRoundRect
 - Afterwarp.h, [179](#)
- CanvasFillRoundRectBottom
 - Afterwarp.h, [179](#)
- CanvasFillRoundRectTop
 - Afterwarp.h, [179](#)
- CanvasFillRoundRectTopInverse
 - Afterwarp.h, [179](#)
- CanvasFlush
 - Afterwarp.h, [180](#)
- CanvasFrameRect
 - Afterwarp.h, [180](#)
- CanvasFrameRoundRect
 - Afterwarp.h, [180](#)
- CanvasGetAttributes
 - Afterwarp.h, [180](#)
- CanvasGetBatchCount
 - Afterwarp.h, [180](#)
- CanvasGetClipRect
 - Afterwarp.h, [181](#)
- CanvasGetContextState
 - Afterwarp.h, [181](#)
- CanvasGetDevice
 - Afterwarp.h, [181](#)
- CanvasGetSamplerState
 - Afterwarp.h, [181](#)
- CanvasGetSignedDistanceField
 - Afterwarp.h, [181](#)
- CanvasGetTransform
 - Afterwarp.h, [181](#)
- CanvasGetViewProjection
 - Afterwarp.h, [182](#)
- CanvasGetWorld
 - Afterwarp.h, [182](#)
- CanvasHexagon
 - Afterwarp.h, [182](#)
- CanvasHexagonGrad
 - Afterwarp.h, [182](#)
- CanvasHighlight
 - Afterwarp.h, [182](#)
- CanvasLine
 - Afterwarp.h, [183](#)
- CanvasLineCircle
 - Afterwarp.h, [183](#)
- CanvasLineEllipse
 - Afterwarp.h, [183](#)
- CanvasLineHexagon
 - Afterwarp.h, [183](#)
- CanvasLineHexagonGrad
 - Afterwarp.h, [184](#)
- CanvasLineQuad
 - Afterwarp.h, [184](#)
- CanvasLines
 - Afterwarp.h, [184](#)
- CanvasLineTriangle
 - Afterwarp.h, [184](#)
- CanvasPixel
 - Afterwarp.h, [185](#)
- CanvasPixels
 - Afterwarp.h, [185](#)
- CanvasQuad
 - Afterwarp.h, [185](#)
- CanvasQuadImage
 - Afterwarp.h, [185](#)
- CanvasRectWithHole
 - Afterwarp.h, [185](#)
- CanvasReset
 - Afterwarp.h, [186](#)
- CanvasResetSamplerState

- Afterwarp.h, [186](#)
- CanvasRibbon
 - Afterwarp.h, [186](#)
- CanvasRibbonGrad
 - Afterwarp.h, [186](#)
- CanvasRibbonTri
 - Afterwarp.h, [186](#)
- CanvasSamplerState, [41](#)
 - addressU, [42](#)
 - addressV, [42](#)
 - borderColor, [42](#)
 - filterMag, [42](#)
 - filterMin, [42](#)
 - filterMip, [42](#)
 - pxt, [18](#)
- CanvasSetAttributes
 - Afterwarp.h, [187](#)
- CanvasSetClipRect
 - Afterwarp.h, [187](#)
- CanvasSetContextState
 - Afterwarp.h, [187](#)
- CanvasSetSamplerState
 - Afterwarp.h, [187](#)
- CanvasSetSignedDistanceField
 - Afterwarp.h, [187](#)
- CanvasSetTransform
 - Afterwarp.h, [188](#)
- CanvasSetViewProjection
 - Afterwarp.h, [188](#)
- CanvasSetWorld
 - Afterwarp.h, [188](#)
- CanvasTape
 - Afterwarp.h, [188](#)
- CanvasTapeGrad
 - Afterwarp.h, [188](#)
- CanvasTapeTri
 - Afterwarp.h, [189](#)
- CanvasTexturedQuad
 - Afterwarp.h, [189](#)
- CanvasTexturedQuadRegion
 - Afterwarp.h, [189](#)
- CanvasTexturedRoundRect
 - Afterwarp.h, [189](#)
- CanvasTexturedRoundRectRegion
 - Afterwarp.h, [190](#)
- CanvasTexturedTriangle
 - Afterwarp.h, [190](#)
- CanvasTexturedTriangleRegion
 - Afterwarp.h, [190](#)
- CanvasTexturedTriangles
 - Afterwarp.h, [191](#)
- CanvasThickLine
 - Afterwarp.h, [191](#)
- CanvasThickLineCircle
 - Afterwarp.h, [191](#)
- CanvasThickLineEllipse
 - Afterwarp.h, [191](#)
- CanvasThickLineHexagon
 - Afterwarp.h, [192](#)
- CanvasThickLineHexagonGrad
 - Afterwarp.h, [192](#)
- CanvasThickLineQuad
 - Afterwarp.h, [192](#)
- CanvasThickLineTriangle
 - Afterwarp.h, [192](#)
- CanvasTriangle
 - Afterwarp.h, [193](#)
- CanvasTriangles
 - Afterwarp.h, [193](#)
- CatmullRom
 - Afterwarp.h, [193](#)
- channel
 - ComputeBindTextureFormat, [45](#)
 - ProgramElement, [77](#)
 - VertexElement, [113](#)
- choppiness
 - OceanWavesParameters, [71](#)
- clampDepthBias
 - RenderingState, [86](#)
- CLUSTERS_CULLING_MODE_PERFORMANCE
 - Afterwarp.Types.h, [421](#)
- CLUSTERS_CULLING_MODE_QUALITY
 - Afterwarp.Types.h, [421](#)
- ClustersCullingMode
 - Afterwarp.Types.h, [396](#)
- Color
 - Afterwarp.Types.h, [396](#)
- color
 - FogParameters, [49](#)
 - MeshBufferEntry, [62](#)
- COLOR_DITHERING_FORMAT_R5G6B5
 - Afterwarp.Types.h, [420](#)
- COLOR_DITHERING_FORMAT_RG3B2
 - Afterwarp.Types.h, [420](#)
- COLOR_DITHERING_FORMAT_RGBA
 - Afterwarp.Types.h, [420](#)
- COLOR_DITHERING_FORMAT_RGB6
 - Afterwarp.Types.h, [420](#)
- COLOR_DITHERING_FORMAT_RGB8
 - Afterwarp.Types.h, [420](#)
- colorBlack
 - Afterwarp.Vectors.h, [507](#)
- ColorDithering_t
 - pxt, [18](#)
- ColorDitheringCreate
 - Afterwarp.h, [193](#)
- ColorDitheringDestroy
 - Afterwarp.h, [194](#)
- ColorDitheringExecute
 - Afterwarp.h, [194](#)
- ColorDitheringExecuteTo
 - Afterwarp.h, [194](#)
- ColorDitheringFormat
 - Afterwarp.Types.h, [396](#)
- ColorDitheringGetDevice
 - Afterwarp.h, [194](#)

- ColorDitheringGetFormat
 - Afterwarp.h, [194](#)
- ColorDitheringSetFormat
 - Afterwarp.h, [194](#)
- ColorPair, [42](#)
 - Afterwarp.Types.h, [396](#)
 - ColorPair, [43](#)
 - first, [43](#)
 - second, [43](#)
- colorPair
 - Afterwarp.Vectors.h, [468](#)
- colorPairBlack
 - Afterwarp.Vectors.h, [507](#)
- colorPairTraslucentBlack
 - Afterwarp.Vectors.h, [507](#)
- colorPairTraslucentWhite
 - Afterwarp.Vectors.h, [507](#)
- colorPairWhite
 - Afterwarp.Vectors.h, [507](#)
- ColorRect, [44](#)
 - Afterwarp.Types.h, [396](#)
 - bottomLeft, [44](#)
 - bottomRight, [44](#)
 - topLeft, [44](#)
 - topRight, [45](#)
- colorRectBlack
 - Afterwarp.Vectors.h, [507](#)
- colorRectTraslucentBlack
 - Afterwarp.Vectors.h, [507](#)
- colorRectTraslucentWhite
 - Afterwarp.Vectors.h, [507](#)
- colorRectWhite
 - Afterwarp.Vectors.h, [508](#)
- ColorToGray
 - Afterwarp.h, [195](#)
- ColorToGray16
 - Afterwarp.h, [195](#)
- ColorToGrayF
 - Afterwarp.h, [195](#)
- colorTraslucentBlack
 - Afterwarp.Vectors.h, [508](#)
- colorTraslucentWhite
 - Afterwarp.Vectors.h, [508](#)
- colorWhite
 - Afterwarp.Vectors.h, [508](#)
- command
 - PathElement, [74](#)
- compareFunc
 - SamplerState, [89](#)
- compareToRef
 - SamplerState, [89](#)
- COMPARISON_FUNC_ALWAYS
 - Afterwarp.Types.h, [409](#)
- COMPARISON_FUNC_EQUAL
 - Afterwarp.Types.h, [409](#)
- COMPARISON_FUNC_GREATER
 - Afterwarp.Types.h, [409](#)
- COMPARISON_FUNC_GREATER_EQUAL
 - Afterwarp.Types.h, [409](#)
- COMPARISON_FUNC_LESS
 - Afterwarp.Types.h, [409](#)
- COMPARISON_FUNC_LESS_EQUAL
 - Afterwarp.Types.h, [409](#)
- COMPARISON_FUNC_NEVER
 - Afterwarp.Types.h, [409](#)
- COMPARISON_FUNC_NOT_EQUAL
 - Afterwarp.Types.h, [409](#)
- ComparisonFunc
 - Afterwarp.Types.h, [396](#)
- ComposeColors
 - Afterwarp.h, [195](#)
- COMPUTE_TEXTURE_ACCESS_READ
 - Afterwarp.Types.h, [413](#)
- COMPUTE_TEXTURE_ACCESS_READ_WRITE
 - Afterwarp.Types.h, [413](#)
- COMPUTE_TEXTURE_ACCESS_WRITE
 - Afterwarp.Types.h, [413](#)
- ComputeBindTextureFormat, [45](#)
 - access, [45](#)
 - channel, [45](#)
 - format, [46](#)
 - layer, [46](#)
 - mipLevel, [46](#)
 - pxt, [18](#)
- ComputeProgram_t
 - pxt, [18](#)
- ComputeProgramBegin
 - Afterwarp.h, [195](#)
- ComputeProgramBindBuffer
 - Afterwarp.h, [195](#)
- ComputeProgramBindTexture
 - Afterwarp.h, [196](#)
- ComputeProgramCommit
 - Afterwarp.h, [196](#)
- ComputeProgramCreate
 - Afterwarp.h, [196](#)
- ComputeProgramDestroy
 - Afterwarp.h, [196](#)
- ComputeProgramDispatch
 - Afterwarp.h, [196](#)
- ComputeProgramEnd
 - Afterwarp.h, [197](#)
- ComputeProgramGetDevice
 - Afterwarp.h, [197](#)
- ComputeProgramResetBindings
 - Afterwarp.h, [197](#)
- ComputeProgramUnbindBuffer
 - Afterwarp.h, [197](#)
- ComputeProgramUnbindTexture
 - Afterwarp.h, [197](#)
- ComputeTextureAccess
 - Afterwarp.Types.h, [397](#)
- contrast
 - AmbientOcclusionParameters, [32](#)
- count
 - ProgramVariable, [79](#)

- VertexElement, [113](#)
- Cubic
 - Afterwarp.h, [197](#)
- cullFace
 - RenderingState, [86](#)
- data
 - Matrix, [59](#)
 - Matrix3x2, [60](#)
 - WidgetProperty, [115](#)
- densityGround
 - FogParameters, [50](#)
- DEPTH_FOG_DISTANCE_EYE_RELATIVE
 - Afterwarp.Types.h, [421](#)
- DEPTH_FOG_DISTANCE_Z_BASED
 - Afterwarp.Types.h, [421](#)
- depthBias
 - AmbientOcclusionParameters, [32](#)
 - RenderingState, [86](#)
- depthFailOp
 - StencilState, [103](#)
- DepthFogDistance
 - Afterwarp.Types.h, [397](#)
- depthFunc
 - RenderingState, [86](#)
- depthPassOp
 - StencilState, [103](#)
- depthStencil
 - TextureParameters, [107](#)
- dest
 - Blend, [39](#)
- DEVICE_ATTRIBUTE_DEBUG
 - Afterwarp.Types.h, [406](#)
- DEVICE_ATTRIBUTE_LEGACY
 - Afterwarp.Types.h, [406](#)
- DEVICE_ATTRIBUTE_LIMITED_EXTENSIONS
 - Afterwarp.Types.h, [406](#)
- DEVICE_ATTRIBUTE_SOFTWARE
 - Afterwarp.Types.h, [406](#)
- DEVICE_BARRIER_ATOMIC_COUNTER
 - Afterwarp.Types.h, [407](#)
- DEVICE_BARRIER_BUFFER_UPDATE
 - Afterwarp.Types.h, [407](#)
- DEVICE_BARRIER_DRAWABLES
 - Afterwarp.Types.h, [407](#)
- DEVICE_BARRIER_SHADER_BUFFER
 - Afterwarp.Types.h, [406](#)
- DEVICE_BARRIER_SHADER_IMAGE_ACCESS
 - Afterwarp.Types.h, [407](#)
- DEVICE_BARRIER_STRUCTURED
 - Afterwarp.Types.h, [407](#)
- DEVICE_BARRIER_TEXTURE_FETCH
 - Afterwarp.Types.h, [407](#)
- DEVICE_BARRIER_TEXTURE_UPDATE
 - Afterwarp.Types.h, [407](#)
- DEVICE_BEHAVIOR_COMPUTE
 - Afterwarp.Types.h, [406](#)
- DEVICE_BEHAVIOR_DEPTH_CLEAR_BAD_PRECISION
 - Afterwarp.Types.h, [406](#)
- DEVICE_BEHAVIOR_DEPTH_CLIP_NEGATIVE
 - Afterwarp.Types.h, [406](#)
- DEVICE_BEHAVIOR_FORCE_BUFFER_UNBIND
 - Afterwarp.Types.h, [406](#)
- DEVICE_BEHAVIOR_PER_SAMPLE_SHADING
 - Afterwarp.Types.h, [406](#)
- DEVICE_BEHAVIOR_POST_DEPTH_COVERAGE
 - Afterwarp.Types.h, [406](#)
- DEVICE_BEHAVIOR_SIGNED_NORM_INT_FORMAT_BUGGED
 - Afterwarp.Types.h, [406](#)
- DEVICE_BEHAVIOR_TESSELLATION
 - Afterwarp.Types.h, [406](#)
- DEVICE_BEHAVIOR_VARIABLE_RATE_REFRESH
 - Afterwarp.Types.h, [406](#)
- DEVICE_CLEAR_LAYER_COLOR
 - Afterwarp.Types.h, [405](#)
- DEVICE_CLEAR_LAYER_DEPTH
 - Afterwarp.Types.h, [405](#)
- DEVICE_CLEAR_LAYER_STENCIL
 - Afterwarp.Types.h, [405](#)
- DEVICE_PLATFORM_ANDROID
 - Afterwarp.Types.h, [408](#)
- DEVICE_PLATFORM_IOS
 - Afterwarp.Types.h, [408](#)
- DEVICE_PLATFORM_LINUX
 - Afterwarp.Types.h, [408](#)
- DEVICE_PLATFORM_OSX
 - Afterwarp.Types.h, [408](#)
- DEVICE_PLATFORM_UNIX
 - Afterwarp.Types.h, [408](#)
- DEVICE_PLATFORM_UNKNOWN
 - Afterwarp.Types.h, [408](#)
- DEVICE_PLATFORM_WINDOWS
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_ALPHA_TO_COVERAGE
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_BLEND_ENABLE
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_COLOR_WRITE
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_CUBE_MAP_SEAMLESS
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_CULL_CLOCKWISE
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_DEPTH_CLIP
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_DEPTH_TEST
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_DEPTH_WRITE
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_LINE_ANTIALIAS
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_MULTISAMPLING
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_PER_SAMPLE_SHADING
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_SCISSOR_CLIP
 - Afterwarp.Types.h, [408](#)

- DEVICE_STATE_STENCIL_TEST
 - Afterwarp.Types.h, [408](#)
- DEVICE_STATE_WIREFRAME
 - Afterwarp.Types.h, [408](#)
- DEVICE_TECHNOLOGY_DIRECT3D
 - Afterwarp.Types.h, [407](#)
- DEVICE_TECHNOLOGY_METAL
 - Afterwarp.Types.h, [407](#)
- DEVICE_TECHNOLOGY_OPENGL
 - Afterwarp.Types.h, [407](#)
- DEVICE_TECHNOLOGY_OPENGL_ES
 - Afterwarp.Types.h, [407](#)
- DEVICE_TECHNOLOGY_PROPRIETARY
 - Afterwarp.Types.h, [407](#)
- DEVICE_TECHNOLOGY_SOFTWARE
 - Afterwarp.Types.h, [407](#)
- DEVICE_TECHNOLOGY_UNKNOWN
 - Afterwarp.Types.h, [407](#)
- DEVICE_TECHNOLOGY_VULKAN
 - Afterwarp.Types.h, [407](#)
- DEVICE_TECHNOLOGY_WEBGL
 - Afterwarp.Types.h, [407](#)
- deviceAttributeExtensions
 - pxt, [30](#)
- DeviceAttributes
 - pxt, [18](#)
- DeviceBehavior
 - pxt, [19](#)
- DeviceClear
 - Afterwarp.h, [198](#)
- DeviceCreate
 - Afterwarp.h, [198](#)
- DeviceCreateWrapped
 - Afterwarp.h, [198](#)
- DeviceDestroy
 - Afterwarp.h, [198](#)
- DeviceGetAttributes
 - Afterwarp.h, [198](#)
- DeviceGetBehavior
 - Afterwarp.h, [199](#)
- DeviceGetLegacyBits
 - Afterwarp.h, [199](#)
- DeviceGetPlatform
 - Afterwarp.h, [199](#)
- DeviceGetPlatformDevice
 - Afterwarp.h, [199](#)
- DeviceGetRenderingState
 - Afterwarp.h, [199](#)
- DeviceGetScissor
 - Afterwarp.h, [199](#)
- DeviceGetTechFeatureVersion
 - Afterwarp.h, [199](#)
- DeviceGetTechnology
 - Afterwarp.h, [200](#)
- DeviceGetTechVersion
 - Afterwarp.h, [200](#)
- DeviceGetViewport
 - Afterwarp.h, [200](#)
- DeviceLegacyBits
 - pxt, [19](#)
- DeviceMemoryBarrier
 - Afterwarp.h, [200](#)
- DevicePlatform
 - Afterwarp.Types.h, [397](#)
- DeviceResetCache
 - Afterwarp.h, [200](#)
- DeviceSetRenderingState
 - Afterwarp.h, [200](#)
- DeviceSetScissor
 - Afterwarp.h, [201](#)
- DeviceSetViewport
 - Afterwarp.h, [201](#)
- DeviceTechnology
 - Afterwarp.Types.h, [397](#)
- diffuse
 - SceneMeshMaterialShading, [97](#)
- direction
 - SceneLight, [92](#)
- DisplaceRB
 - Afterwarp.h, [201](#)
- distance
 - FogParameters, [50](#)
- effect
 - FontParameters, [55](#)
 - TextRenderModifiers, [105](#)
- element
 - ProgramElement, [77](#)
- ELEMENT_FORMAT_BYTE
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_DOUBLE
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_FLOAT
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_FLOAT_11_11_10
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_HALF_FLOAT
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_INT
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_SHORT
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_UNDEFINED
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_UNSIGNED_BYTE
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_UNSIGNED_INT
 - Afterwarp.Types.h, [412](#)
- ELEMENT_FORMAT_UNSIGNED_SHORT
 - Afterwarp.Types.h, [412](#)
- ElementFormat
 - Afterwarp.Types.h, [397](#)
- emissive
 - SceneMeshMaterialShading, [97](#)
- emissiveColor
 - ObjectMaterial, [66](#)
 - OceanMaterial, [69](#)

- eventCreate
 - ApplicationEvents, 37
- eventDestroy
 - ApplicationEvents, 37
- EventFunc
 - pxt, 19
- eventHook
 - ApplicationEvents, 37
- EventHookFunc
 - pxt, 19
- eventIdle
 - ApplicationEvents, 37
- eventKey
 - ApplicationEvents, 38
- eventMouse
 - ApplicationEvents, 38
- eventRender
 - ApplicationEvents, 38
- eventResize
 - ApplicationEvents, 38
- eventStatus
 - ApplicationEvents, 38
- exponent1
 - ShadowParameters, 100
- exponent2
 - ShadowParameters, 101
- extinction
 - FogParameters, 50
 - OceanMaterial, 69
- failOp
 - StencilState, 103
- family
 - FontParameters, 55
- FILE_CHOOSER_DIALOG_DIRECTORY
 - Afterwarp.Types.h, 431
- FILE_CHOOSER_DIALOG_OPEN_FILE
 - Afterwarp.Types.h, 431
- FILE_CHOOSER_DIALOG_SAVE_FILE
 - Afterwarp.Types.h, 431
- FileChooserDialog
 - Afterwarp.Types.h, 397
- fillBrightness
 - FontEffect, 52
- fillOpacity
 - FontEffect, 52
- filterMag
 - CanvasSamplerState, 42
 - SamplerState, 89
- filterMin
 - CanvasSamplerState, 42
 - SamplerState, 90
- filterMip
 - CanvasSamplerState, 42
 - SamplerState, 90
- first
 - ColorPair, 43
- firstIndex
 - MeshMetaTagPortion, 64
- firstVertex
 - MeshMetaTagPortion, 64
- Flip
 - ImageRegion, 56
- FloatColor, 46
 - Afterwarp.Types.h, 397
 - alpha, 47
 - blue, 47
 - green, 47
 - red, 47
- floatColor
 - Afterwarp.Vectors.h, 468
- floatColorAdd
 - Afterwarp.Vectors.h, 468
- floatColorAddRGB
 - Afterwarp.Vectors.h, 468
- floatColorAverage
 - Afterwarp.Vectors.h, 468
- floatColorAverageRGB
 - Afterwarp.Vectors.h, 469
- floatColorBlack
 - Afterwarp.Vectors.h, 508
- floatColorBlackRGB
 - Afterwarp.Vectors.h, 508
- floatColorDivide
 - Afterwarp.Vectors.h, 469
- floatColorDivideRGB
 - Afterwarp.Vectors.h, 469
- floatColorEmpty
 - Afterwarp.Vectors.h, 469
- floatColorEmptyRGB
 - Afterwarp.Vectors.h, 469
- floatColorEquals
 - Afterwarp.Vectors.h, 469
- floatColorEqualsRGB
 - Afterwarp.Vectors.h, 470
- floatColorFromValue
 - Afterwarp.Vectors.h, 470
- floatColorFromValueRGB
 - Afterwarp.Vectors.h, 470
- floatColorGray
 - Afterwarp.Vectors.h, 470
- floatColorGrayRGB
 - Afterwarp.Vectors.h, 470
- floatColorInverse
 - Afterwarp.Vectors.h, 470
- floatColorInverseRGB
 - Afterwarp.Vectors.h, 471
- floatColorLerp
 - Afterwarp.Vectors.h, 471
- floatColorLerpRGB
 - Afterwarp.Vectors.h, 471
- floatColorMultiply
 - Afterwarp.Vectors.h, 471
- floatColorMultiplyRGB
 - Afterwarp.Vectors.h, 471
- floatColorNegate
 - Afterwarp.Vectors.h, 471

- floatColorNegateRGB
 - Afterwarp.Vectors.h, [472](#)
- floatColorPremultiply
 - Afterwarp.Vectors.h, [472](#)
- FloatColorRGB, [47](#)
 - Afterwarp.Types.h, [398](#)
 - blue, [48](#)
 - green, [48](#)
 - red, [48](#)
- floatColorRGB
 - Afterwarp.Vectors.h, [472](#)
- floatColorSaturate
 - Afterwarp.Vectors.h, [472](#)
- floatColorSaturateRGB
 - Afterwarp.Vectors.h, [472](#)
- floatColorSubtract
 - Afterwarp.Vectors.h, [472](#)
- floatColorSubtractRGB
 - Afterwarp.Vectors.h, [472](#)
- floatColorToRGB
 - Afterwarp.Vectors.h, [473](#)
- floatColorTraslucentBlack
 - Afterwarp.Vectors.h, [508](#)
- floatColorTraslucentWhite
 - Afterwarp.Vectors.h, [508](#)
- floatColorUnpremultiply
 - Afterwarp.Vectors.h, [473](#)
- floatColorWhite
 - Afterwarp.Vectors.h, [509](#)
- floatColorWhiteRGB
 - Afterwarp.Vectors.h, [509](#)
- floatToColor
 - Afterwarp.Vectors.h, [473](#)
- floatToColorRGB
 - Afterwarp.Vectors.h, [473](#)
- FOG_FORMULA_EXPONENTIAL
 - Afterwarp.Types.h, [420](#)
- FOG_FORMULA_GROUND
 - Afterwarp.Types.h, [420](#)
- FOG_FORMULA_LINEAR
 - Afterwarp.Types.h, [420](#)
- FogFormula
 - Afterwarp.Types.h, [398](#)
- FogParameters, [48](#)
 - bias, [49](#)
 - color, [49](#)
 - densityGround, [50](#)
 - distance, [50](#)
 - extinction, [50](#)
 - groundPlane, [50](#)
 - opacity, [50](#)
 - pxt, [19](#)
 - scattering, [50](#)
- FONT_ATTRIBUTE_STRIKE_OUT
 - Afterwarp.Types.h, [419](#)
- FONT_ATTRIBUTE_UNDERLINE
 - Afterwarp.Types.h, [419](#)
- FONT_BORDER_HEAVY
 - Afterwarp.Types.h, [419](#)
- FONT_BORDER_NONE
 - Afterwarp.Types.h, [419](#)
- FONT_BORDER_NORMAL
 - Afterwarp.Types.h, [419](#)
- FONT_BORDER_SEMI_HEAVY
 - Afterwarp.Types.h, [419](#)
- FONT_SLANT_ITALIC
 - Afterwarp.Types.h, [419](#)
- FONT_SLANT_NONE
 - Afterwarp.Types.h, [419](#)
- FONT_SLANT_OBLIQUE
 - Afterwarp.Types.h, [419](#)
- FONT_STRETCH_CONDENSED
 - Afterwarp.Types.h, [418](#)
- FONT_STRETCH_EXPANDED
 - Afterwarp.Types.h, [418](#)
- FONT_STRETCH_EXTRA_CONDENSED
 - Afterwarp.Types.h, [418](#)
- FONT_STRETCH_EXTRA_EXPANDED
 - Afterwarp.Types.h, [418](#)
- FONT_STRETCH_NORMAL
 - Afterwarp.Types.h, [418](#)
- FONT_STRETCH_SEMI_CONDENSED
 - Afterwarp.Types.h, [418](#)
- FONT_STRETCH_SEMI_EXPANDED
 - Afterwarp.Types.h, [418](#)
- FONT_STRETCH_ULTRA_CONDENSED
 - Afterwarp.Types.h, [418](#)
- FONT_STRETCH_ULTRA_EXPANDED
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_BOLD
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_EXTRA_BOLD
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_EXTRA_HEAVY
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_EXTRA_LIGHT
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_HEAVY
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_LIGHT
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_MEDIUM
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_NORMAL
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_SEMI_BOLD
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_SEMI_LIGHT
 - Afterwarp.Types.h, [418](#)
- FONT_WEIGHT_THIN
 - Afterwarp.Types.h, [418](#)
- FontAttributes
 - Afterwarp.Types.h, [398](#)
- FontBorder
 - Afterwarp.Types.h, [398](#)
- FontEffect, [51](#)

- borderBrightness, [51](#)
- borderOpacity, [51](#)
- borderThickness, [52](#)
- borderType, [52](#)
- fillBrightness, [52](#)
- fillOpacity, [52](#)
- pxt, [19](#)
- shadowBrightness, [52](#)
- shadowDistance, [52](#)
- shadowOpacity, [52](#)
- shadowSmoothness, [53](#)
- signedFieldDistance, [53](#)
- fontEffectDefault
 - Afterwarp.Vectors.h, [509](#)
- FontParameters, [53](#)
 - attributes, [55](#)
 - effect, [55](#)
 - family, [55](#)
 - FontParameters, [54](#)
 - FontSettings, [54](#)
 - pxt, [19](#)
 - size, [55](#)
 - slant, [55](#)
 - stretch, [55](#)
 - weight, [55](#)
- FontSettings
 - FontParameters, [54](#)
- FontSlant
 - Afterwarp.Types.h, [398](#)
- FontStretch
 - Afterwarp.Types.h, [398](#)
- FontWeight
 - Afterwarp.Types.h, [398](#)
- format
 - ComputeBindTextureFormat, [46](#)
 - ProgramVariable, [79](#)
 - TextureParameters, [107](#)
 - VertexElement, [113](#)
- fresnel
 - OceanMaterial, [69](#)
- frostedGlass
 - ObjectMaterial, [67](#)
- frostedPower
 - ToneMappingBloom, [110](#)
- func
 - StencilState, [104](#)
- GainTransform
 - Afterwarp.h, [201](#)
- GaussianBlur_t
 - pxt, [19](#)
- GaussianBlurCreate
 - Afterwarp.h, [201](#)
- GaussianBlurDestroy
 - Afterwarp.h, [201](#)
- GaussianBlurGetChroma
 - Afterwarp.h, [202](#)
- GaussianBlurGetDevice
 - Afterwarp.h, [202](#)
- GaussianBlurGetFixedSamples
 - Afterwarp.h, [202](#)
- GaussianBlurGetHardwareFiltering
 - Afterwarp.h, [202](#)
- GaussianBlurGetSamples
 - Afterwarp.h, [202](#)
- GaussianBlurGetSigma
 - Afterwarp.h, [202](#)
- GaussianBlurSetChroma
 - Afterwarp.h, [202](#)
- GaussianBlurSetHardwareFiltering
 - Afterwarp.h, [203](#)
- GaussianBlurSetSamples
 - Afterwarp.h, [203](#)
- GaussianBlurSetSigma
 - Afterwarp.h, [203](#)
- GaussianBlurUpdate
 - Afterwarp.h, [203](#)
- GaussianBlurUpdateAt
 - Afterwarp.h, [203](#)
- GetColorAlpha
 - Afterwarp.h, [204](#)
- GetColorAlphaF
 - Afterwarp.h, [204](#)
- GetSystemTicks
 - Afterwarp.h, [204](#)
- glassyBuckets
 - ToneMappingBloom, [110](#)
- glowIntensity
 - SelectionHighlightParameters, [99](#)
- Grapher_t
 - pxt, [20](#)
- GrapherArrow
 - Afterwarp.h, [204](#)
- GrapherBegin
 - Afterwarp.h, [204](#)
- GrapherBoundingBox
 - Afterwarp.h, [205](#)
- GrapherCreate
 - Afterwarp.h, [205](#)
- GrapherDestroy
 - Afterwarp.h, [205](#)
- GrapherDottedLine
 - Afterwarp.h, [205](#)
- GrapherEnd
 - Afterwarp.h, [205](#)
- GrapherFlush
 - Afterwarp.h, [206](#)
- GrapherGetBatchCount
 - Afterwarp.h, [206](#)
- GrapherGetDevice
 - Afterwarp.h, [206](#)
- GrapherGetTransform
 - Afterwarp.h, [206](#)
- GrapherLine
 - Afterwarp.h, [206](#)
- GrapherLines
 - Afterwarp.h, [206](#)

- GrapherPoint
 - Afterwarp.h, [207](#)
- GrapherPoints
 - Afterwarp.h, [207](#)
- GrapherReset
 - Afterwarp.h, [207](#)
- GrapherSetTransform
 - Afterwarp.h, [207](#)
- green
 - FloatColor, [47](#)
 - FloatColorRGB, [48](#)
- groundPlane
 - FogParameters, [50](#)
- group
 - SceneMeshLatch, [94](#)
- handleIconBig
 - ApplicationConfiguration, [34](#)
- handleIconSmall
 - ApplicationConfiguration, [34](#)
- handleInstance
 - ApplicationConfiguration, [34](#)
- height
 - ImageRegion, [57](#)
 - Rect, [83](#)
 - RectF, [84](#)
 - TextureParameters, [107](#)
- Hermite
 - Afterwarp.h, [207](#)
- IMAGE_REGION_FLIP
 - Afterwarp.Types.h, [417](#)
- IMAGE_REGION_MIRROR
 - Afterwarp.Types.h, [417](#)
- IMAGE_REGION_ROTATE
 - Afterwarp.Types.h, [417](#)
- ImageAtlas_t
 - pxt, [20](#)
- ImageAtlasClearRegions
 - Afterwarp.h, [208](#)
- ImageAtlasClearTextures
 - Afterwarp.h, [208](#)
- ImageAtlasCreate
 - Afterwarp.h, [208](#)
- ImageAtlasCreateRegion
 - Afterwarp.h, [208](#)
- ImageAtlasCreateTexture
 - Afterwarp.h, [208](#)
- ImageAtlasDestroy
 - Afterwarp.h, [209](#)
- ImageAtlasGetDevice
 - Afterwarp.h, [209](#)
- ImageAtlasMakeRegions
 - Afterwarp.h, [209](#)
- ImageAtlasPackRegion
 - Afterwarp.h, [209](#)
- ImageAtlasPackSurface
 - Afterwarp.h, [209](#)
- ImageAtlasRegion
 - Afterwarp.h, [210](#)
- ImageAtlasRegionCount
 - Afterwarp.h, [210](#)
- ImageAtlasRemoveRegion
 - Afterwarp.h, [210](#)
- ImageAtlasRemoveTexture
 - Afterwarp.h, [210](#)
- ImageAtlasTexture
 - Afterwarp.h, [210](#)
- ImageAtlasTextureCount
 - Afterwarp.h, [210](#)
- ImageRegion, [56](#)
 - Flip, [56](#)
 - height, [57](#)
 - index, [57](#)
 - left, [57](#)
 - Mirror, [56](#)
 - pxt, [20](#)
 - Rotate, [56](#)
 - top, [57](#)
 - width, [57](#)
- index
 - ImageRegion, [57](#)
 - ProgramElement, [77](#)
 - ProgramVariable, [79](#)
- indexCount
 - MeshMetaTagPortion, [64](#)
 - SceneMeshMaterialRange, [95](#)
- indexStart
 - SceneMeshMaterialRange, [95](#)
- intensity
 - SceneLight, [92](#)
- interleave
 - TextRenderModifiers, [105](#)
- InvertColor
 - Afterwarp.h, [211](#)
- KawaseBlur_t
 - pxt, [20](#)
- KawaseBlurCreate
 - Afterwarp.h, [211](#)
- KawaseBlurDestroy
 - Afterwarp.h, [211](#)
- KawaseBlurGetDevice
 - Afterwarp.h, [211](#)
- KawaseBlurGetOffset
 - Afterwarp.h, [211](#)
- KawaseBlurGetPasses
 - Afterwarp.h, [211](#)
- KawaseBlurSetOffset
 - Afterwarp.h, [211](#)
- KawaseBlurSetPasses
 - Afterwarp.h, [212](#)
- KawaseBlurSinglePass
 - Afterwarp.h, [212](#)
- KawaseBlurUpdate
 - Afterwarp.h, [212](#)
- kernelBias
 - AmbientOcclusionParameters, [32](#)

- Key
 - Afterwarp.Types.h, [399](#), [435](#)
- KEY_ADD
 - Afterwarp.Types.h, [437](#)
- KEY_ALT_LEFT
 - Afterwarp.Types.h, [436](#)
- KEY_ALT_RIGHT
 - Afterwarp.Types.h, [436](#)
- KEY_APPS
 - Afterwarp.Types.h, [437](#)
- KEY_BACKSPACE
 - Afterwarp.Types.h, [436](#)
- KEY_CANCEL
 - Afterwarp.Types.h, [436](#)
- KEY_CAPSLOCK
 - Afterwarp.Types.h, [436](#)
- KEY_CLEAR
 - Afterwarp.Types.h, [436](#)
- KEY_CTRL_LEFT
 - Afterwarp.Types.h, [436](#)
- KEY_CTRL_RIGHT
 - Afterwarp.Types.h, [436](#)
- KEY_DECIMAL
 - Afterwarp.Types.h, [437](#)
- KEY_DELETE
 - Afterwarp.Types.h, [436](#)
- KEY_DIVIDE
 - Afterwarp.Types.h, [437](#)
- KEY_DOWN
 - Afterwarp.Types.h, [436](#)
- KEY_END
 - Afterwarp.Types.h, [435](#)
- KEY_ESCAPE
 - Afterwarp.Types.h, [436](#)
- KEY_EVENT_PRESSED
 - Afterwarp.Types.h, [430](#)
- KEY_EVENT_RELEASED
 - Afterwarp.Types.h, [430](#)
- KEY_EVENT_TYPING
 - Afterwarp.Types.h, [430](#)
- KEY_EXECUTE
 - Afterwarp.Types.h, [437](#)
- KEY_F1
 - Afterwarp.Types.h, [436](#)
- KEY_F10
 - Afterwarp.Types.h, [436](#)
- KEY_F11
 - Afterwarp.Types.h, [436](#)
- KEY_F12
 - Afterwarp.Types.h, [436](#)
- KEY_F2
 - Afterwarp.Types.h, [436](#)
- KEY_F3
 - Afterwarp.Types.h, [436](#)
- KEY_F4
 - Afterwarp.Types.h, [436](#)
- KEY_F5
 - Afterwarp.Types.h, [436](#)
- KEY_F6
 - Afterwarp.Types.h, [436](#)
- KEY_F7
 - Afterwarp.Types.h, [436](#)
- KEY_F8
 - Afterwarp.Types.h, [436](#)
- KEY_F9
 - Afterwarp.Types.h, [436](#)
- KEY_HELP
 - Afterwarp.Types.h, [437](#)
- KEY_HOME
 - Afterwarp.Types.h, [435](#)
- KEY_INSERT
 - Afterwarp.Types.h, [436](#)
- KEY_LEFT
 - Afterwarp.Types.h, [436](#)
- KEY_LINEFEED
 - Afterwarp.Types.h, [436](#)
- KEY_MEDIA_NEXT_TRACK
 - Afterwarp.Types.h, [437](#)
- KEY_MEDIA_PLAY_PAUSE
 - Afterwarp.Types.h, [437](#)
- KEY_MEDIA_PREV_TRACK
 - Afterwarp.Types.h, [437](#)
- KEY_MEDIA_STOP
 - Afterwarp.Types.h, [437](#)
- KEY_MULTIPLY
 - Afterwarp.Types.h, [437](#)
- KEY_NULL
 - Afterwarp.Types.h, [435](#)
- KEY_NUM_LOCK
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_0
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_1
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_2
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_3
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_4
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_5
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_6
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_7
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_8
 - Afterwarp.Types.h, [436](#)
- KEY_NUMPAD_9
 - Afterwarp.Types.h, [437](#)
- KEY_PAGE_DOWN
 - Afterwarp.Types.h, [436](#)
- KEY_PAGE_UP
 - Afterwarp.Types.h, [436](#)
- KEY_PAUSE
 - Afterwarp.Types.h, [436](#)

- KEY_PRINT
 - Afterwarp.Types.h, [437](#)
- KEY_PRINT_SCREEN
 - Afterwarp.Types.h, [435](#)
- KEY_RETURN
 - Afterwarp.Types.h, [436](#)
- KEY_RIGHT
 - Afterwarp.Types.h, [436](#)
- KEY_SCROLL_LOCK
 - Afterwarp.Types.h, [436](#)
- KEY_SELECT
 - Afterwarp.Types.h, [437](#)
- KEY_SEPARATOR
 - Afterwarp.Types.h, [437](#)
- KEY_SHIFT_LEFT
 - Afterwarp.Types.h, [436](#)
- KEY_SHIFT_RIGHT
 - Afterwarp.Types.h, [436](#)
- KEY_SLEEP
 - Afterwarp.Types.h, [437](#)
- KEY_SPACE
 - Afterwarp.Types.h, [436](#)
- KEY_SUBTRACT
 - Afterwarp.Types.h, [437](#)
- KEY_SUPER_LEFT
 - Afterwarp.Types.h, [436](#)
- KEY_SUPER_RIGHT
 - Afterwarp.Types.h, [436](#)
- KEY_SYS_REQ
 - Afterwarp.Types.h, [436](#)
- KEY_TAB
 - Afterwarp.Types.h, [436](#)
- KEY_UP
 - Afterwarp.Types.h, [436](#)
- KEY_VOLUME_DOWN
 - Afterwarp.Types.h, [437](#)
- KEY_VOLUME_MUTE
 - Afterwarp.Types.h, [437](#)
- KEY_VOLUME_UP
 - Afterwarp.Types.h, [437](#)
- KeyboardFunc
 - pxt, [20](#)
- KeyEvent
 - Afterwarp.Types.h, [399](#)
- layer
 - ComputeBindTextureFormat, [46](#)
- layers
 - TextureParameters, [107](#)
- left
 - ImageRegion, [57](#)
 - Margins, [58](#)
 - Rect, [83](#)
 - RectF, [84](#)
- LEGACY_BIT_DEPTH_FORMAT_RAWZ
 - Afterwarp.Types.h, [405](#)
- LEGACY_BIT_DIFFERENT_BITDEPTH_MRT
 - Afterwarp.Types.h, [405](#)
- LEGACY_BIT_MISSING_DEPTH_TEXTURE
 - Afterwarp.Types.h, [405](#)
- length
 - PathElement, [74](#)
- Lerp
 - Afterwarp.h, [212](#)
- LibraryBool
 - pxt, [20](#)
- LibraryGetVersion
 - Afterwarp.h, [212](#)
- LibrarySerialCode
 - Afterwarp.h, [213](#)
- lighting
 - SceneMeshMaterialShading, [97](#)
- LINE_CAPS_BUTT
 - Afterwarp.Types.h, [417](#)
- LINE_CAPS_ROUND
 - Afterwarp.Types.h, [417](#)
- LINE_CAPS_SQUARE
 - Afterwarp.Types.h, [417](#)
- LineCaps
 - Afterwarp.Types.h, [399](#)
- location
 - WidgetProperty, [115](#)
- makeBounds
 - Afterwarp.Vectors.h, [473](#)
- makeBoundsF
 - Afterwarp.Vectors.h, [473](#)
- MakeColor
 - Afterwarp.h, [213](#)
- MakeColorAlpha
 - Afterwarp.h, [213](#)
- MakeColorAlphaF
 - Afterwarp.h, [213](#)
- MakeColorF
 - Afterwarp.h, [213](#)
- MakeColorGray
 - Afterwarp.h, [213](#)
- MakeColorGrayF
 - Afterwarp.h, [214](#)
- makeColorPair
 - Afterwarp.Vectors.h, [474](#)
- MakeColorRGB
 - Afterwarp.h, [214](#)
- MakeColorRGBF
 - Afterwarp.h, [214](#)
- MakeColorWithGray
 - Afterwarp.h, [214](#)
- MakeColorWithGrayF
 - Afterwarp.h, [214](#)
- makeFloatColor
 - Afterwarp.Vectors.h, [474](#)
- makeFloatColorRGB
 - Afterwarp.Vectors.h, [474](#)
- makePoint
 - Afterwarp.Vectors.h, [474](#)
- makePointF
 - Afterwarp.Vectors.h, [474](#)
- makeQuad

- Afterwarp.Vectors.h, [474](#)
- makeQuadRotated
 - Afterwarp.Vectors.h, [475](#)
- makeQuadRotatedAt
 - Afterwarp.Vectors.h, [475](#)
- makeQuadRotatedTL
 - Afterwarp.Vectors.h, [475](#)
- makeQuadScaled
 - Afterwarp.Vectors.h, [475](#)
- makeQuadSkewedHoriz
 - Afterwarp.Vectors.h, [476](#)
- makeQuadSkewedVert
 - Afterwarp.Vectors.h, [476](#)
- makeQuadWith
 - Afterwarp.Vectors.h, [476](#)
- makeRect
 - Afterwarp.Vectors.h, [476](#)
- makeRectF
 - Afterwarp.Vectors.h, [476](#)
- makeVector
 - Afterwarp.Vectors.h, [477](#)
- makeVector3D
 - Afterwarp.Vectors.h, [477](#)
- Margins, [57](#)
 - Afterwarp.Types.h, [399](#)
 - bottom, [58](#)
 - left, [58](#)
 - right, [58](#)
 - top, [58](#)
- Matrix, [59](#)
 - Afterwarp.Types.h, [399](#)
 - data, [59](#)
- Matrix3x2, [59](#)
 - Afterwarp.Types.h, [399](#)
 - data, [60](#)
- matrixAdjoint
 - Afterwarp.Vectors.h, [477](#)
- matrixDeterminant
 - Afterwarp.Vectors.h, [477](#)
- matrixDeterminant3x2
 - Afterwarp.Vectors.h, [477](#)
- matrixEmpty
 - Afterwarp.Vectors.h, [477](#)
- matrixEmpty3x2
 - Afterwarp.Vectors.h, [478](#)
- matrixEquals
 - Afterwarp.Vectors.h, [478](#)
- matrixEquals3x2
 - Afterwarp.Vectors.h, [478](#)
- matrixEyePosition
 - Afterwarp.Vectors.h, [478](#)
- matrixGet3x2
 - Afterwarp.Vectors.h, [478](#)
- matrixHeadingPitchBank
 - Afterwarp.Vectors.h, [478](#)
- matrixIdentity
 - Afterwarp.Vectors.h, [509](#)
- matrixIdentity3x2
 - Afterwarp.Vectors.h, [509](#)
- matrixInverse
 - Afterwarp.Vectors.h, [479](#)
- matrixInverse3x2
 - Afterwarp.Vectors.h, [479](#)
- matrixInverseTranspose3x2
 - Afterwarp.Vectors.h, [479](#)
- matrixLookAt
 - Afterwarp.Vectors.h, [479](#)
- matrixMultiply
 - Afterwarp.Vectors.h, [479](#)
- matrixMultiply3x2
 - Afterwarp.Vectors.h, [479](#)
- matrixOrthogonalBDS
 - Afterwarp.Vectors.h, [480](#)
- matrixOrthogonalVOL
 - Afterwarp.Vectors.h, [480](#)
- matrixPerspectiveBDS
 - Afterwarp.Vectors.h, [480](#)
- matrixPerspectiveFOVX
 - Afterwarp.Vectors.h, [480](#)
- matrixPerspectiveFOVY
 - Afterwarp.Vectors.h, [480](#)
- matrixPerspectiveVOL
 - Afterwarp.Vectors.h, [481](#)
- matrixQuaternion
 - Afterwarp.Vectors.h, [481](#)
- matrixReflect
 - Afterwarp.Vectors.h, [481](#)
- matrixRescale
 - Afterwarp.Vectors.h, [481](#)
- matrixRescale3x2
 - Afterwarp.Vectors.h, [481](#)
- matrixRotate
 - Afterwarp.Vectors.h, [482](#)
- matrixRotate3x2
 - Afterwarp.Vectors.h, [482](#)
- matrixRotateBy3x2
 - Afterwarp.Vectors.h, [482](#)
- matrixRotateX
 - Afterwarp.Vectors.h, [482](#)
- matrixRotateY
 - Afterwarp.Vectors.h, [482](#)
- matrixRotateZ
 - Afterwarp.Vectors.h, [482](#)
- matrixScale
 - Afterwarp.Vectors.h, [483](#)
- matrixScale3x2
 - Afterwarp.Vectors.h, [483](#)
- matrixScaleBy3x2
 - Afterwarp.Vectors.h, [483](#)
- matrixSub3x3
 - Afterwarp.Vectors.h, [483](#)
- matrixTranslate
 - Afterwarp.Vectors.h, [483](#)
- matrixTranslate3x2
 - Afterwarp.Vectors.h, [483](#)
- matrixTranspose

- Afterwarp.Vectors.h, [483](#)
- matrixWorldPosition
 - Afterwarp.Vectors.h, [484](#)
- matrixYawPitchRoll
 - Afterwarp.Vectors.h, [484](#)
- matrixZero
 - Afterwarp.Vectors.h, [509](#)
- matrixZero3x2
 - Afterwarp.Vectors.h, [509](#)
- MAX
 - Afterwarp.Vectors.h, [468](#)
- maxLOD
 - SamplerState, [90](#)
- MESH_ALIGN_NEGATIVE
 - Afterwarp.Types.h, [427](#)
- MESH_ALIGN_ORIGIN
 - Afterwarp.Types.h, [427](#)
- MESH_ALIGN_POSITIVE
 - Afterwarp.Types.h, [427](#)
- MESH_ALIGN_UNALIGNED
 - Afterwarp.Types.h, [427](#)
- MESH_LOADING_OPTION_EXPORT_COLORS
 - Afterwarp.Types.h, [428](#)
- MESH_LOADING_OPTION_EXPORT_NORMALS
 - Afterwarp.Types.h, [428](#)
- MESH_LOADING_OPTION_EXPORT_TANGENTS
 - Afterwarp.Types.h, [428](#)
- MESH_LOADING_OPTION_EXPORT_TEXTURE_COORDINATES
 - Afterwarp.Types.h, [428](#)
- MESH_LOADING_OPTION_EXPORT_WEIGHTS
 - Afterwarp.Types.h, [428](#)
- MESH_LOADING_OPTION_MATERIAL_TEXTURE_MISSING
 - Afterwarp.Types.h, [428](#)
- MESH_LOADING_OPTION_MATERIAL_VERTEX_COLOR
 - Afterwarp.Types.h, [428](#)
- MESH_LOADING_OPTION_MATERIAL_VERTEX_COLOR_PREFERRED
 - Afterwarp.Types.h, [428](#)
- MESH_LOADING_OPTION_STRIP_GEOMETRY_NAMES
 - Afterwarp.Types.h, [428](#)
- MESH_META_TAG_TYPE_GEOMETRY
 - Afterwarp.Types.h, [425](#)
- MESH_META_TAG_TYPE_INDETERMINATE
 - Afterwarp.Types.h, [425](#)
- MESH_META_TAG_TYPE_OBJECT
 - Afterwarp.Types.h, [425](#)
- MeshAlign
 - Afterwarp.Types.h, [399](#)
- MeshAligns, [60](#)
 - bias, [61](#)
 - MeshAligns, [60](#)
 - pxt, [20](#)
 - x, [61](#)
 - y, [61](#)
 - z, [61](#)
- MeshBoundsTagOffset
 - Afterwarp.h, [215](#)
- MeshBoundsToMatrixModel
 - Afterwarp.h, [215](#)
- MeshBoundsToMatrixVolume
 - Afterwarp.h, [215](#)
- MeshBoundsToMatrixVolumeTag
 - Afterwarp.h, [215](#)
- MeshBuffer_t
 - pxt, [21](#)
- MeshBufferCalculateBounds
 - Afterwarp.h, [215](#)
- MeshBufferCalculateFlatNormals
 - Afterwarp.h, [216](#)
- MeshBufferCalculateNormals
 - Afterwarp.h, [216](#)
- MeshBufferCalculateNormalsWeld
 - Afterwarp.h, [216](#)
- MeshBufferCentralize
 - Afterwarp.h, [216](#)
- MeshBufferClear
 - Afterwarp.h, [216](#)
- MeshBufferCombine
 - Afterwarp.h, [217](#)
- MeshBufferConeSimple
 - Afterwarp.h, [217](#)
- MeshBufferCreate
 - Afterwarp.h, [217](#)
- MeshBufferCreateModel
 - Afterwarp.h, [217](#)
- MeshBufferCube
 - Afterwarp.h, [217](#)
- MeshBufferCubeMinimal
 - Afterwarp.h, [218](#)
- MeshBufferCubeRound
 - Afterwarp.h, [218](#)
- MeshBufferCylinder
 - Afterwarp.h, [218](#)
- MeshBufferDestroy
 - Afterwarp.h, [219](#)
- MeshBufferDisc
 - Afterwarp.h, [219](#)
- MeshBufferEliminateUnusedVertices
 - Afterwarp.h, [219](#)
- MeshBufferEntry, [62](#)
 - color, [62](#)
 - normal, [62](#)
 - position, [63](#)
 - pxt, [21](#)
 - tangent, [63](#)
 - texCoord, [63](#)
- MeshBufferExtrusion
 - Afterwarp.h, [219](#)
- MeshBufferFrustumVolume
 - Afterwarp.h, [220](#)
- MeshBufferGeosphere
 - Afterwarp.h, [220](#)
- MeshBufferGetIndexCount
 - Afterwarp.h, [220](#)
- MeshBufferGetIndices
 - Afterwarp.h, [220](#)
- MeshBufferGetTransform

- Afterwarp.h, [220](#)
- MeshBufferGetVertexCount
 - Afterwarp.h, [221](#)
- MeshBufferGetVertices
 - Afterwarp.h, [221](#)
- MeshBufferInvertIndexOrder
 - Afterwarp.h, [221](#)
- MeshBufferInvertNormals
 - Afterwarp.h, [221](#)
- MeshBufferJoinDuplicateVertices
 - Afterwarp.h, [221](#)
- MeshBufferLoadFromFile
 - Afterwarp.h, [221](#)
- MeshBufferLoadFromFileEx
 - Afterwarp.h, [222](#)
- MeshBufferPlane
 - Afterwarp.h, [222](#)
- MeshBufferSaveToFile
 - Afterwarp.h, [222](#)
- MeshBufferSaveToFileEx
 - Afterwarp.h, [223](#)
- MeshBufferSetIndexCount
 - Afterwarp.h, [223](#)
- MeshBufferSetTransform
 - Afterwarp.h, [223](#)
- MeshBufferSetVertexCount
 - Afterwarp.h, [223](#)
- MeshBufferSuperEllipse
 - Afterwarp.h, [223](#)
- MeshBufferSupertoroid
 - Afterwarp.h, [224](#)
- MeshBufferTorus
 - Afterwarp.h, [224](#)
- MeshBufferTorusKnot
 - Afterwarp.h, [225](#)
- MeshBufferTransferIndexElements
 - Afterwarp.h, [225](#)
- MeshBufferTransferIndices
 - Afterwarp.h, [225](#)
- MeshBufferTransferVertexElements
 - Afterwarp.h, [226](#)
- MeshBufferTransferVertices
 - Afterwarp.h, [226](#)
- MeshBufferTransformVertices
 - Afterwarp.h, [226](#)
- MeshBufferVoxelize
 - Afterwarp.h, [226](#)
- MeshLoadSaveFeedback
 - pxt, [21](#)
- MeshMetaTag_t
 - pxt, [21](#)
- MeshMetaTagGetBounds
 - Afterwarp.h, [227](#)
- MeshMetaTagGetName
 - Afterwarp.h, [227](#)
- MeshMetaTagGetParent
 - Afterwarp.h, [227](#)
- MeshMetaTagGetPortionCount
 - Afterwarp.h, [227](#)
- MeshMetaTagGetType
 - Afterwarp.h, [227](#)
- MeshMetaTagPortion, [63](#)
 - boundsMax, [64](#)
 - boundsMin, [64](#)
 - firstIndex, [64](#)
 - firstVertex, [64](#)
 - indexCount, [64](#)
 - pxt, [21](#)
 - vertexCount, [65](#)
- MeshMetaTagPortionAdd
 - Afterwarp.h, [227](#)
- MeshMetaTagPortionErase
 - Afterwarp.h, [228](#)
- MeshMetaTagPortionGet
 - Afterwarp.h, [228](#)
- MeshMetaTagPortionsClear
 - Afterwarp.h, [228](#)
- MeshMetaTagPortionsCopy
 - Afterwarp.h, [228](#)
- MeshMetaTags_t
 - pxt, [21](#)
- MeshMetaTagsAdd
 - Afterwarp.h, [228](#)
- MeshMetaTagsClear
 - Afterwarp.h, [228](#)
- MeshMetaTagsCopy
 - Afterwarp.h, [229](#)
- MeshMetaTagsCount
 - Afterwarp.h, [229](#)
- MeshMetaTagsCreate
 - Afterwarp.h, [229](#)
- MeshMetaTagsDestroy
 - Afterwarp.h, [229](#)
- MeshMetaTagsErase
 - Afterwarp.h, [229](#)
- MeshMetaTagsGetByIndex
 - Afterwarp.h, [229](#)
- MeshMetaTagsGetByName
 - Afterwarp.h, [230](#)
- MeshMetaTagsTakeAway
 - Afterwarp.h, [230](#)
- MeshModel_t
 - pxt, [21](#)
- MeshModelCreate
 - Afterwarp.h, [230](#)
- MeshModelDestroy
 - Afterwarp.h, [230](#)
- MeshModelDismantle
 - Afterwarp.h, [230](#)
- MeshModelDraw
 - Afterwarp.h, [230](#)
- MeshModelDrawInstances
 - Afterwarp.h, [231](#)
- MeshModelGetChannel
 - Afterwarp.h, [231](#)
- MeshModelGetIndexBuffer

- Afterwarp.h, [231](#)
- MeshModelGetIndexCount
 - Afterwarp.h, [231](#)
- MeshModelGetVertexBuffer
 - Afterwarp.h, [231](#)
- MeshModelGetVertexCount
 - Afterwarp.h, [232](#)
- MeshModelRenderable
 - Afterwarp.h, [232](#)
- MeshModelTakeAway
 - Afterwarp.h, [232](#)
- MeshVoxel_t
 - pxt, [22](#)
- MeshVoxelComputeParameters
 - Afterwarp.h, [232](#)
- MeshVoxelCreate
 - Afterwarp.h, [232](#)
- MeshVoxelDestroy
 - Afterwarp.h, [232](#)
- MeshVoxelExtents
 - Afterwarp.h, [233](#)
- MeshVoxelIntersect
 - Afterwarp.h, [233](#)
- MeshVoxelLoadFromFile
 - Afterwarp.h, [233](#)
- MeshVoxelLoadFromFileInMemory
 - Afterwarp.h, [233](#)
- MeshVoxelSaveToFile
 - Afterwarp.h, [233](#)
- MeshVoxelTakeAway
 - Afterwarp.h, [234](#)
- MeshVoxelVisualize
 - Afterwarp.h, [234](#)
- MeshVoxelVisualizeFunc
 - pxt, [22](#)
- MIN
 - Afterwarp.Vectors.h, [468](#)
- minLOD
 - SamplerState, [90](#)
- mipLevel
 - ComputeBindTextureFormat, [46](#)
- Mirror
 - ImageRegion, [56](#)
- MODEL_TRANSFORM_GLOBAL
 - Afterwarp.Types.h, [427](#)
- MODEL_TRANSFORM_GLOBAL_MODEL
 - Afterwarp.Types.h, [427](#)
- MODEL_TRANSFORM_GLOBAL_VOLUME
 - Afterwarp.Types.h, [427](#)
- MODEL_TRANSFORM_LOCAL
 - Afterwarp.Types.h, [426](#)
- MODEL_TRANSFORM_LOCAL_MODEL
 - Afterwarp.Types.h, [426](#)
- MODEL_TRANSFORM_LOCAL_VOLUME
 - Afterwarp.Types.h, [426](#)
- ModelTransform
 - Afterwarp.Types.h, [399](#)
- MOUSE_BUTTON_LEFT
 - Afterwarp.Types.h, [430](#)
- MOUSE_BUTTON_MIDDLE
 - Afterwarp.Types.h, [430](#)
- MOUSE_BUTTON_NONE
 - Afterwarp.Types.h, [429](#)
- MOUSE_BUTTON_RIGHT
 - Afterwarp.Types.h, [430](#)
- MOUSE_EVENT_MOUSE_DOWN
 - Afterwarp.Types.h, [429](#)
- MOUSE_EVENT_MOUSE_ENTER
 - Afterwarp.Types.h, [429](#)
- MOUSE_EVENT_MOUSE_LEAVE
 - Afterwarp.Types.h, [429](#)
- MOUSE_EVENT_MOUSE_MOVE
 - Afterwarp.Types.h, [429](#)
- MOUSE_EVENT_MOUSE_UP
 - Afterwarp.Types.h, [429](#)
- MOUSE_EVENT_WHEEL_DOWN
 - Afterwarp.Types.h, [429](#)
- MOUSE_EVENT_WHEEL_UP
 - Afterwarp.Types.h, [429](#)
- MouseButton
 - Afterwarp.Types.h, [400](#)
- MouseEvent
 - Afterwarp.Types.h, [400](#)
- MouseFunc
 - pxt, [22](#)
- MultiplyColors
 - Afterwarp.h, [234](#)
- multisamples
 - TextureParameters, [107](#)
- name
 - ProgramElement, [77](#)
 - ProgramVariable, [79](#)
 - SceneMeshLatch, [94](#)
 - VertexElement, [114](#)
 - WidgetProperty, [115](#)
- normal
 - MeshBufferEntry, [62](#)
- nx
 - ApplicationConfiguration, [35](#)
- OBJECT_MODEL_ATTRIBUTE_DISABLE_REALIGN
 - Afterwarp.Types.h, [428](#)
- OBJECT_MODEL_ATTRIBUTE_HIERARCHYLESS
 - Afterwarp.Types.h, [428](#)
- OBJECT_MODEL_ATTRIBUTE_NON_VIEWABLE
 - Afterwarp.Types.h, [428](#)
- OBJECT_MODEL_ATTRIBUTE_SELECTABLE
 - Afterwarp.Types.h, [428](#)
- OBJECT_MODEL_ATTRIBUTE_TRANSPARENT
 - Afterwarp.Types.h, [428](#)
- OBJECT_MODEL_ATTRIBUTE_VISIBLE
 - Afterwarp.Types.h, [427](#)
- OBJECT_MODEL_COMPARE_DEPTH
 - Afterwarp.Types.h, [427](#)
- OBJECT_MODEL_COMPARE_DEPTH_REVERSE
 - Afterwarp.Types.h, [427](#)

- OBJECT_MODEL_COMPARE_MESHES
 - Afterwarp.Types.h, [427](#)
- OBJECT_MODEL_COMPARE_OBJECTS
 - Afterwarp.Types.h, [427](#)
- OBJECT_MODEL_COMPARE_ORDERED
 - Afterwarp.Types.h, [427](#)
- ObjectMaterial, [65](#)
 - albedoColor, [66](#)
 - ambientColor, [66](#)
 - bitmask, [66](#)
 - emissiveColor, [66](#)
 - frostedGlass, [67](#)
 - occlusion, [67](#)
 - payload, [67](#)
 - pxt, [22](#)
 - roughness, [67](#)
 - shadows, [67](#)
 - specularColor, [67](#)
 - specularExponent, [67](#)
 - technique, [67](#)
- ObjectMaterials_t
 - pxt, [22](#)
- ObjectMaterialsAdd
 - Afterwarp.h, [234](#)
- ObjectMaterialsClear
 - Afterwarp.h, [234](#)
- ObjectMaterialsCreate
 - Afterwarp.h, [234](#)
- ObjectMaterialsDestroy
 - Afterwarp.h, [235](#)
- ObjectMaterialsErase
 - Afterwarp.h, [235](#)
- ObjectMaterialsGetCount
 - Afterwarp.h, [235](#)
- ObjectMaterialsGetElement
 - Afterwarp.h, [235](#)
- ObjectModel_t
 - pxt, [22](#)
- ObjectModelCompareFunc
 - pxt, [22](#)
- ObjectModelConnectLatches
 - Afterwarp.h, [235](#)
- ObjectModelConnectLatchesByName
 - Afterwarp.h, [235](#)
- ObjectModelGetAABB
 - Afterwarp.h, [236](#)
- ObjectModelGetAlignments
 - Afterwarp.h, [236](#)
- ObjectModelGetAttributes
 - Afterwarp.h, [236](#)
- ObjectModelGetChild
 - Afterwarp.h, [236](#)
- ObjectModelGetChildCount
 - Afterwarp.h, [236](#)
- ObjectModelGetColor
 - Afterwarp.h, [236](#)
- ObjectModelGetConnectedLatches
 - Afterwarp.h, [237](#)
- ObjectModelGetDepthBias
 - Afterwarp.h, [237](#)
- ObjectModelGetDescription
 - Afterwarp.h, [237](#)
- ObjectModelGetHighlight
 - Afterwarp.h, [237](#)
- ObjectModelGetID
 - Afterwarp.h, [237](#)
- ObjectModelGetLatchTransform
 - Afterwarp.h, [237](#)
- ObjectModelGetLatchTransformByName
 - Afterwarp.h, [238](#)
- ObjectModelGetLatchWaypointCouple
 - Afterwarp.h, [238](#)
- ObjectModelGetLatchWaypointCoupleMatrix
 - Afterwarp.h, [238](#)
- ObjectModelGetLayers
 - Afterwarp.h, [238](#)
- ObjectModelGetMaterial
 - Afterwarp.h, [238](#)
- ObjectModelGetMesh
 - Afterwarp.h, [239](#)
- ObjectModelGetMeshName
 - Afterwarp.h, [239](#)
- ObjectModelGetName
 - Afterwarp.h, [239](#)
- ObjectModelGetNext
 - Afterwarp.h, [239](#)
- ObjectModelGetOrderIndex
 - Afterwarp.h, [239](#)
- ObjectModelGetOwner
 - Afterwarp.h, [239](#)
- ObjectModelGetParent
 - Afterwarp.h, [240](#)
- ObjectModelGetPayload
 - Afterwarp.h, [240](#)
- ObjectModelGetPosition
 - Afterwarp.h, [240](#)
- ObjectModelGetSize
 - Afterwarp.h, [240](#)
- ObjectModelGetTransform
 - Afterwarp.h, [240](#)
- ObjectModelGetVoxel
 - Afterwarp.h, [240](#)
- ObjectModelGetWaypointDistance
 - Afterwarp.h, [241](#)
- ObjectModelID
 - pxt, [23](#)
- ObjectModelInvalidate
 - Afterwarp.h, [241](#)
- ObjectModels_t
 - pxt, [23](#)
- ObjectModelsAdd
 - Afterwarp.h, [241](#)
- ObjectModelsAddWithID
 - Afterwarp.h, [241](#)
- ObjectModelsClear
 - Afterwarp.h, [241](#)

- ObjectModelsClearViews
 - Afterwarp.h, [241](#)
- ObjectModelsCreate
 - Afterwarp.h, [242](#)
- ObjectModelsCreateView
 - Afterwarp.h, [242](#)
- ObjectModelsDestroy
 - Afterwarp.h, [242](#)
- ObjectModelsErase
 - Afterwarp.h, [242](#)
- ObjectModelsEraseView
 - Afterwarp.h, [242](#)
- ObjectModelSetAlignments
 - Afterwarp.h, [242](#)
- ObjectModelSetAttributes
 - Afterwarp.h, [243](#)
- ObjectModelSetColor
 - Afterwarp.h, [243](#)
- ObjectModelSetDepthBias
 - Afterwarp.h, [243](#)
- ObjectModelSetDescription
 - Afterwarp.h, [243](#)
- ObjectModelSetHighlight
 - Afterwarp.h, [243](#)
- ObjectModelSetLayers
 - Afterwarp.h, [243](#)
- ObjectModelSetMaterial
 - Afterwarp.h, [244](#)
- ObjectModelSetMesh
 - Afterwarp.h, [244](#)
- ObjectModelSetMeshName
 - Afterwarp.h, [244](#)
- ObjectModelSetName
 - Afterwarp.h, [244](#)
- ObjectModelSetOrderIndex
 - Afterwarp.h, [244](#)
- ObjectModelSetParent
 - Afterwarp.h, [244](#)
- ObjectModelSetSize
 - Afterwarp.h, [245](#)
- ObjectModelSetTransform
 - Afterwarp.h, [245](#)
- ObjectModelSetVoxel
 - Afterwarp.h, [245](#)
- ObjectModelsGetFirst
 - Afterwarp.h, [245](#)
- ObjectModelsGetMeshes
 - Afterwarp.h, [245](#)
- ObjectModelsGetObjectByID
 - Afterwarp.h, [245](#)
- ObjectModelsGetObjectByName
 - Afterwarp.h, [246](#)
- ObjectModelsGetObjectCount
 - Afterwarp.h, [246](#)
- ObjectModelsIterator_t
 - pxt, [23](#)
- ObjectModelsPayload
 - Afterwarp.h, [246](#)
- ObjectModelView_t
 - pxt, [23](#)
- ObjectModelViewAutoDraw
 - Afterwarp.h, [246](#)
- ObjectModelViewCompare
 - Afterwarp.Types.h, [400](#)
- ObjectModelViewGetIntersectedObjects
 - Afterwarp.h, [246](#)
- ObjectModelViewGetIntersectedRays
 - Afterwarp.h, [246](#)
- ObjectModelViewGetLayers
 - Afterwarp.h, [247](#)
- ObjectModelViewGetObject
 - Afterwarp.h, [247](#)
- ObjectModelViewGetObjectCount
 - Afterwarp.h, [247](#)
- ObjectModelViewGetObjectsNotCulled
 - Afterwarp.h, [247](#)
- ObjectModelViewGetOwner
 - Afterwarp.h, [247](#)
- ObjectModelViewGetProjection
 - Afterwarp.h, [247](#)
- ObjectModelViewGetView
 - Afterwarp.h, [248](#)
- ObjectModelViewGetViewProjection
 - Afterwarp.h, [248](#)
- ObjectModelViewInvalidate
 - Afterwarp.h, [248](#)
- ObjectModelViewSelect
 - Afterwarp.h, [248](#)
- ObjectModelViewSelectAny
 - Afterwarp.h, [248](#)
- ObjectModelViewSetLayers
 - Afterwarp.h, [248](#)
- ObjectModelViewSetProjection
 - Afterwarp.h, [249](#)
- ObjectModelViewSetView
 - Afterwarp.h, [249](#)
- ObjectModelViewSort
 - Afterwarp.h, [249](#)
- ObjectModelViewSortWith
 - Afterwarp.h, [249](#)
- ObjectModelViewUpdate
 - Afterwarp.h, [249](#)
- ObjectModelViewUpdateNeeded
 - Afterwarp.h, [249](#)
- ObjectPayload
 - pxt, [23](#)
- occlusion
 - ObjectMaterial, [67](#)
 - ParallaxMappingParameters, [72](#)
- OceanMaterial, [68](#)
 - albedoColor, [69](#)
 - ambientColor, [69](#)
 - bitmask, [69](#)
 - emissiveColor, [69](#)
 - extinction, [69](#)
 - fresnel, [69](#)

- pxt, [23](#)
 - shadows, [69](#)
 - specularColor, [70](#)
 - specularExponent, [70](#)
- OceanSimulation_t
 - pxt, [23](#)
- OceanSimulationCreate
 - Afterwarp.h, [250](#)
- OceanSimulationDestroy
 - Afterwarp.h, [250](#)
- OceanSimulationGetAttributes
 - Afterwarp.h, [250](#)
- OceanSimulationGetDevice
 - Afterwarp.h, [250](#)
- OceanSimulationGetMaterial
 - Afterwarp.h, [250](#)
- OceanSimulationGetPlane
 - Afterwarp.h, [250](#)
- OceanSimulationGetProjection
 - Afterwarp.h, [250](#)
- OceanSimulationGetSamplerShadow
 - Afterwarp.h, [251](#)
- OceanSimulationGetScale
 - Afterwarp.h, [251](#)
- OceanSimulationGetSections
 - Afterwarp.h, [251](#)
- OceanSimulationGetView
 - Afterwarp.h, [251](#)
- OceanSimulationGetViewDistance
 - Afterwarp.h, [251](#)
- OceanSimulationGetWavesParameters
 - Afterwarp.h, [251](#)
- OceanSimulationGetWavesTexture
 - Afterwarp.h, [252](#)
- OceanSimulationRender
 - Afterwarp.h, [252](#)
- OceanSimulationSetAttributes
 - Afterwarp.h, [252](#)
- OceanSimulationSetMaterial
 - Afterwarp.h, [252](#)
- OceanSimulationSetPlane
 - Afterwarp.h, [252](#)
- OceanSimulationSetProjection
 - Afterwarp.h, [252](#)
- OceanSimulationSetScale
 - Afterwarp.h, [253](#)
- OceanSimulationSetSections
 - Afterwarp.h, [253](#)
- OceanSimulationSetView
 - Afterwarp.h, [253](#)
- OceanSimulationSetViewDistance
 - Afterwarp.h, [253](#)
- OceanSimulationSetWavesParameters
 - Afterwarp.h, [253](#)
- OceanSimulationUpdate
 - Afterwarp.h, [253](#)
- OceanWavesParameters, [70](#)
 - choppiness, [71](#)
 - pxt, [24](#)
 - resolution, [71](#)
 - waveSize, [71](#)
 - wind, [71](#)
- offset
 - ProgramVariable, [79](#)
 - VertexElement, [114](#)
- op
 - Blend, [39](#)
- opacity
 - FogParameters, [50](#)
- orientation
 - SceneMeshLatch, [94](#)
- outlineColor
 - SelectionHighlightParameters, [99](#)
- outlineDistanceMaxSDF
 - SignedDistanceField, [102](#)
- outlineDistanceMinSDF
 - SignedDistanceField, [102](#)
- outlineOffsetSDF
 - SignedDistanceField, [102](#)
- outlineStart
 - SelectionHighlightParameters, [99](#)
- ParallaxMappingParameters, [72](#)
 - occlusion, [72](#)
 - pxt, [24](#)
 - samplesMax, [72](#)
 - samplesMin, [72](#)
 - scale, [72](#)
- parameterCount
 - ApplicationConfiguration, [35](#)
- parameterStrings
 - ApplicationConfiguration, [35](#)
- PATH_JOINT_BEVEL
 - Afterwarp.Types.h, [417](#)
- PATH_JOINT_MITER
 - Afterwarp.Types.h, [417](#)
- PATH_JOINT_MITER_BEVEL
 - Afterwarp.Types.h, [417](#)
- PATH_JOINT_NONE
 - Afterwarp.Types.h, [417](#)
- PATH_JOINT_ROUND
 - Afterwarp.Types.h, [417](#)
- PATH_JOINT_SIMPLE
 - Afterwarp.Types.h, [417](#)
- PathBrokerCreate
 - Afterwarp.h, [254](#)
- PathBrokerDestroy
 - Afterwarp.h, [254](#)
- PathBrokerFill
 - Afterwarp.h, [254](#)
- PathBrokerReset
 - Afterwarp.h, [254](#)
- PathBrokerStroke
 - Afterwarp.h, [254](#)
- PathElement, [73](#)
 - command, [74](#)
 - length, [74](#)

- PathElement, [74](#)
 - reserved, [74](#)
 - tag, [74](#)
 - value, [75](#)
- PathJoint
 - Afterwarp.Types.h, [400](#)
- payload
 - ObjectMaterial, [67](#)
 - SceneLight, [92](#)
- PIXEL_FORMAT_A8
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_B5G6R5
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_BGR10A2
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_BGR10X2
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_BGR5A1
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_BGR5X1
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_BGR8
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_BGRA4
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_BGRA8
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_BGRA8_SRGB
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_BGRX4
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_BGRX8
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_D16
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_D24S8
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_D32F
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_D32S8F
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_I8
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_L16
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_L8
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_LA4
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_LA8
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_R16
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_R16F
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_R16I
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_R16S
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_R16U
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_R32F
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_R32I
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_R32U
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_R8
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_R8I
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_R8S
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_R8U
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_R_BC
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_RG11B10F
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RG16
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_RG16F
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RG16I
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RG16S
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_RG16U
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RG32F
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RG32I
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RG32U
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RG8
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_RG8I
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RG8S
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_RG8U
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RG_BC
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_RGB10A2
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGB10A2U
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGB8
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_RGB9E5F
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGB_BC
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_RGB_BC_SRGB
 - Afterwarp.Types.h, [433](#)

- Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_RGBA16
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_RGBA16F
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGBA16I
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGBA16S
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_RGBA16U
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGBA32F
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGBA32I
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGBA32U
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGBA8
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_RGBA8_SRGB
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGBA8I
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGBA8S
 - Afterwarp.Types.h, [433](#)
- PIXEL_FORMAT_RGBA8U
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_RGBA_BC
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_RGBA_BC_SRGB
 - Afterwarp.Types.h, [435](#)
- PIXEL_FORMAT_RGBX8
 - Afterwarp.Types.h, [434](#)
- PIXEL_FORMAT_UNKNOWN
 - Afterwarp.Types.h, [433](#)
- PixelFormat
 - Afterwarp.Types.h, [400](#)
- PixelFormatBits
 - Afterwarp.h, [254](#)
- PixelFormatConvert
 - Afterwarp.h, [255](#)
- PixelFormatConvertArray
 - Afterwarp.h, [255](#)
- PixelFormatValid
 - Afterwarp.h, [255](#)
- Point, [75](#)
 - Afterwarp.Types.h, [400](#)
 - x, [75](#)
 - y, [75](#)
- POINT_SHAPE_CROSS
 - Afterwarp.Types.h, [418](#)
- POINT_SHAPE_ROUND
 - Afterwarp.Types.h, [418](#)
- POINT_SHAPE_SQUARE
 - Afterwarp.Types.h, [418](#)
- POINT_SHAPE_STAR
 - Afterwarp.Types.h, [418](#)
- POINT_SHAPE_TRIANGLE
 - Afterwarp.Types.h, [418](#)
- pointAdd
 - Afterwarp.Vectors.h, [484](#)
- pointAddF
 - Afterwarp.Vectors.h, [484](#)
- pointAngle
 - Afterwarp.Vectors.h, [484](#)
- pointAngleF
 - Afterwarp.Vectors.h, [484](#)
- pointAxisX
 - Afterwarp.Vectors.h, [509](#)
- pointAxisXF
 - Afterwarp.Vectors.h, [510](#)
- pointAxisY
 - Afterwarp.Vectors.h, [510](#)
- pointAxisYF
 - Afterwarp.Vectors.h, [510](#)
- pointCross
 - Afterwarp.Vectors.h, [485](#)
- pointCrossF
 - Afterwarp.Vectors.h, [485](#)
- pointDistance
 - Afterwarp.Vectors.h, [485](#)
- pointDistanceF
 - Afterwarp.Vectors.h, [485](#)
- pointDivide
 - Afterwarp.Vectors.h, [485](#)
- pointDivideF
 - Afterwarp.Vectors.h, [485](#)
- pointDot
 - Afterwarp.Vectors.h, [486](#)
- pointDotF
 - Afterwarp.Vectors.h, [486](#)
- pointEmpty
 - Afterwarp.Vectors.h, [486](#)
- pointEmptyF
 - Afterwarp.Vectors.h, [486](#)
- pointEquals
 - Afterwarp.Vectors.h, [486](#)
- pointEqualsF
 - Afterwarp.Vectors.h, [486](#)
- PointF, [76](#)
 - Afterwarp.Types.h, [400](#)
 - x, [76](#)
 - y, [76](#)
- pointF
 - Afterwarp.Vectors.h, [487](#)
- pointInRect
 - Afterwarp.Vectors.h, [487](#)
- pointInRectF
 - Afterwarp.Vectors.h, [487](#)
- pointLength
 - Afterwarp.Vectors.h, [487](#)
- pointLengthF
 - Afterwarp.Vectors.h, [487](#)
- pointLerp
 - Afterwarp.Vectors.h, [487](#)
- pointLerpF
 - Afterwarp.Vectors.h, [487](#)

- Afterwarp.Vectors.h, [488](#)
- pointMultiply
 - Afterwarp.Vectors.h, [488](#)
- pointMultiplyF
 - Afterwarp.Vectors.h, [488](#)
- pointNegate
 - Afterwarp.Vectors.h, [488](#)
- pointNegateF
 - Afterwarp.Vectors.h, [488](#)
- pointNormalizeF
 - Afterwarp.Vectors.h, [488](#)
- pointRescale
 - Afterwarp.Vectors.h, [489](#)
- pointRescaleF
 - Afterwarp.Vectors.h, [489](#)
- PointShape
 - Afterwarp.Types.h, [400](#)
- pointSubtract
 - Afterwarp.Vectors.h, [489](#)
- pointSubtractF
 - Afterwarp.Vectors.h, [489](#)
- pointTransformF
 - Afterwarp.Vectors.h, [489](#)
- pointUnity
 - Afterwarp.Vectors.h, [510](#)
- pointUnityF
 - Afterwarp.Vectors.h, [510](#)
- pointValue
 - Afterwarp.Vectors.h, [489](#)
- pointValueF
 - Afterwarp.Vectors.h, [490](#)
- pointZero
 - Afterwarp.Vectors.h, [510](#)
- pointZeroF
 - Afterwarp.Vectors.h, [510](#)
- position
 - ApplicationConfiguration, [35](#)
 - MeshBufferEntry, [63](#)
 - SceneLight, [92](#)
 - SceneMeshLatch, [94](#)
 - TextEntryRect, [105](#)
- positionMax
 - CameraConstraints, [40](#)
- positionMin
 - CameraConstraints, [40](#)
- PremultiplyAlpha
 - Afterwarp.h, [255](#)
- PRIMITIVE_TOPOLOGY_LINE_STRIP
 - Afterwarp.Types.h, [411](#)
- PRIMITIVE_TOPOLOGY_LINE_STRIP_ADJACENCY
 - Afterwarp.Types.h, [412](#)
- PRIMITIVE_TOPOLOGY_LINES
 - Afterwarp.Types.h, [411](#)
- PRIMITIVE_TOPOLOGY_LINES_ADJACENCY
 - Afterwarp.Types.h, [412](#)
- PRIMITIVE_TOPOLOGY_POINTS
 - Afterwarp.Types.h, [411](#)
- PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP
 - Afterwarp.Types.h, [412](#)
- PRIMITIVE_TOPOLOGY_TRIANGLE_STRIP_ADJACENCY
 - Afterwarp.Types.h, [412](#)
- PRIMITIVE_TOPOLOGY_TRIANGLES
 - Afterwarp.Types.h, [412](#)
- PRIMITIVE_TOPOLOGY_TRIANGLES_ADJACENCY
 - Afterwarp.Types.h, [412](#)
- PRIMITIVE_TOPOLOGY_UNKNOWN
 - Afterwarp.Types.h, [411](#)
- PrimitiveTopology
 - Afterwarp.Types.h, [401](#)
- Program_t
 - pxt, [24](#)
- ProgramBegin
 - Afterwarp.h, [255](#)
- ProgramBind
 - Afterwarp.h, [255](#)
- ProgramCommit
 - Afterwarp.h, [256](#)
- ProgramCreate
 - Afterwarp.h, [256](#)
- ProgramDestroy
 - Afterwarp.h, [256](#)
- ProgramDraw
 - Afterwarp.h, [256](#)
- ProgramDrawIndexed
 - Afterwarp.h, [257](#)
- ProgramDrawInstances
 - Afterwarp.h, [257](#)
- ProgramDrawInstancesIndexed
 - Afterwarp.h, [257](#)
- ProgramElement, [77](#)
 - channel, [77](#)
 - element, [77](#)
 - index, [77](#)
 - name, [77](#)
 - pxt, [24](#)
 - shader, [78](#)
 - size, [78](#)
- ProgramEnd
 - Afterwarp.h, [257](#)
- ProgramGetDevice
 - Afterwarp.h, [257](#)
- ProgramResetBindings
 - Afterwarp.h, [258](#)
- ProgramResetCache
 - Afterwarp.h, [258](#)
- ProgramSetPatchVertices
 - Afterwarp.h, [258](#)
- ProgramUnbind
 - Afterwarp.h, [258](#)
- ProgramUpdateByIndex
 - Afterwarp.h, [258](#)
- ProgramUpdateByName
 - Afterwarp.h, [258](#)
- ProgramVariable, [78](#)
 - count, [79](#)
 - format, [79](#)

- index, 79
- name, 79
- offset, 79
- pxt, 24
- shader, 79
- size, 79
- pxt, 11
 - ActorCamera_t, 17
 - AmbientOcclusionParameters, 17
 - Application_t, 17
 - ApplicationConfiguration, 17
 - ApplicationEvents, 17
 - BoolFunc, 17
 - Buffer_t, 17
 - CameraConstraints, 17
 - Canvas_t, 18
 - CanvasAttributes, 18
 - CanvasSamplerState, 18
 - ColorDithering_t, 18
 - ComputeBindTextureFormat, 18
 - ComputeProgram_t, 18
 - deviceAttributeExtensions, 30
 - DeviceAttributes, 18
 - DeviceBehavior, 19
 - DeviceLegacyBits, 19
 - EventFunc, 19
 - EventHookFunc, 19
 - FogParameters, 19
 - FontEffect, 19
 - FontParameters, 19
 - GaussianBlur_t, 19
 - Grapher_t, 20
 - ImageAtlas_t, 20
 - ImageRegion, 20
 - KawaseBlur_t, 20
 - KeyboardFunc, 20
 - LibraryBool, 20
 - MeshAligns, 20
 - MeshBuffer_t, 21
 - MeshBufferEntry, 21
 - MeshLoadSaveFeedback, 21
 - MeshMetaTag_t, 21
 - MeshMetaTagPortion, 21
 - MeshMetaTags_t, 21
 - MeshModel_t, 21
 - MeshVoxel_t, 22
 - MeshVoxelVisualizeFunc, 22
 - MouseFunc, 22
 - ObjectMaterial, 22
 - ObjectMaterials_t, 22
 - ObjectModel_t, 22
 - ObjectModelCompareFunc, 22
 - ObjectModelIID, 23
 - ObjectModels_t, 23
 - ObjectModelsIterator_t, 23
 - ObjectModelView_t, 23
 - ObjectPayload, 23
 - OceanMaterial, 23
 - OceanSimulation_t, 23
 - OceanWavesParameters, 24
 - ParallaxMappingParameters, 24
 - Program_t, 24
 - ProgramElement, 24
 - ProgramVariable, 24
 - RenderingState, 24
 - Sampler_t, 24
 - SamplerState, 24
 - Scene_t, 25
 - SceneAttributes, 25
 - SceneLight, 25
 - SceneLights_t, 25
 - SceneMesh_t, 25
 - SceneMeshes_t, 25
 - SceneMeshLatch, 25
 - SceneMeshLatches_t, 25
 - SceneMeshMaterial_t, 26
 - SceneMeshMaterialRange, 26
 - SceneMeshMaterials_t, 26
 - SceneMeshMaterialShading, 26
 - SelectionHighlight_t, 26
 - SelectionHighlightParameters, 26
 - ShadowCaster_t, 26
 - ShadowCastingAtlas_t, 26
 - ShadowParameters, 27
 - SignedDistanceField, 27
 - SpatialFog_t, 27
 - StatusFunc, 27
 - Surface_t, 27
 - TextEntryRect, 27
 - TextModeller_t, 27
 - TextRenderer_t, 27
 - TextRenderModifiers, 28
 - Texture_t, 28
 - TextureAttributes, 28
 - TextureCabinet_t, 28
 - TextureCabinetAttributes, 28
 - TextureParameters, 28
 - Timer_t, 28
 - ToneMappingBloom, 28
 - UnicodeChar, 29
 - UntypedHandle, 29
 - VertexElement, 29
 - Widget_t, 29
 - WidgetExternalEvent, 29
 - WidgetProperty, 29
 - WidgetPropertyData, 29
- Quad, 80
 - Afterwarp.Types.h, 401
 - bottomLeft, 80
 - bottomRight, 80
 - topLeft, 81
 - topRight, 81
- quadFlip
 - Afterwarp.Vectors.h, 490
- quadFromRect
 - Afterwarp.Vectors.h, 490

- quadFromRectF
 - Afterwarp.Vectors.h, [490](#)
- quadMirror
 - Afterwarp.Vectors.h, [490](#)
- quadOffset
 - Afterwarp.Vectors.h, [490](#)
- quadScale
 - Afterwarp.Vectors.h, [491](#)
- quadTransform
 - Afterwarp.Vectors.h, [491](#)
- quadUnity
 - Afterwarp.Vectors.h, [510](#)
- quadZero
 - Afterwarp.Vectors.h, [511](#)
- Quaternion, [81](#)
 - Afterwarp.Types.h, [401](#)
 - w, [82](#)
 - x, [82](#)
 - y, [82](#)
 - z, [82](#)
- quaternionAngle
 - Afterwarp.Vectors.h, [491](#)
- quaternionAxis
 - Afterwarp.Vectors.h, [491](#)
- quaternionConjugate
 - Afterwarp.Vectors.h, [491](#)
- quaternionDot
 - Afterwarp.Vectors.h, [491](#)
- quaternionExponentiate
 - Afterwarp.Vectors.h, [492](#)
- quaternionIdentity
 - Afterwarp.Vectors.h, [511](#)
- quaternionLength
 - Afterwarp.Vectors.h, [492](#)
- quaternionMatrix
 - Afterwarp.Vectors.h, [492](#)
- quaternionMultiply
 - Afterwarp.Vectors.h, [492](#)
- quaternionNormalize
 - Afterwarp.Vectors.h, [492](#)
- quaternionRotate
 - Afterwarp.Vectors.h, [492](#)
- quaternionRotateInertialToObject
 - Afterwarp.Vectors.h, [493](#)
- quaternionRotateObjectToInertial
 - Afterwarp.Vectors.h, [493](#)
- quaternionRotateX
 - Afterwarp.Vectors.h, [493](#)
- quaternionRotateY
 - Afterwarp.Vectors.h, [493](#)
- quaternionRotateZ
 - Afterwarp.Vectors.h, [493](#)
- quaternionSlerp
 - Afterwarp.Vectors.h, [493](#)
- radius
 - AmbientOcclusionParameters, [32](#)
- RandomSequenceGenerate
 - Afterwarp.h, [259](#)
- RandomSequenceGenerate64
 - Afterwarp.h, [259](#)
- RandomSequenceGenerateDouble
 - Afterwarp.h, [259](#)
- RandomSequenceGenerateFloat
 - Afterwarp.h, [259](#)
- RandomSequenceGenerateGaussian
 - Afterwarp.h, [259](#)
- RandomSequenceGenerateRanged
 - Afterwarp.h, [259](#)
- RandomSequenceInit
 - Afterwarp.h, [260](#)
- RandomSequenceInitBySeed
 - Afterwarp.h, [260](#)
- RayCreate
 - Afterwarp.h, [260](#)
- RayIntersectCubeVolume
 - Afterwarp.h, [260](#)
- RayIntersectPlane
 - Afterwarp.h, [260](#)
- RayIntersectTriangle
 - Afterwarp.h, [261](#)
- Rect, [82](#)
 - Afterwarp.Types.h, [401](#)
 - height, [83](#)
 - left, [83](#)
 - top, [83](#)
 - width, [83](#)
- rectangle
 - TextEntryRect, [105](#)
- rectEmpty
 - Afterwarp.Vectors.h, [494](#)
- rectEmptyF
 - Afterwarp.Vectors.h, [494](#)
- rectEquals
 - Afterwarp.Vectors.h, [494](#)
- rectEqualsF
 - Afterwarp.Vectors.h, [494](#)
- RectF, [83](#)
 - Afterwarp.Types.h, [401](#)
 - height, [84](#)
 - left, [84](#)
 - top, [84](#)
 - width, [84](#)
- rectF
 - Afterwarp.Vectors.h, [494](#)
- rectGetBottom
 - Afterwarp.Vectors.h, [494](#)
- rectGetBottomF
 - Afterwarp.Vectors.h, [495](#)
- rectGetBottomLeft
 - Afterwarp.Vectors.h, [495](#)
- rectGetBottomLeftF
 - Afterwarp.Vectors.h, [495](#)
- rectGetBottomRight
 - Afterwarp.Vectors.h, [495](#)
- rectGetBottomRightF
 - Afterwarp.Vectors.h, [495](#)

- rectGetRight
 - Afterwarp.Vectors.h, [495](#)
- rectGetRightF
 - Afterwarp.Vectors.h, [495](#)
- rectGetSize
 - Afterwarp.Vectors.h, [496](#)
- rectGetSizeF
 - Afterwarp.Vectors.h, [496](#)
- rectGetTopLeft
 - Afterwarp.Vectors.h, [496](#)
- rectGetTopLeftF
 - Afterwarp.Vectors.h, [496](#)
- rectGetTopRight
 - Afterwarp.Vectors.h, [496](#)
- rectGetTopRightF
 - Afterwarp.Vectors.h, [496](#)
- rectInflate
 - Afterwarp.Vectors.h, [496](#)
- rectInflateF
 - Afterwarp.Vectors.h, [497](#)
- rectInRect
 - Afterwarp.Vectors.h, [497](#)
- rectInRectF
 - Afterwarp.Vectors.h, [497](#)
- rectIntersect
 - Afterwarp.Vectors.h, [497](#)
- rectIntersectF
 - Afterwarp.Vectors.h, [497](#)
- rectJoin
 - Afterwarp.Vectors.h, [497](#)
- rectJoinF
 - Afterwarp.Vectors.h, [498](#)
- rectOffset
 - Afterwarp.Vectors.h, [498](#)
- rectOffsetF
 - Afterwarp.Vectors.h, [498](#)
- rectOverlap
 - Afterwarp.Vectors.h, [498](#)
- rectOverlapF
 - Afterwarp.Vectors.h, [498](#)
- rectSetBottom
 - Afterwarp.Vectors.h, [498](#)
- rectSetBottomF
 - Afterwarp.Vectors.h, [499](#)
- rectSetBottomRight
 - Afterwarp.Vectors.h, [499](#)
- rectSetBottomRightF
 - Afterwarp.Vectors.h, [499](#)
- rectSetRight
 - Afterwarp.Vectors.h, [499](#)
- rectSetRightF
 - Afterwarp.Vectors.h, [499](#)
- rectSetSize
 - Afterwarp.Vectors.h, [499](#)
- rectSetSizeF
 - Afterwarp.Vectors.h, [500](#)
- rectSetTopLeft
 - Afterwarp.Vectors.h, [500](#)
- rectSetTopLeftF
 - Afterwarp.Vectors.h, [500](#)
- rectToIntRectF
 - Afterwarp.Vectors.h, [500](#)
- rectZero
 - Afterwarp.Vectors.h, [511](#)
- rectZeroF
 - Afterwarp.Vectors.h, [511](#)
- red
 - FloatColor, [47](#)
 - FloatColorRGB, [48](#)
- RenderingState, [85](#)
 - blendAlpha, [86](#)
 - blendColor, [86](#)
 - blendConstant, [86](#)
 - clampDepthBias, [86](#)
 - cullFace, [86](#)
 - depthBias, [86](#)
 - depthFunc, [86](#)
 - pxt, [24](#)
 - slopeDepthBias, [87](#)
 - states, [87](#)
 - stencilBack, [87](#)
 - stencilFront, [87](#)
 - stencilRefMask, [87](#)
 - stencilRefValue, [87](#)
 - stencilWriteMask, [87](#)
- renderingType
 - ApplicationConfiguration, [35](#)
- reserved
 - PathElement, [74](#)
 - SceneLight, [93](#)
 - WidgetProperty, [116](#)
- resolution
 - OceanWavesParameters, [71](#)
- right
 - Margins, [58](#)
- Rotate
 - ImageRegion, [56](#)
- rotationMax
 - CameraConstraints, [41](#)
- rotationMin
 - CameraConstraints, [41](#)
- roughness
 - ObjectMaterial, [67](#)
 - SceneMeshMaterialShading, [97](#)
- Sampler_t
 - pxt, [24](#)
- SamplerBind
 - Afterwarp.h, [261](#)
- SamplerCreate
 - Afterwarp.h, [261](#)
- SamplerDestroy
 - Afterwarp.h, [261](#)
- SamplerGetDevice
 - Afterwarp.h, [261](#)
- SamplerGetState
 - Afterwarp.h, [262](#)

- SamplerSetState
 - Afterwarp.h, [262](#)
- SamplerState, [88](#)
 - address, [89](#)
 - anisotropy, [89](#)
 - biasLOD, [89](#)
 - borderColor, [89](#)
 - compareFunc, [89](#)
 - compareToRef, [89](#)
 - filterMag, [89](#)
 - filterMin, [90](#)
 - filterMip, [90](#)
 - maxLOD, [90](#)
 - minLOD, [90](#)
 - pxt, [24](#)
- SamplerUnbind
 - Afterwarp.h, [262](#)
- samples
 - AmbientOcclusionParameters, [32](#)
- samplesMax
 - ParallaxMappingParameters, [72](#)
- samplesMin
 - ParallaxMappingParameters, [72](#)
- scale
 - ParallaxMappingParameters, [72](#)
 - TextRenderModifiers, [106](#)
- scattering
 - FogParameters, [50](#)
- SCENE_ATTRIBUTE_COLORING
 - Afterwarp.Types.h, [424](#)
- SCENE_ATTRIBUTE_DEPTH_PREPASS
 - Afterwarp.Types.h, [424](#)
- SCENE_ATTRIBUTE_GLASSY
 - Afterwarp.Types.h, [424](#)
- SCENE_ATTRIBUTE_INSTANCING
 - Afterwarp.Types.h, [424](#)
- SCENE_ATTRIBUTE_LINEAR_DEPTHS
 - Afterwarp.Types.h, [424](#)
- SCENE_ATTRIBUTE_MODELING
 - Afterwarp.Types.h, [424](#)
- SCENE_ATTRIBUTE_NORMALS
 - Afterwarp.Types.h, [424](#)
- SCENE_ATTRIBUTE_SHADOWS_CUBIC
 - Afterwarp.Types.h, [424](#)
- SCENE_ATTRIBUTE_SINGLE_PASS
 - Afterwarp.Types.h, [424](#)
- SCENE_ATTRIBUTE_TEXTUREING_CUBIC
 - Afterwarp.Types.h, [424](#)
- SCENE_MESH_LATCH_TYPE_JOINT
 - Afterwarp.Types.h, [425](#)
- SCENE_MESH_LATCH_TYPE_WAYPOINT
 - Afterwarp.Types.h, [425](#)
- SCENE_MESH_TEXTURE_DIFFUSE
 - Afterwarp.Types.h, [425](#)
- SCENE_MESH_TEXTURE_NORMALS
 - Afterwarp.Types.h, [425](#)
- SCENE_MESH_TEXTURE_PARALLAX
 - Afterwarp.Types.h, [425](#)
- SCENE_MESH_VERTEX_ELEMENTS_INDEX_UNDEFINED
 - Afterwarp.Types.h, [425](#)
- SCENE_SAMPLER_TYPE_ALBEDO
 - Afterwarp.Types.h, [424](#)
- SCENE_SAMPLER_TYPE_NORMAL_MAP
 - Afterwarp.Types.h, [424](#)
- SCENE_SAMPLER_TYPE_PARALLAX_MAP
 - Afterwarp.Types.h, [424](#)
- SCENE_SAMPLER_TYPE_SHADOW_MAP
 - Afterwarp.Types.h, [424](#)
- Scene_t
 - pxt, [25](#)
- SCENE_TEXTURE_TYPE_ALBEDO
 - Afterwarp.Types.h, [424](#)
- SCENE_TEXTURE_TYPE_NORMAL_MAP
 - Afterwarp.Types.h, [424](#)
- SCENE_TEXTURE_TYPE_PARALLAX_MAP
 - Afterwarp.Types.h, [424](#)
- SceneAttributes
 - pxt, [25](#)
- SceneBegin
 - Afterwarp.h, [262](#)
- SceneCreateDepthsNormals
 - Afterwarp.h, [262](#)
- SceneCreateModeling
 - Afterwarp.h, [262](#)
- SceneDestroy
 - Afterwarp.h, [263](#)
- SceneEnd
 - Afterwarp.h, [263](#)
- SceneGetActiveVertexElements
 - Afterwarp.h, [263](#)
- SceneGetAttributes
 - Afterwarp.h, [263](#)
- SceneGetDevice
 - Afterwarp.h, [263](#)
- SceneGetInstancesCount
 - Afterwarp.h, [263](#)
- SceneGetLights
 - Afterwarp.h, [263](#)
- SceneGetMaterial
 - Afterwarp.h, [264](#)
- SceneGetParallaxMappingParameters
 - Afterwarp.h, [264](#)
- SceneGetProgram
 - Afterwarp.h, [264](#)
- SceneGetProjection
 - Afterwarp.h, [264](#)
- SceneGetSampler
 - Afterwarp.h, [264](#)
- SceneGetShadowCastingAtlas
 - Afterwarp.h, [264](#)
- SceneGetTexture
 - Afterwarp.h, [265](#)
- SceneGetTextureCabinet
 - Afterwarp.h, [265](#)
- SceneGetToneMappingCoefficients
 - Afterwarp.h, [265](#)

- SceneGetVertexElementCount
 - Afterwarp.h, [265](#)
- SceneGetVertexElements
 - Afterwarp.h, [265](#)
- SceneGetView
 - Afterwarp.h, [265](#)
- SceneGetWorld
 - Afterwarp.h, [266](#)
- SceneInstances
 - Afterwarp.h, [266](#)
- SceneLight, [90](#)
 - albedoColor, [91](#)
 - ambientColor, [91](#)
 - angle, [91](#)
 - angleCutoff, [92](#)
 - attenuationEnd, [92](#)
 - attenuationStart, [92](#)
 - bitmask, [92](#)
 - direction, [92](#)
 - intensity, [92](#)
 - payload, [92](#)
 - position, [92](#)
 - pxt, [25](#)
 - reserved, [93](#)
 - shadowCaster, [93](#)
 - specularColor, [93](#)
 - specularExponent, [93](#)
- SceneLights_t
 - pxt, [25](#)
- SceneLightsAdd
 - Afterwarp.h, [266](#)
- SceneLightsClear
 - Afterwarp.h, [266](#)
- SceneLightsCreate
 - Afterwarp.h, [266](#)
- SceneLightsDestroy
 - Afterwarp.h, [266](#)
- SceneLightsErase
 - Afterwarp.h, [267](#)
- SceneLightsExecute
 - Afterwarp.h, [267](#)
- SceneLightsGetClusters
 - Afterwarp.h, [267](#)
- SceneLightsGetClusterSize
 - Afterwarp.h, [267](#)
- SceneLightsGetCount
 - Afterwarp.h, [267](#)
- SceneLightsGetCullingMode
 - Afterwarp.h, [267](#)
- SceneLightsGetDepthSlices
 - Afterwarp.h, [268](#)
- SceneLightsGetDevice
 - Afterwarp.h, [268](#)
- SceneLightsGetElement
 - Afterwarp.h, [268](#)
- SceneLightsGetIndices
 - Afterwarp.h, [268](#)
- SceneLightsGetViewSize
 - Afterwarp.h, [268](#)
- SceneLightsRenderDebug
 - Afterwarp.h, [268](#)
- SceneLightsSetClusterSize
 - Afterwarp.h, [269](#)
- SceneLightsSetCullingMode
 - Afterwarp.h, [269](#)
- SceneLightsSetDepthSlices
 - Afterwarp.h, [269](#)
- SceneLightsSetViewSize
 - Afterwarp.h, [269](#)
- SceneMesh_t
 - pxt, [25](#)
- SceneMeshAutoDraw
 - Afterwarp.h, [269](#)
- SceneMeshAutoDrawSliced
 - Afterwarp.h, [270](#)
- SceneMeshes_t
 - pxt, [25](#)
- SceneMeshesAdd
 - Afterwarp.h, [270](#)
- SceneMeshesAddFromBuffer
 - Afterwarp.h, [270](#)
- SceneMeshesAddFromFile
 - Afterwarp.h, [271](#)
- SceneMeshesClear
 - Afterwarp.h, [271](#)
- SceneMeshesCreate
 - Afterwarp.h, [271](#)
- SceneMeshesDestroy
 - Afterwarp.h, [271](#)
- SceneMeshesErase
 - Afterwarp.h, [272](#)
- SceneMeshesGetCount
 - Afterwarp.h, [272](#)
- SceneMeshesGetDevice
 - Afterwarp.h, [272](#)
- SceneMeshesGetMeshByIndex
 - Afterwarp.h, [272](#)
- SceneMeshesGetMeshByName
 - Afterwarp.h, [272](#)
- SceneMeshesPayload
 - Afterwarp.h, [272](#)
- SceneMeshesSlice
 - Afterwarp.h, [273](#)
- SceneMeshGetBounds
 - Afterwarp.h, [273](#)
- SceneMeshGetLatches
 - Afterwarp.h, [273](#)
- SceneMeshGetMaterials
 - Afterwarp.h, [273](#)
- SceneMeshGetModel
 - Afterwarp.h, [273](#)
- SceneMeshGetName
 - Afterwarp.h, [273](#)
- SceneMeshGetPayload
 - Afterwarp.h, [274](#)
- SceneMeshGetScale
 - Afterwarp.h, [274](#)

- Afterwarp.h, [274](#)
- SceneMeshGetSize
 - Afterwarp.h, [274](#)
- SceneMeshGetSlice
 - Afterwarp.h, [274](#)
- SceneMeshGetTags
 - Afterwarp.h, [274](#)
- SceneMeshGetVertexElementsIndex
 - Afterwarp.h, [274](#)
- SceneMeshGetVoxel
 - Afterwarp.h, [275](#)
- SceneMeshLatch, [93](#)
 - group, [94](#)
 - name, [94](#)
 - orientation, [94](#)
 - position, [94](#)
 - pxt, [25](#)
 - type, [94](#)
- SceneMeshLatches_t
 - pxt, [25](#)
- SceneMeshLatchesAdd
 - Afterwarp.h, [275](#)
- SceneMeshLatchesClear
 - Afterwarp.h, [275](#)
- SceneMeshLatchesCopy
 - Afterwarp.h, [275](#)
- SceneMeshLatchesCreate
 - Afterwarp.h, [275](#)
- SceneMeshLatchesDestroy
 - Afterwarp.h, [275](#)
- SceneMeshLatchesErase
 - Afterwarp.h, [276](#)
- SceneMeshLatchesGetCount
 - Afterwarp.h, [276](#)
- SceneMeshLatchesGetLatch
 - Afterwarp.h, [276](#)
- SceneMeshLatchesGetLatchIndex
 - Afterwarp.h, [276](#)
- SceneMeshLatchesGetWaypointCouple
 - Afterwarp.h, [276](#)
- SceneMeshLatchesGetWaypointDistance
 - Afterwarp.h, [276](#)
- SceneMeshLatchesInvalidateWaypoints
 - Afterwarp.h, [277](#)
- SceneMeshLatchesLoadFromFile
 - Afterwarp.h, [277](#)
- SceneMeshLatchesLoadFromFileInMemory
 - Afterwarp.h, [277](#)
- SceneMeshLatchesSaveToFile
 - Afterwarp.h, [277](#)
- SceneMeshLatchesSetLatch
 - Afterwarp.h, [277](#)
- SceneMeshLatchesTakeAway
 - Afterwarp.h, [277](#)
- SceneMeshMaterial_t
 - pxt, [26](#)
- SceneMeshMaterialAddRange
 - Afterwarp.h, [278](#)
- SceneMeshMaterialClearRanges
 - Afterwarp.h, [278](#)
- SceneMeshMaterialCommit
 - Afterwarp.h, [278](#)
- SceneMeshMaterialCopy
 - Afterwarp.h, [278](#)
- SceneMeshMaterialGetName
 - Afterwarp.h, [278](#)
- SceneMeshMaterialGetRange
 - Afterwarp.h, [278](#)
- SceneMeshMaterialGetRangeCount
 - Afterwarp.h, [279](#)
- SceneMeshMaterialGetShading
 - Afterwarp.h, [279](#)
- SceneMeshMaterialGetTexture
 - Afterwarp.h, [279](#)
- SceneMeshMaterialRange, [95](#)
 - indexCount, [95](#)
 - indexStart, [95](#)
 - pxt, [26](#)
- SceneMeshMaterialReleaseTextures
 - Afterwarp.h, [279](#)
- SceneMeshMaterials_t
 - pxt, [26](#)
- SceneMeshMaterialsAdd
 - Afterwarp.h, [279](#)
- SceneMeshMaterialsClear
 - Afterwarp.h, [279](#)
- SceneMeshMaterialsCommit
 - Afterwarp.h, [280](#)
- SceneMeshMaterialsCopy
 - Afterwarp.h, [280](#)
- SceneMeshMaterialsCreate
 - Afterwarp.h, [280](#)
- SceneMeshMaterialsDestroy
 - Afterwarp.h, [280](#)
- SceneMeshMaterialsErase
 - Afterwarp.h, [280](#)
- SceneMeshMaterialSetRange
 - Afterwarp.h, [280](#)
- SceneMeshMaterialSetShading
 - Afterwarp.h, [281](#)
- SceneMeshMaterialSetTexture
 - Afterwarp.h, [281](#)
- SceneMeshMaterialsGetCount
 - Afterwarp.h, [281](#)
- SceneMeshMaterialsGetMaterial
 - Afterwarp.h, [281](#)
- SceneMeshMaterialsGetName
 - Afterwarp.h, [281](#)
- SceneMeshMaterialShading, [96](#)
 - ambient, [97](#)
 - bloom, [97](#)
 - diffuse, [97](#)
 - emissive, [97](#)
 - lighting, [97](#)
 - pxt, [26](#)
 - roughness, [97](#)

- specular, [97](#)
- specularExponent, [97](#)
- SceneMeshMaterialsSetName
 - Afterwarp.h, [281](#)
- SceneMeshMaterialsTakeAway
 - Afterwarp.h, [282](#)
- SceneMeshMaterialsTexturing
 - Afterwarp.h, [282](#)
- SceneMeshSetSlice
 - Afterwarp.h, [282](#)
- SceneMeshSetVertexElementsIndex
 - Afterwarp.h, [282](#)
- ScenePrepare
 - Afterwarp.h, [282](#)
- SceneRendering
 - Afterwarp.h, [282](#)
- SceneResetCache
 - Afterwarp.h, [283](#)
- SceneSamplerType
 - Afterwarp.Types.h, [401](#)
- SceneSetActiveVertexElements
 - Afterwarp.h, [283](#)
- SceneSetAttributes
 - Afterwarp.h, [283](#)
- SceneSetLights
 - Afterwarp.h, [283](#)
- SceneSetMaterial
 - Afterwarp.h, [283](#)
- SceneSetParallaxMappingParameters
 - Afterwarp.h, [283](#)
- SceneSetProjection
 - Afterwarp.h, [284](#)
- SceneSetShadowCastingAtlas
 - Afterwarp.h, [284](#)
- SceneSetTexture
 - Afterwarp.h, [284](#)
- SceneSetTextureCabinet
 - Afterwarp.h, [284](#)
- SceneSetToneMappingCoefficients
 - Afterwarp.h, [284](#)
- SceneSetVertexElements
 - Afterwarp.h, [284](#)
- SceneSetVertexElementsFromTextModeller
 - Afterwarp.h, [285](#)
- SceneSetView
 - Afterwarp.h, [285](#)
- SceneSetWorld
 - Afterwarp.h, [285](#)
- SceneTextureType
 - Afterwarp.Types.h, [401](#)
- second
 - ColorPair, [43](#)
- SELECTION_HIGHLIGHT_TEXTURE_HIGHLIGHT
 - Afterwarp.Types.h, [420](#)
- SELECTION_HIGHLIGHT_TEXTURE_HIGHLIGHT_MASK
 - Afterwarp.Types.h, [420](#)
- SelectionHighlight_t
 - pxt, [26](#)
- SelectionHighlightBegin
 - Afterwarp.h, [285](#)
- SelectionHighlightClear
 - Afterwarp.h, [285](#)
- SelectionHighlightCreate
 - Afterwarp.h, [285](#)
- SelectionHighlightDestroy
 - Afterwarp.h, [286](#)
- SelectionHighlightEnd
 - Afterwarp.h, [286](#)
- SelectionHighlightFilter
 - Afterwarp.h, [286](#)
- SelectionHighlightGetDepthStencil
 - Afterwarp.h, [286](#)
- SelectionHighlightGetDevice
 - Afterwarp.h, [286](#)
- SelectionHighlightGetGrayscale
 - Afterwarp.h, [286](#)
- SelectionHighlightGetParameters
 - Afterwarp.h, [287](#)
- SelectionHighlightGetSamples
 - Afterwarp.h, [287](#)
- SelectionHighlightGetSize
 - Afterwarp.h, [287](#)
- SelectionHighlightGetTexture
 - Afterwarp.h, [287](#)
- SelectionHighlightGetTextureCoordinates
 - Afterwarp.h, [287](#)
- SelectionHighlightParameters, [98](#)
 - blurOffset, [99](#)
 - blurPasses, [99](#)
 - glowIntensity, [99](#)
 - outlineColor, [99](#)
 - outlineStart, [99](#)
 - pxt, [26](#)
- SelectionHighlightRendering
 - Afterwarp.h, [287](#)
- SelectionHighlightSetParameters
 - Afterwarp.h, [288](#)
- SelectionHighlightSetSize
 - Afterwarp.h, [288](#)
- SelectionHighlightTextureType
 - Afterwarp.Types.h, [401](#)
- shader
 - ProgramElement, [78](#)
 - ProgramVariable, [79](#)
- SHADER_ELEMENT_CONSTANT_BUFFER
 - Afterwarp.Types.h, [413](#)
- SHADER_ELEMENT_RW_STRUCTURED_BUFFER
 - Afterwarp.Types.h, [413](#)
- SHADER_ELEMENT_RW_TYPED_BUFFER
 - Afterwarp.Types.h, [413](#)
- SHADER_ELEMENT_STRUCTURED_BUFFER
 - Afterwarp.Types.h, [413](#)
- SHADER_ELEMENT_TEXTURE
 - Afterwarp.Types.h, [413](#)
- SHADER_ELEMENT_TYPED_BUFFER
 - Afterwarp.Types.h, [413](#)

- SHADER_ELEMENT_UNDEFINED
 - Afterwarp.Types.h, [413](#)
- SHADER_INDEX_UNDEFINED
 - Afterwarp.Types.h, [413](#)
- SHADER_TYPE_COMPUTE
 - Afterwarp.Types.h, [412](#)
- SHADER_TYPE_DOMAIN
 - Afterwarp.Types.h, [412](#)
- SHADER_TYPE_FRAGMENT
 - Afterwarp.Types.h, [412](#)
- SHADER_TYPE_GEOMETRY
 - Afterwarp.Types.h, [412](#)
- SHADER_TYPE_HULL
 - Afterwarp.Types.h, [412](#)
- SHADER_TYPE_UNDEFINED
 - Afterwarp.Types.h, [412](#)
- SHADER_TYPE_VERTEX
 - Afterwarp.Types.h, [412](#)
- ShaderElement
 - Afterwarp.Types.h, [402](#)
- ShaderType
 - Afterwarp.Types.h, [402](#)
- shadowBrightness
 - FontEffect, [52](#)
- shadowCaster
 - SceneLight, [93](#)
- ShadowCaster_t
 - pxt, [26](#)
- ShadowCasterBegin
 - Afterwarp.h, [288](#)
- ShadowCasterClear
 - Afterwarp.h, [288](#)
- ShadowCasterEnd
 - Afterwarp.h, [288](#)
- ShadowCasterFilter
 - Afterwarp.h, [288](#)
- ShadowCasterGetAtlas
 - Afterwarp.h, [288](#)
- ShadowCasterGetDevice
 - Afterwarp.h, [289](#)
- ShadowCasterGetPosition
 - Afterwarp.h, [289](#)
- ShadowCasterGetSize
 - Afterwarp.h, [289](#)
- ShadowCasterGetTexture
 - Afterwarp.h, [289](#)
- ShadowCasterGetViewProjection
 - Afterwarp.h, [289](#)
- ShadowCasterRendering
 - Afterwarp.h, [289](#)
- ShadowCasterSetViewProjection
 - Afterwarp.h, [290](#)
- ShadowCastingAtlas_t
 - pxt, [26](#)
- ShadowCastingAtlasAdd
 - Afterwarp.h, [290](#)
- ShadowCastingAtlasBorderFill
 - Afterwarp.h, [290](#)
- ShadowCastingAtlasClear
 - Afterwarp.h, [290](#)
- ShadowCastingAtlasCreate
 - Afterwarp.h, [290](#)
- ShadowCastingAtlasDestroy
 - Afterwarp.h, [290](#)
- ShadowCastingAtlasErase
 - Afterwarp.h, [291](#)
- ShadowCastingAtlasGetCaster
 - Afterwarp.h, [291](#)
- ShadowCastingAtlasGetCount
 - Afterwarp.h, [291](#)
- ShadowCastingAtlasGetDevice
 - Afterwarp.h, [291](#)
- ShadowCastingAtlasGetPadding
 - Afterwarp.h, [291](#)
- ShadowCastingAtlasGetParameters
 - Afterwarp.h, [291](#)
- ShadowCastingAtlasGetSamples
 - Afterwarp.h, [292](#)
- ShadowCastingAtlasGetSize
 - Afterwarp.h, [292](#)
- ShadowCastingAtlasGetTechnique
 - Afterwarp.h, [292](#)
- ShadowCastingAtlasGetTexture
 - Afterwarp.h, [292](#)
- ShadowCastingAtlasSetParameters
 - Afterwarp.h, [292](#)
- shadowDistance
 - FontEffect, [52](#)
- shadowOpacity
 - FontEffect, [52](#)
- ShadowParameters, [99](#)
 - bias, [100](#)
 - bleedSigma, [100](#)
 - blurSamples, [100](#)
 - blurSigma, [100](#)
 - exponent1, [100](#)
 - exponent2, [101](#)
 - pxt, [27](#)
 - variance, [101](#)
- shadows
 - ObjectMaterial, [67](#)
 - OceanMaterial, [69](#)
- shadowSmoothness
 - FontEffect, [53](#)
- SignedDistanceField, [101](#)
 - outlineDistanceMaxSDF, [102](#)
 - outlineDistanceMinSDF, [102](#)
 - outlineOffsetSDF, [102](#)
 - pxt, [27](#)
 - signedFieldDistance, [102](#)
 - superSampleSDF, [102](#)
- signedFieldDistance
 - FontEffect, [53](#)
 - SignedDistanceField, [102](#)
- SineAccelerate
 - Afterwarp.h, [292](#)

- SineCycle
 - Afterwarp.h, [293](#)
- SineDecelerate
 - Afterwarp.h, [293](#)
- SineTransform
 - Afterwarp.h, [293](#)
- SineTwoCycle
 - Afterwarp.h, [293](#)
- size
 - ApplicationConfiguration, [35](#)
 - FontParameters, [55](#)
 - ProgramElement, [78](#)
 - ProgramVariable, [79](#)
- slant
 - FontParameters, [55](#)
- slopeDepthBias
 - RenderingState, [87](#)
- SmootherStep
 - Afterwarp.h, [293](#)
- SmoothStep
 - Afterwarp.h, [293](#)
- source
 - Blend, [39](#)
- SpatialFog_t
 - pxt, [27](#)
- SpatialFogCreate
 - Afterwarp.h, [294](#)
- SpatialFogDestroy
 - Afterwarp.h, [294](#)
- SpatialFogExecute
 - Afterwarp.h, [294](#)
- SpatialFogExecuteGlassy
 - Afterwarp.h, [294](#)
- SpatialFogGetDepthDistance
 - Afterwarp.h, [294](#)
- SpatialFogGetDevice
 - Afterwarp.h, [294](#)
- SpatialFogGetFormula
 - Afterwarp.h, [295](#)
- SpatialFogGetParameters
 - Afterwarp.h, [295](#)
- SpatialFogGetProjection
 - Afterwarp.h, [295](#)
- SpatialFogGetView
 - Afterwarp.h, [295](#)
- SpatialFogSetDepthDistance
 - Afterwarp.h, [295](#)
- SpatialFogSetFormula
 - Afterwarp.h, [295](#)
- SpatialFogSetParameters
 - Afterwarp.h, [296](#)
- SpatialFogSetProjection
 - Afterwarp.h, [296](#)
- SpatialFogSetView
 - Afterwarp.h, [296](#)
- specular
 - SceneMeshMaterialShading, [97](#)
- specularColor
 - ObjectMaterial, [67](#)
 - OceanMaterial, [70](#)
 - SceneLight, [93](#)
- specularExponent
 - ObjectMaterial, [67](#)
 - OceanMaterial, [70](#)
 - SceneLight, [93](#)
 - SceneMeshMaterialShading, [97](#)
- startup
 - ApplicationConfiguration, [35](#)
- state
 - ApplicationConfiguration, [35](#)
- states
 - RenderingState, [87](#)
- StatusFunc
 - pxt, [27](#)
- STENCIL_OP_DECREMENT
 - Afterwarp.Types.h, [409](#)
- STENCIL_OP_DECREMENT_WARP
 - Afterwarp.Types.h, [409](#)
- STENCIL_OP_INCREMENT
 - Afterwarp.Types.h, [409](#)
- STENCIL_OP_INCREMENT_WARP
 - Afterwarp.Types.h, [409](#)
- STENCIL_OP_INVERT
 - Afterwarp.Types.h, [409](#)
- STENCIL_OP_KEEP
 - Afterwarp.Types.h, [409](#)
- STENCIL_OP_REPLACE
 - Afterwarp.Types.h, [409](#)
- STENCIL_OP_ZERO
 - Afterwarp.Types.h, [409](#)
- stencilBack
 - RenderingState, [87](#)
- stencilFront
 - RenderingState, [87](#)
- StencilOp
 - Afterwarp.Types.h, [402](#)
- stencilRefMask
 - RenderingState, [87](#)
- stencilRefValue
 - RenderingState, [87](#)
- StencilState, [103](#)
 - depthFailOp, [103](#)
 - depthPassOp, [103](#)
 - failOp, [103](#)
 - func, [104](#)
- stencilWriteMask
 - RenderingState, [87](#)
- strength
 - AmbientOcclusionParameters, [33](#)
- stretch
 - FontParameters, [55](#)
- SubtractColors
 - Afterwarp.h, [296](#)
- SUPER_SAMPLE_SDF_NO_SUPER_SAMPLE
 - Afterwarp.Types.h, [416](#)
- SUPER_SAMPLE_SDF_SUPER_SAMPLE_16X

- Afterwarp.Types.h, [416](#)
- SUPER_SAMPLE_SDF_SUPER_SAMPLE_4X
 - Afterwarp.Types.h, [416](#)
- SuperSampleSDF
 - Afterwarp.Types.h, [402](#)
- superSampleSDF
 - SignedDistanceField, [102](#)
- Surface_t
 - pxt, [27](#)
- SurfaceClear
 - Afterwarp.h, [296](#)
- SurfaceClearWith
 - Afterwarp.h, [296](#)
- SurfaceConvertFormat
 - Afterwarp.h, [297](#)
- SurfaceCopyFrom
 - Afterwarp.h, [297](#)
- SurfaceCopyRect
 - Afterwarp.h, [297](#)
- SurfaceCreate
 - Afterwarp.h, [297](#)
- SurfaceCreateFromFile
 - Afterwarp.h, [297](#)
- SurfaceCreateFromFileInMemory
 - Afterwarp.h, [298](#)
- SurfaceDestroy
 - Afterwarp.h, [298](#)
- SurfaceEmpty
 - Afterwarp.h, [298](#)
- SurfaceFlip
 - Afterwarp.h, [298](#)
- SurfaceGetBits
 - Afterwarp.h, [298](#)
- SurfaceGetByteSize
 - Afterwarp.h, [299](#)
- SurfaceGetBytesPerPixel
 - Afterwarp.h, [299](#)
- SurfaceGetFormat
 - Afterwarp.h, [299](#)
- SurfaceGetHeight
 - Afterwarp.h, [299](#)
- SurfaceGetPitch
 - Afterwarp.h, [299](#)
- SurfaceGetPixel
 - Afterwarp.h, [299](#)
- SurfaceGetPixelBilinear
 - Afterwarp.h, [300](#)
- SurfaceGetPixelPtr
 - Afterwarp.h, [300](#)
- SurfaceGetPremultipliedAlpha
 - Afterwarp.h, [300](#)
- SurfaceGetScanline
 - Afterwarp.h, [300](#)
- SurfaceGetWidth
 - Afterwarp.h, [300](#)
- SurfaceHasAlphaChannel
 - Afterwarp.h, [300](#)
- SurfaceInvert
 - Afterwarp.h, [301](#)
- SurfaceMakeSignedDistanceField
 - Afterwarp.h, [301](#)
- SurfaceMirror
 - Afterwarp.h, [301](#)
- SurfacePremultiplyAlpha
 - Afterwarp.h, [301](#)
- SurfaceResetAlpha
 - Afterwarp.h, [301](#)
- SurfaceResize
 - Afterwarp.h, [302](#)
- SurfaceSaveToFile
 - Afterwarp.h, [302](#)
- SurfaceSaveToFileInMemory
 - Afterwarp.h, [302](#)
- SurfaceSetPixel
 - Afterwarp.h, [302](#)
- SurfaceSetPremultipliedAlpha
 - Afterwarp.h, [302](#)
- SurfaceShrinkFrom
 - Afterwarp.h, [303](#)
- SurfaceStretchRect
 - Afterwarp.h, [303](#)
- SurfaceStretchRectBilinear
 - Afterwarp.h, [303](#)
- SurfaceUnpremultiplyAlpha
 - Afterwarp.h, [303](#)
- SwapChainBegin
 - Afterwarp.h, [303](#)
- SwapChainCreate
 - Afterwarp.h, [304](#)
- SwapChainDestroy
 - Afterwarp.h, [304](#)
- SwapChainEnd
 - Afterwarp.h, [304](#)
- SwapChainGetDepthStencil
 - Afterwarp.h, [304](#)
- SwapChainGetDevice
 - Afterwarp.h, [304](#)
- SwapChainGetFormat
 - Afterwarp.h, [304](#)
- SwapChainGetHeight
 - Afterwarp.h, [305](#)
- SwapChainGetMultisamples
 - Afterwarp.h, [305](#)
- SwapChainGetVSync
 - Afterwarp.h, [305](#)
- SwapChainGetWidth
 - Afterwarp.h, [305](#)
- SwapChainGetWindowHandle
 - Afterwarp.h, [305](#)
- SwapChainResize
 - Afterwarp.h, [305](#)
- tag
 - PathElement, [74](#)
- tangent
 - MeshBufferEntry, [63](#)
- technique

- ObjectMaterial, 67
- TECHNIQUE_LIGHTING_COOK_TORRANCE
 - Afterwarp.Types.h, 421
- TECHNIQUE_LIGHTING_MINNAERT
 - Afterwarp.Types.h, 421
- TECHNIQUE_LIGHTING_OREN_NAYER
 - Afterwarp.Types.h, 421
- TECHNIQUE_LIGHTING_PHONG
 - Afterwarp.Types.h, 421
- TECHNIQUE_SHADOWS_ESM
 - Afterwarp.Types.h, 421
- TECHNIQUE_SHADOWS_ESM_WARP
 - Afterwarp.Types.h, 421
- TECHNIQUE_SHADOWS_EVSM
 - Afterwarp.Types.h, 421
- TECHNIQUE_SHADOWS_NONE
 - Afterwarp.Types.h, 421
- TechniqueLighting
 - Afterwarp.Types.h, 402
- TechniqueShadows
 - Afterwarp.Types.h, 402
- texCoord
 - MeshBufferEntry, 63
- TEXT_ALIGNMENT_END
 - Afterwarp.Types.h, 419
- TEXT_ALIGNMENT_MIDDLE
 - Afterwarp.Types.h, 419
- TEXT_ALIGNMENT_START
 - Afterwarp.Types.h, 419
- TextAlignment
 - Afterwarp.Types.h, 402
- TextEntryRect, 104
 - position, 105
 - pxt, 27
 - rectangle, 105
- TextModeller_t
 - pxt, 27
- TextModellerClear
 - Afterwarp.h, 305
- TextModellerCopyToMeshBuffer
 - Afterwarp.h, 306
- TextModellerCreate
 - Afterwarp.h, 306
- TextModellerDestroy
 - Afterwarp.h, 306
- TextModellerDraw
 - Afterwarp.h, 306
- TextModellerDrawCurved
 - Afterwarp.h, 306
- TextModellerDrawCurvedToCanvas
 - Afterwarp.h, 307
- TextModellerDrawCurvedToCanvasBuffer
 - Afterwarp.h, 307
- TextModellerDrawDepthCurved
 - Afterwarp.h, 307
- TextModellerDrawToCanvas
 - Afterwarp.h, 308
- TextModellerDrawToCanvasBuffer
 - Afterwarp.h, 308
- TextModellerExtent
 - Afterwarp.h, 308
- TextModellerExtentByShape
 - Afterwarp.h, 308
- TextModellerGetDevice
 - Afterwarp.h, 309
- TextModellerGetFontProvider
 - Afterwarp.h, 309
- TextModellerGetTransform
 - Afterwarp.h, 309
- TextModellerPrepare
 - Afterwarp.h, 309
- TextModellerRects
 - Afterwarp.h, 309
- TextModellerRender
 - Afterwarp.h, 309
- TextModellerReset
 - Afterwarp.h, 310
- TextModellerSetFontParameters
 - Afterwarp.h, 310
- TextModellerSetFontProvider
 - Afterwarp.h, 310
- TextModellerSetTransform
 - Afterwarp.h, 310
- TextRenderer_t
 - pxt, 27
- TextRendererCreate
 - Afterwarp.h, 310
- TextRendererDestroy
 - Afterwarp.h, 310
- TextRendererDraw
 - Afterwarp.h, 311
- TextRendererDrawAligned
 - Afterwarp.h, 311
- TextRendererDrawAlignedByPixels
 - Afterwarp.h, 311
- TextRendererDrawCentered
 - Afterwarp.h, 311
- TextRendererDrawCenteredByPixels
 - Afterwarp.h, 312
- TextRendererExtent
 - Afterwarp.h, 312
- TextRendererExtentByPixels
 - Afterwarp.h, 312
- TextRendererGetCanvas
 - Afterwarp.h, 312
- TextRendererGetFontParameters
 - Afterwarp.h, 312
- TextRendererRects
 - Afterwarp.h, 313
- TextRendererSetFontParameters
 - Afterwarp.h, 313
- TextRenderModifiers, 105
 - effect, 105
 - interleave, 105
 - pxt, 28
 - scale, 106

- verticalSpace, 106
- TEXTURE_ADDRESS_BORDER
 - Afterwarp.Types.h, 414
- TEXTURE_ADDRESS_CLAMP
 - Afterwarp.Types.h, 414
- TEXTURE_ADDRESS_MIRROR
 - Afterwarp.Types.h, 414
- TEXTURE_ADDRESS_MIRROR_ONCE
 - Afterwarp.Types.h, 414
- TEXTURE_ADDRESS_WRAP
 - Afterwarp.Types.h, 414
- TEXTURE_ATTRIBUTE_DRAWABLE
 - Afterwarp.Types.h, 414
- TEXTURE_ATTRIBUTE_DYNAMIC
 - Afterwarp.Types.h, 414
- TEXTURE_ATTRIBUTE_MIPMAPPING
 - Afterwarp.Types.h, 414
- TEXTURE_ATTRIBUTE_PREMULTIPLIED_ALPHA
 - Afterwarp.Types.h, 415
- TEXTURE_CABINET_ATTRIBUTE_AMBIENT_OCCLUSION
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_ATTRIBUTE_AMBIENT_OCCLUSION_TEXTURE
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_ATTRIBUTE_BLOOM
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_ATTRIBUTE_BLOOM_CUBIC
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_ATTRIBUTE_BLOOM_DOWNSCALE
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_ATTRIBUTE_GLASSY
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_ATTRIBUTE_GLASSY_FAST
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_ATTRIBUTE_GLASSY_FOG
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_ATTRIBUTE_GLASSY_FROSTED
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_ATTRIBUTE_LINEAR_DEPTHS
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_FILTER_TYPE_BLOOM
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_FILTER_TYPE_GLASSY
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_FILTER_TYPE_GLASSY_FOG
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_FILTER_TYPE_OCCLUSION
 - Afterwarp.Types.h, 423
- TEXTURE_CABINET_PASS_COLOR
 - Afterwarp.Types.h, 422
- TEXTURE_CABINET_PASS_DEPTH
 - Afterwarp.Types.h, 422
- TEXTURE_CABINET_PASS_GLASSY
 - Afterwarp.Types.h, 422
- TEXTURE_CABINET_TYPE_COLOR
 - Afterwarp.Types.h, 422
- TEXTURE_CABINET_TYPE_COLOR_HDR
 - Afterwarp.Types.h, 422
- TEXTURE_CABINET_TYPE_DEPTH
 - Afterwarp.Types.h, 422
- TEXTURE_CABINET_TYPE_LINEAR_DEPTHS
 - Afterwarp.Types.h, 422
- TEXTURE_CABINET_TYPE_NORMALS
 - Afterwarp.Types.h, 422
- TEXTURE_COMPUTE
 - Afterwarp.Types.h, 415
- TEXTURE_FIDELITY_EXTREME
 - Afterwarp.Types.h, 422
- TEXTURE_FIDELITY_HIGH
 - Afterwarp.Types.h, 422
- TEXTURE_FIDELITY_LOW
 - Afterwarp.Types.h, 422
- TEXTURE_FIDELITY_MEDIUM
 - Afterwarp.Types.h, 422
- TEXTURE_FILTER_ANISOTROPIC
 - Afterwarp.Types.h, 414
- TEXTURE_FILTER_LINEAR
 - Afterwarp.Types.h, 414
- TEXTURE_FILTER_NEAREST
 - Afterwarp.Types.h, 414
- TEXTURE_FILTER_NONE
 - Afterwarp.Types.h, 414
- TEXTURE_SCRATCH
 - Afterwarp.Types.h, 415
- Texture_t
 - pxt, 28
- TEXTURE_TYPE_CUBE_MAP
 - Afterwarp.Types.h, 415
- TEXTURE_TYPE_SURFACE
 - Afterwarp.Types.h, 415
- TEXTURE_TYPE_VOLUME
 - Afterwarp.Types.h, 415
- TextureAddress
 - Afterwarp.Types.h, 402
- TextureAttach
 - Afterwarp.h, 313
- TextureAttributes
 - pxt, 28
- TextureBegin
 - Afterwarp.h, 313
- TextureBind
 - Afterwarp.h, 313
- TextureCabinet_t
 - pxt, 28
- TextureCabinetAttributes
 - pxt, 28
- TextureCabinetBegin
 - Afterwarp.h, 314
- TextureCabinetClear
 - Afterwarp.h, 314
- TextureCabinetCreate
 - Afterwarp.h, 314
- TextureCabinetDestroy
 - Afterwarp.h, 314
- TextureCabinetEnd
 - Afterwarp.h, 314
- TextureCabinetFilter

- Afterwarp.h, [314](#)
- TextureCabinetFilterType
 - Afterwarp.Types.h, [403](#)
- TextureCabinetGetAmbientOcclusionParameters
 - Afterwarp.h, [315](#)
- TextureCabinetGetAttributes
 - Afterwarp.h, [315](#)
- TextureCabinetGetDepthStencil
 - Afterwarp.h, [315](#)
- TextureCabinetGetDevice
 - Afterwarp.h, [315](#)
- TextureCabinetGetFidelity
 - Afterwarp.h, [315](#)
- TextureCabinetGetFinalTexture
 - Afterwarp.h, [315](#)
- TextureCabinetGetSamples
 - Afterwarp.h, [316](#)
- TextureCabinetGetSize
 - Afterwarp.h, [316](#)
- TextureCabinetGetTexture
 - Afterwarp.h, [316](#)
- TextureCabinetGetToneMappingBloom
 - Afterwarp.h, [316](#)
- TextureCabinetPass
 - Afterwarp.Types.h, [403](#)
- TextureCabinetPresent
 - Afterwarp.h, [316](#)
- TextureCabinetRendering
 - Afterwarp.h, [316](#)
- TextureCabinetResolve
 - Afterwarp.h, [317](#)
- TextureCabinetRetrieveGlassyMetrics
 - Afterwarp.h, [317](#)
- TextureCabinetSetAmbientOcclusionParameters
 - Afterwarp.h, [317](#)
- TextureCabinetSetAttributes
 - Afterwarp.h, [317](#)
- TextureCabinetSetSize
 - Afterwarp.h, [317](#)
- TextureCabinetSetToneMappingBloom
 - Afterwarp.h, [317](#)
- TextureCabinetType
 - Afterwarp.Types.h, [403](#)
- TextureClear
 - Afterwarp.h, [318](#)
- TextureClearWith
 - Afterwarp.h, [318](#)
- TextureCopy
 - Afterwarp.h, [318](#)
- TextureCopyFromSurface
 - Afterwarp.h, [318](#)
- TextureCopyToSurface
 - Afterwarp.h, [318](#)
- TextureCreate
 - Afterwarp.h, [319](#)
- TextureCreateFromFile
 - Afterwarp.h, [319](#)
- TextureCreateFromFileInMemory
 - Afterwarp.h, [319](#)
- TextureCreateFromFileNormalsAndOcclusion
 - Afterwarp.h, [319](#)
- TextureCreateFromFileParallax
 - Afterwarp.h, [320](#)
- TextureCreateFromSurface
 - Afterwarp.h, [320](#)
- TextureDestroy
 - Afterwarp.h, [320](#)
- TextureDetach
 - Afterwarp.h, [320](#)
- TextureEnd
 - Afterwarp.h, [320](#)
- TextureFidelity
 - Afterwarp.Types.h, [403](#)
- TextureFilter
 - Afterwarp.Types.h, [403](#)
- TextureGenerateMipMaps
 - Afterwarp.h, [320](#)
- TextureGetDevice
 - Afterwarp.h, [321](#)
- TextureGetParameters
 - Afterwarp.h, [321](#)
- TextureGetPlatformHandle
 - Afterwarp.h, [321](#)
- TextureLoadFromFile
 - Afterwarp.h, [321](#)
- TextureParameters, [106](#)
 - attributes, [107](#)
 - depthStencil, [107](#)
 - format, [107](#)
 - height, [107](#)
 - layers, [107](#)
 - multisamples, [107](#)
 - pxt, [28](#)
 - type, [107](#)
 - width, [108](#)
- TextureResetCache
 - Afterwarp.h, [321](#)
- TextureRetrieve
 - Afterwarp.h, [321](#)
- TextureSaveToFile
 - Afterwarp.h, [322](#)
- TextureType
 - Afterwarp.Types.h, [403](#)
- TextureUnbind
 - Afterwarp.h, [322](#)
- TextureUpdate
 - Afterwarp.h, [322](#)
- Timer_t
 - pxt, [28](#)
- TimerCreate
 - Afterwarp.h, [322](#)
- TimerDestroy
 - Afterwarp.h, [322](#)
- TimerExtractTokens
 - Afterwarp.h, [323](#)
- TimerGetFrameRate

- Afterwarp.h, [323](#)
- TimerGetLatency
 - Afterwarp.h, [323](#)
- TimerGetSkippedTimeSlices
 - Afterwarp.h, [323](#)
- TimerGetSpeed
 - Afterwarp.h, [323](#)
- TimerGetTimeSlice
 - Afterwarp.h, [323](#)
- TimerNextSlice
 - Afterwarp.h, [324](#)
- TimerReset
 - Afterwarp.h, [324](#)
- TimerSetSpeed
 - Afterwarp.h, [324](#)
- TimerTrimSkippedTimeSlices
 - Afterwarp.h, [324](#)
- TimerUpdate
 - Afterwarp.h, [324](#)
- TimerUpdateNextSlice
 - Afterwarp.h, [324](#)
- toneFactors
 - ToneMappingBloom, [110](#)
- ToneMappingBloom, [108](#)
 - bloomBlurSamples, [109](#)
 - bloomBlurSigma, [109](#)
 - bloomCoefficients, [109](#)
 - bloomColorShift, [109](#)
 - bloomGamma, [110](#)
 - bloomThreshold, [110](#)
 - frostedPower, [110](#)
 - glassyBuckets, [110](#)
 - pxt, [28](#)
 - toneFactors, [110](#)
 - toneWhite, [110](#)
- toneWhite
 - ToneMappingBloom, [110](#)
- top
 - ImageRegion, [57](#)
 - Margins, [58](#)
 - Rect, [83](#)
 - RectF, [84](#)
- topLeft
 - ColorRect, [44](#)
 - Quad, [81](#)
- topRight
 - ColorRect, [45](#)
 - Quad, [81](#)
- TRIANGLE_FACE_BACK
 - Afterwarp.Types.h, [409](#)
- TRIANGLE_FACE_BOTH
 - Afterwarp.Types.h, [409](#)
- TRIANGLE_FACE_FRONT
 - Afterwarp.Types.h, [409](#)
- TRIANGLE_FACE_NONE
 - Afterwarp.Types.h, [409](#)
- TriangleFace
 - Afterwarp.Types.h, [403](#)
- type
 - SceneMeshLatch, [94](#)
 - TextureParameters, [107](#)
 - WidgetProperty, [116](#)
- UnicodeChar
 - pxt, [29](#)
- UnpremultiplyAlpha
 - Afterwarp.h, [325](#)
- UntypedHandle
 - pxt, [29](#)
- value
 - PathElement, [75](#)
- valueBool
 - WidgetPropertyData, [117](#)
- valueColor
 - WidgetPropertyData, [117](#)
- valueColorPair
 - WidgetPropertyData, [117](#)
- valueColorRect
 - WidgetPropertyData, [118](#)
- valueEnum
 - WidgetPropertyData, [118](#)
- valueFloat
 - WidgetPropertyData, [118](#)
- valueFont
 - WidgetPropertyData, [118](#)
- valueInt64
 - WidgetPropertyData, [118](#)
- valueInteger
 - WidgetPropertyData, [118](#)
- valueMargins
 - WidgetPropertyData, [118](#)
- valuePoint
 - WidgetPropertyData, [118](#)
- valueReal
 - WidgetPropertyData, [119](#)
- valueRect
 - WidgetPropertyData, [119](#)
- valueString
 - WidgetPropertyData, [119](#)
- valueVector
 - WidgetPropertyData, [119](#)
- variance
 - ShadowParameters, [101](#)
- Vector, [111](#)
 - Afterwarp.Types.h, [403](#)
 - x, [111](#)
 - y, [111](#)
 - z, [111](#)
- Vector4, [112](#)
 - Afterwarp.Types.h, [404](#)
 - w, [112](#)
 - x, [112](#)
 - y, [112](#)
 - z, [112](#)
- vectorAdd
 - Afterwarp.Vectors.h, [500](#)

- vectorAdd3D
 - Afterwarp.Vectors.h, [500](#)
- vectorAngle
 - Afterwarp.Vectors.h, [501](#)
- vectorAxisW3D
 - Afterwarp.Vectors.h, [511](#)
- vectorAxisX
 - Afterwarp.Vectors.h, [511](#)
- vectorAxisX3D
 - Afterwarp.Vectors.h, [511](#)
- vectorAxisY
 - Afterwarp.Vectors.h, [511](#)
- vectorAxisY3D
 - Afterwarp.Vectors.h, [512](#)
- vectorAxisZ
 - Afterwarp.Vectors.h, [512](#)
- vectorAxisZ3D
 - Afterwarp.Vectors.h, [512](#)
- vectorCross
 - Afterwarp.Vectors.h, [501](#)
- vectorDistance
 - Afterwarp.Vectors.h, [501](#)
- vectorDistance3D
 - Afterwarp.Vectors.h, [501](#)
- vectorDivide
 - Afterwarp.Vectors.h, [501](#)
- vectorDivide3D
 - Afterwarp.Vectors.h, [501](#)
- vectorDot
 - Afterwarp.Vectors.h, [502](#)
- vectorDot3D
 - Afterwarp.Vectors.h, [502](#)
- vectorEmpty
 - Afterwarp.Vectors.h, [502](#)
- vectorEmpty3D
 - Afterwarp.Vectors.h, [502](#)
- vectorEpsilon
 - Afterwarp.Vectors.h, [512](#)
- vectorEquals
 - Afterwarp.Vectors.h, [502](#)
- vectorEquals3D
 - Afterwarp.Vectors.h, [502](#)
- vectorFromValue3D
 - Afterwarp.Vectors.h, [503](#)
- vectorLength
 - Afterwarp.Vectors.h, [503](#)
- vectorLength3D
 - Afterwarp.Vectors.h, [503](#)
- vectorLerp
 - Afterwarp.Vectors.h, [503](#)
- vectorLerp3D
 - Afterwarp.Vectors.h, [503](#)
- vectorMultiply
 - Afterwarp.Vectors.h, [503](#)
- vectorMultiply3D
 - Afterwarp.Vectors.h, [504](#)
- vectorNegate
 - Afterwarp.Vectors.h, [504](#)
- vectorNegate3D
 - Afterwarp.Vectors.h, [504](#)
- vectorNormalize
 - Afterwarp.Vectors.h, [504](#)
- vectorNormalize3D
 - Afterwarp.Vectors.h, [504](#)
- vectorParallel
 - Afterwarp.Vectors.h, [504](#)
- vectorPerpendicular
 - Afterwarp.Vectors.h, [505](#)
- vectorProject
 - Afterwarp.Vectors.h, [505](#)
- vectorProjectTarget
 - Afterwarp.Vectors.h, [505](#)
- vectorReflect
 - Afterwarp.Vectors.h, [505](#)
- vectorRescale
 - Afterwarp.Vectors.h, [505](#)
- vectorRescale3D
 - Afterwarp.Vectors.h, [505](#)
- vectorSubtract
 - Afterwarp.Vectors.h, [506](#)
- vectorSubtract3D
 - Afterwarp.Vectors.h, [506](#)
- vectorTransform
 - Afterwarp.Vectors.h, [506](#)
- vectorTransform3D
 - Afterwarp.Vectors.h, [506](#)
- vectorUnity
 - Afterwarp.Vectors.h, [512](#)
- vectorUnity3D
 - Afterwarp.Vectors.h, [512](#)
- vectorValue
 - Afterwarp.Vectors.h, [506](#)
- vectorZero
 - Afterwarp.Vectors.h, [512](#)
- vectorZero3D
 - Afterwarp.Vectors.h, [513](#)
- VERTEX_CHANNEL_INDEX_BUFFER
 - Afterwarp.Types.h, [413](#)
- VERTEX_CHANNEL_NO_BUFFERS
 - Afterwarp.Types.h, [413](#)
- VERTEX_CHANNEL_NORMALIZED
 - Afterwarp.Types.h, [413](#)
- vertexCount
 - MeshMetaTagPortion, [65](#)
- VertexElement, [113](#)
 - channel, [113](#)
 - count, [113](#)
 - format, [113](#)
 - name, [114](#)
 - offset, [114](#)
 - pxt, [29](#)
- VertexElementsEstimatePitch
 - Afterwarp.h, [325](#)
- verticalSpace
 - TextRenderModifiers, [106](#)
- VolumeCalculateNearFarPlanes

- Afterwarp.h, [325](#)
- VolumeCalculateVisibleFrame
 - Afterwarp.h, [325](#)
- w
 - Quaternion, [82](#)
 - Vector4, [112](#)
- waveSize
 - OceanWavesParameters, [71](#)
- weight
 - FontParameters, [55](#)
- WIDGET_ALIGNMENT_BOTTOM
 - Afterwarp.Types.h, [432](#)
- WIDGET_ALIGNMENT_CLIENT
 - Afterwarp.Types.h, [431](#)
- WIDGET_ALIGNMENT_LEFT
 - Afterwarp.Types.h, [431](#)
- WIDGET_ALIGNMENT_NONE
 - Afterwarp.Types.h, [431](#)
- WIDGET_ALIGNMENT_RIGHT
 - Afterwarp.Types.h, [432](#)
- WIDGET_ALIGNMENT_TOP
 - Afterwarp.Types.h, [432](#)
- WIDGET_MANAGER_ATTRIBUTE_COMPOSITION
 - Afterwarp.Types.h, [437](#)
- WIDGET_MANAGER_ATTRIBUTE_DESIGN
 - Afterwarp.Types.h, [437](#)
- WIDGET_MANAGER_TEXTURE_TYPE_AUXILIARY
 - Afterwarp.Types.h, [433](#)
- WIDGET_MANAGER_TEXTURE_TYPE_CANVAS
 - Afterwarp.Types.h, [433](#)
- WIDGET_MANAGER_TEXTURE_TYPE_SCREEN
 - Afterwarp.Types.h, [433](#)
- WIDGET_PROPERTY_ATTRIBUTE_ASSIGNED
 - Afterwarp.Types.h, [433](#)
- WIDGET_PROPERTY_BEHAVIOR_EVENT
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_BEHAVIOR_INDIRECT
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_BEHAVIOR_NORMAL
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_BEHAVIOR_STATIC
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_BEHAVIOR_VOLATILE
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_BOOLEAN
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_COLOR
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_COLOR_PAIR
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_COLOR_RECT
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_ENUMERATION
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_FLOAT
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_FONT
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_INT64
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_INTEGER
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_MARGINS
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_NONE
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_POINT
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_REAL
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_RECT
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_STRING
 - Afterwarp.Types.h, [432](#)
- WIDGET_PROPERTY_TYPE_VECTOR
 - Afterwarp.Types.h, [432](#)
- Widget_t
 - pxt, [29](#)
- WidgetAcceptKey
 - Afterwarp.h, [325](#)
- WidgetAcceptMouse
 - Afterwarp.h, [325](#)
- WidgetAccomodate
 - Afterwarp.h, [326](#)
- WidgetAlignment
 - Afterwarp.Types.h, [404](#)
- WidgetBringToFront
 - Afterwarp.h, [326](#)
- WidgetCreate
 - Afterwarp.h, [326](#)
- WidgetDestroy
 - Afterwarp.h, [326](#)
- WidgetExternalEvent
 - pxt, [29](#)
- WidgetFindAt
 - Afterwarp.h, [326](#)
- WidgetGetChildByIndex
 - Afterwarp.h, [326](#)
- WidgetGetChildCount
 - Afterwarp.h, [327](#)
- WidgetGetChildIndex
 - Afterwarp.h, [327](#)
- WidgetGetExternalEvent
 - Afterwarp.h, [327](#)
- WidgetGetManager
 - Afterwarp.h, [327](#)
- WidgetGetParent
 - Afterwarp.h, [327](#)
- WidgetGetPayload
 - Afterwarp.h, [327](#)
- WidgetGetProperty
 - Afterwarp.h, [328](#)
- WidgetGetPropertyAsString
 - Afterwarp.h, [328](#)
- WidgetGetPropertyCount
 - Afterwarp.h, [328](#)

- WidgetInvalidate
 - Afterwarp.h, [328](#)
- WidgetInvokeEvent
 - Afterwarp.h, [328](#)
- WidgetLocalToScreen
 - Afterwarp.h, [328](#)
- WidgetManagerAttribute
 - Afterwarp.Types.h, [437](#)
- WidgetManagerBatchCount
 - Afterwarp.h, [329](#)
- WidgetManagerCreate
 - Afterwarp.h, [329](#)
- WidgetManagerGetApplication
 - Afterwarp.h, [329](#)
- WidgetManagerGetAttributes
 - Afterwarp.h, [329](#)
- WidgetManagerGetFormat
 - Afterwarp.h, [329](#)
- WidgetManagerGetMultisamples
 - Afterwarp.h, [329](#)
- WidgetManagerGetTextRenderer
 - Afterwarp.h, [330](#)
- WidgetManagerGetTexture
 - Afterwarp.h, [330](#)
- WidgetManagerGetWidgetByName
 - Afterwarp.h, [330](#)
- WidgetManagerGetWidgetByPayload
 - Afterwarp.h, [330](#)
- WidgetManagerPresent
 - Afterwarp.h, [330](#)
- WidgetManagerTextureType
 - Afterwarp.Types.h, [404](#)
- WidgetProperty, [114](#)
 - attributes, [115](#)
 - behavior, [115](#)
 - data, [115](#)
 - location, [115](#)
 - name, [115](#)
 - pxt, [29](#)
 - reserved, [116](#)
 - type, [116](#)
- WidgetPropertyBehavior
 - Afterwarp.Types.h, [404](#)
- WidgetPropertyData, [116](#)
 - pxt, [29](#)
 - valueBool, [117](#)
 - valueColor, [117](#)
 - valueColorPair, [117](#)
 - valueColorRect, [118](#)
 - valueEnum, [118](#)
 - valueFloat, [118](#)
 - valueFont, [118](#)
 - valueInt64, [118](#)
 - valueInteger, [118](#)
 - valueMargins, [118](#)
 - valuePoint, [118](#)
 - valueReal, [119](#)
 - valueRect, [119](#)
 - valueString, [119](#)
 - valueVector, [119](#)
- WidgetPropertyIdentify
 - Afterwarp.h, [330](#)
- WidgetPropertyType
 - Afterwarp.Types.h, [404](#)
- WidgetScreenToLocal
 - Afterwarp.h, [331](#)
- WidgetSendToBack
 - Afterwarp.h, [331](#)
- WidgetSetExternalEvent
 - Afterwarp.h, [331](#)
- WidgetSetParent
 - Afterwarp.h, [331](#)
- WidgetSetProperty
 - Afterwarp.h, [331](#)
- WidgetUpdate
 - Afterwarp.h, [332](#)
- width
 - ImageRegion, [57](#)
 - Rect, [83](#)
 - RectF, [84](#)
 - TextureParameters, [108](#)
- win
 - ApplicationConfiguration, [36](#)
- wind
 - OceanWavesParameters, [71](#)
- windowClassName
 - ApplicationConfiguration, [36](#)
- x
 - MeshAligns, [61](#)
 - Point, [75](#)
 - PointF, [76](#)
 - Quaternion, [82](#)
 - Vector, [111](#)
 - Vector4, [112](#)
- y
 - MeshAligns, [61](#)
 - Point, [75](#)
 - PointF, [76](#)
 - Quaternion, [82](#)
 - Vector, [111](#)
 - Vector4, [112](#)
- z
 - MeshAligns, [61](#)
 - Quaternion, [82](#)
 - Vector, [111](#)
 - Vector4, [112](#)