GPU Benchmarking tool

Generated by Doxygen 1.8.14

Contents

1 Namespace Index **1**

1.1 Namespace List 1

2 Hierarchical Index **3**

2.1 Class Hierarchy 3

3 Class Index **5**

3.1 Class List 5

4 File Index **7**

4.1 File List 7

5 Namespace Documentation **9**

5.1 math Namespace Reference 9

5.2 Ui Namespace Reference 9

5.2.1 Detailed Description 9

6 Class Documentation **11**

6.1 Chart Struct Reference 11

6.1.1 Detailed Description 11

6.1.2 Member Data Documentation 11

6.1.2.1 axisX 11

6.1.2.2 axisY 12

6.1.2.3 chart 12

6.1.2.4 chartView 12

6.1.2.5 series 12

6.2 GLDescriptions Struct Reference . . . 12
6.2.1 Detailed Description . . . 23
6.2.2 Member Data Documentation . . . 23
6.2.2.1 gl_active_texture . . . 23
6.2.2.2 gl_aliased_line_width_range . . . 23
6.2.2.3 gl_array_buffer_binding . . . 23
6.2.2.4 gl_blend . . . 23
6.2.2.5 gl_blend_color . . . 23
6.2.2.6 gl_blend_dst_alpha . . . 24
6.2.2.7 gl_blend_dst_rgb . . . 24
6.2.2.8 gl_blend_equation_alpha . . . 24
6.2.2.9 gl_blend_equation_rgb . . . 24
6.2.2.10 gl_blend_src_alpha . . . 24
6.2.2.11 gl_blend_src_rgb . . . 24
6.2.2.12 gl_color_clear_value . . . 25
6.2.2.13 gl_color_logic_op . . . 25
6.2.2.14 gl_color_writemask . . . 25
6.2.2.15 gl_compressed_texture_formats . . . 25
6.2.2.16 gl_context_flags . . . 25
6.2.2.17 gl_cull_face . . . 25
6.2.2.18 gl_cull_face_mode . . . 26
6.2.2.19 gl_current_program . . . 26
6.2.2.20 gl_debug_group_stack_depth . . . 26
6.2.2.21 gl_depth_clear_value . . . 26
6.2.2.22 gl_depth_func . . . 26
6.2.2.23 gl_depth_range . . . 26
6.2.2.24 gl_depth_test . . . 27
6.2.2.25 gl_depth_writemask . . . 27
6.2.2.26 gl_dispatch_indirect_buffer_binding . . . 27
6.2.2.27 gl_dither . . . 27

6.2.2.28 gl_doublebuffer . . . 27
6.2.2.29 gl_draw_buffer . . . 27
6.2.2.30 gl_draw_framebuffer_binding . . . 28
6.2.2.31 gl_element_array_buffer_binding . . . 28
6.2.2.32 gl_fragment_shader_derivative_hint . . . 28
6.2.2.33 gl_implementation_color_read_format . . . 28
6.2.2.34 gl_implementation_color_read_type . . . 28
6.2.2.35 gl_layer_provoking_vertex . . . 28
6.2.2.36 gl_line_smooth . . . 29
6.2.2.37 gl_line_smooth_hint . . . 29
6.2.2.38 gl_line_width . . . 29
6.2.2.39 gl_logic_op_mode . . . 29
6.2.2.40 gl_major_version . . . 29
6.2.2.41 gl_max_3d_texture_size . . . 29
6.2.2.42 gl_max_array_texture_layers . . . 30
6.2.2.43 gl_max_clip_distances . . . 30
6.2.2.44 gl_max_color_texture_samples . . . 30
6.2.2.45 gl_max_combined_atomic_counters . . . 30
6.2.2.46 gl_max_combined_compute_uniform_components . . . 30
6.2.2.47 gl_max_combined_fragment_uniform_components . . . 30
6.2.2.48 gl_max_combined_geometry_uniform_components . . . 31
6.2.2.49 gl_max_combined_shader_storage_blocks . . . 31
6.2.2.50 gl_max_combined_texture_image_units . . . 31
6.2.2.51 gl_max_combined_uniform_blocks . . . 31
6.2.2.52 gl_max_combined_vertex_uniform_components . . . 31
6.2.2.53 gl_max_compute_atomic_counter_buffers . . . 31
6.2.2.54 gl_max_compute_atomic_counters . . . 32
6.2.2.55 gl_max_compute_shader_storage_blocks . . . 32
6.2.2.56 gl_max_compute_texture_image_units . . . 32
6.2.2.57 gl_max_compute_uniform_blocks . . . 32

6.2.2.58 gl_max_compute_uniform_components . . . 32
6.2.2.59 gl_max_compute_work_group_count . . . 32
6.2.2.60 gl_max_compute_work_group_size . . . 33
6.2.2.61 gl_max_cube_map_texture_size . . . 33
6.2.2.62 gl_max_debug_group_stack_depth . . . 33
6.2.2.63 gl_max_depth_texture_samples . . . 33
6.2.2.64 gl_max_draw_buffers . . . 33
6.2.2.65 gl_max_dual_source_draw_buffers . . . 33
6.2.2.66 gl_max_element_index . . . 34
6.2.2.67 gl_max_elements_indices . . . 34
6.2.2.68 gl_max_elements_vertices . . . 34
6.2.2.69 gl_max_fragment_atomic_counters . . . 34
6.2.2.70 gl_max_fragment_input_components . . . 34
6.2.2.71 gl_max_fragment_shader_storage_blocks . . . 34
6.2.2.72 gl_max_fragment_uniform_blocks . . . 34
6.2.2.73 gl_max_fragment_uniform_components . . . 35
6.2.2.74 gl_max_fragment_uniform_vectors . . . 35
6.2.2.75 gl_max_framebuffer_height . . . 35
6.2.2.76 gl_max_framebuffer_layers . . . 35
6.2.2.77 gl_max_framebuffer_samples . . . 35
6.2.2.78 gl_max_framebuffer_width . . . 35
6.2.2.79 gl_max_geometry_atomic_counters . . . 36
6.2.2.80 gl_max_geometry_input_components . . . 36
6.2.2.81 gl_max_geometry_output_components . . . 36
6.2.2.82 gl_max_geometry_shader_storage_blocks . . . 36
6.2.2.83 gl_max_geometry_texture_image_units . . . 36
6.2.2.84 gl_max_geometry_uniform_blocks . . . 36
6.2.2.85 gl_max_geometry_uniform_components . . . 37
6.2.2.86 gl_max_integer_samples . . . 37
6.2.2.87 gl_max_label_length . . . 37

6.2.2.88 gl_max_program_texel_offset . . . 37
6.2.2.89 gl_max_rectangle_texture_size . . . 37
6.2.2.90 gl_max_renderbuffer_size . . . 37
6.2.2.91 gl_max_sample_mask_words . . . 38
6.2.2.92 gl_max_server_wait_timeout . . . 38
6.2.2.93 gl_max_shader_storage_buffer_bindings . . . 38
6.2.2.94 gl_max_tess_control_atomic_counters . . . 38
6.2.2.95 gl_max_tess_control_shader_storage_blocks . . . 38
6.2.2.96 gl_max_tess_evaluation_atomic_counters . . . 38
6.2.2.97 gl_max_tess_evaluation_shader_storage_blocks . . . 39
6.2.2.98 gl_max_texture_buffer_size . . . 39
6.2.2.99 gl_max_texture_image_units . . . 39
6.2.2.100 gl_max_texture_lod_bias . . . 39
6.2.2.101 gl_max_texture_size . . . 39
6.2.2.102 gl_max_uniform_block_size . . . 39
6.2.2.103 gl_max_uniform_buffer_bindings . . . 40
6.2.2.104 gl_max_uniform_locations . . . 40
6.2.2.105 gl_max_varying_components . . . 40
6.2.2.106 gl_max_varying_floats . . . 40
6.2.2.107 gl_max_varying_vectors . . . 40
6.2.2.108 gl_max_vertex_atomic_counters . . . 40
6.2.2.109 gl_max_vertex_attrib_bindings . . . 41
6.2.2.110 gl_max_vertex_attrib_relative_offset . . . 41
6.2.2.111 gl_max_vertex_attribs . . . 41
6.2.2.112 gl_max_vertex_output_components . . . 41
6.2.2.113 gl_max_vertex_shader_storage_blocks . . . 41
6.2.2.114 gl_max_vertex_texture_image_units . . . 41
6.2.2.115 gl_max_vertex_uniform_blocks . . . 42
6.2.2.116 gl_max_vertex_uniform_components . . . 42
6.2.2.117 gl_max_vertex_uniform_vectors . . . 42

6.2.2.118 gl_max_viewport_dims . . . 42
6.2.2.119 gl_max_viewports . . . 42
6.2.2.120 gl_min_map_buffer_alignment . . . 42
6.2.2.121 gl_min_program_texel_offset . . . 43
6.2.2.122 gl_minor_version . . . 43
6.2.2.123 gl_num_compressed_texture_formats . . . 43
6.2.2.124 gl_num_extensions . . . 43
6.2.2.125 gl_num_program_binary_formats . . . 43
6.2.2.126 gl_num_shader_binary_formats . . . 43
6.2.2.127 gl_pack_alignment . . . 44
6.2.2.128 gl_pack_image_height . . . 44
6.2.2.129 gl_pack_lsb_first . . . 44
6.2.2.130 gl_pack_row_length . . . 44
6.2.2.131 gl_pack_skip_images . . . 44
6.2.2.132 gl_pack_skip_pixels . . . 44
6.2.2.133 gl_pack_skip_rows . . . 45
6.2.2.134 gl_pack_swap_bytes . . . 45
6.2.2.135 gl_pixel_pack_buffer_binding . . . 45
6.2.2.136 gl_pixel_unpack_buffer_binding . . . 45
6.2.2.137 gl_point_fade_threshold_size . . . 45
6.2.2.138 gl_point_size . . . 45
6.2.2.139 gl_point_size_granularity . . . 46
6.2.2.140 gl_point_size_range . . . 46
6.2.2.141 gl_polygon_offset_factor . . . 46
6.2.2.142 gl_polygon_offset_fill . . . 46
6.2.2.143 gl_polygon_offset_line . . . 46
6.2.2.144 gl_polygon_offset_point . . . 46
6.2.2.145 gl_polygon_offset_units . . . 47
6.2.2.146 gl_polygon_smooth . . . 47
6.2.2.147 gl_polygon_smooth_hint . . . 47

6.2.2.148 gl_primitive_restart_index 47
6.2.2.149 gl_program_binary_formats 47
6.2.2.150 gl_program_pipeline_binding 47
6.2.2.151 gl_program_point_size 48
6.2.2.152 gl_provoking_vertex 48
6.2.2.153 gl_read_buffer 48
6.2.2.154 gl_read_framebuffer_binding 48
6.2.2.155 gl_renderbuffer_binding 48
6.2.2.156 gl_sample_buffers 48
6.2.2.157 gl_sample_coverage_invert 49
6.2.2.158 gl_sample_coverage_value 49
6.2.2.159 gl_sampler_binding 49
6.2.2.160 gl_samples 49
6.2.2.161 gl_scissor_box 49
6.2.2.162 gl_scissor_test 49
6.2.2.163 gl_shader_compiler 50
6.2.2.164 gl_shader_storage_buffer_binding 50
6.2.2.165 gl_shader_storage_buffer_offset_alignment 50
6.2.2.166 gl_shader_storage_buffer_size 50
6.2.2.167 gl_shader_storage_buffer_start 50
6.2.2.168 gl_smooth_line_width_granularity 51
6.2.2.169 gl_smooth_line_width_range 51
6.2.2.170 gl_stencil_back_fail 51
6.2.2.171 gl_stencil_back_func 51
6.2.2.172 gl_stencil_back_pass_depth_fail 51
6.2.2.173 gl_stencil_back_pass_depth_pass 51
6.2.2.174 gl_stencil_back_ref 52
6.2.2.175 gl_stencil_back_value_mask 52
6.2.2.176 gl_stencil_back_writemask 52
6.2.2.177 gl_stencil_clear_value 52

6.2.2.178 gl_stencil_fail . . . 52
6.2.2.179 gl_stencil_func . . . 52
6.2.2.180 gl_stencil_pass_depth_fail . . . 53
6.2.2.181 gl_stencil_pass_depth_pass . . . 53
6.2.2.182 gl_stencil_ref . . . 53
6.2.2.183 gl_stencil_test . . . 53
6.2.2.184 gl_stencil_value_mask . . . 53
6.2.2.185 gl_stencil_writemask . . . 54
6.2.2.186 gl_stereo . . . 54
6.2.2.187 gl_subpixel_bits . . . 54
6.2.2.188 gl_texture_binding_1d . . . 54
6.2.2.189 gl_texture_binding_1d_array . . . 54
6.2.2.190 gl_texture_binding_2d . . . 54
6.2.2.191 gl_texture_binding_2d_array . . . 55
6.2.2.192 gl_texture_binding_2d_multisample . . . 55
6.2.2.193 gl_texture_binding_2d_multisample_array . . . 55
6.2.2.194 gl_texture_binding_3d . . . 55
6.2.2.195 gl_texture_binding_buffer . . . 55
6.2.2.196 gl_texture_binding_cube_map . . . 55
6.2.2.197 gl_texture_binding_rectangle . . . 56
6.2.2.198 gl_texture_buffer_offset_alignment . . . 56
6.2.2.199 gl_texture_compression_hint . . . 56
6.2.2.200 gl_timestamp . . . 56
6.2.2.201 gl_transform_feedback_buffer_binding . . . 56
6.2.2.202 gl_transform_feedback_buffer_size . . . 56
6.2.2.203 gl_transform_feedback_buffer_start . . . 57
6.2.2.204 gl_uniform_buffer_binding . . . 57
6.2.2.205 gl_uniform_buffer_offset_alignment . . . 57
6.2.2.206 gl_uniform_buffer_size . . . 57
6.2.2.207 gl_uniform_buffer_start . . . 57

6.2.2.208 gl_unpack_alignment . . . 58
6.2.2.209 gl_unpack_image_height . . . 58
6.2.2.210 gl_unpack_lsb_first . . . 58
6.2.2.211 gl_unpack_row_length . . . 58
6.2.2.212 gl_unpack_skip_images . . . 58
6.2.2.213 gl_unpack_skip_pixels . . . 58
6.2.2.214 gl_unpack_skip_rows . . . 59
6.2.2.215 gl_unpack_swap_bytes . . . 59
6.2.2.216 gl_vertex_array_binding . . . 59
6.2.2.217 gl_vertex_binding_divisor . . . 59
6.2.2.218 gl_vertex_binding_offset . . . 59
6.2.2.219 gl_vertex_binding_stride . . . 59
6.2.2.220 gl_viewport . . . 60
6.2.2.221 gl_viewport_bounds_range . . . 60
6.2.2.222 gl_viewport_index_provoking_vertex . . . 60
6.2.2.223 gl_viewport_subpixel_bits . . . 60
6.3 GLWidget Class Reference . . . 60
6.3.1 Detailed Description . . . 61
6.3.2 Constructor & Destructor Documentation . . . 61
6.3.2.1 GLWidget() . . . 61
6.3.2.2 ~GLWidget() . . . 61
6.3.3 Member Function Documentation . . . 62
6.3.3.1 initializeGL() . . . 62
6.3.3.2 paintGL() . . . 62
6.3.3.3 resizeGL() . . . 62
6.3.3.4 setProfiler() . . . 62
6.3.3.5 setScene() . . . 63
6.3.3.6 setTargetFramerate() . . . 63
6.4 GLWindow Class Reference . . . 63
6.4.1 Detailed Description . . . 64

6.4.2 Constructor & Destructor Documentation . . . 64
6.4.2.1 GLWindow() . . . 64
6.4.2.2 ~GLWindow() . . . 64
6.4.3 Member Function Documentation . . . 64
6.4.3.1 setProfiler() . . . 64
6.4.3.2 setTargetFramerate() . . . 65
6.5 MainWindow Class Reference . . . 65
6.5.1 Detailed Description . . . 65
6.5.2 Constructor & Destructor Documentation . . . 66
6.5.2.1 MainWindow() . . . 66
6.5.2.2 ~MainWindow() . . . 66
6.6 Mesh Class Reference . . . 66
6.6.1 Detailed Description . . . 67
6.6.2 Constructor & Destructor Documentation . . . 67
6.6.2.1 Mesh() . . . 67
6.6.3 Member Function Documentation . . . 67
6.6.3.1 load() . . . 67
6.6.4 Member Data Documentation . . . 67
6.6.4.1 aabb . . . 68
6.6.4.2 geom . . . 68
6.6.4.3 vertexCount . . . 68
6.7 Model Class Reference . . . 68
6.7.1 Detailed Description . . . 69
6.7.2 Constructor & Destructor Documentation . . . 69
6.7.2.1 Model() [1/2] . . . 69
6.7.2.2 Model() [2/2] . . . 69
6.7.2.3 ~Model() . . . 69
6.7.3 Member Function Documentation . . . 69
6.7.3.1 setup() . . . 69
6.7.4 Member Data Documentation . . . 70

6.7.4.1 indices . . . 70
6.7.4.2 modelObject . . . 70
6.7.4.3 obj . . . 70
6.7.4.4 shaderProgram . . . 70
6.7.4.5 texture . . . 70
6.7.4.6 uvsBuffer . . . 70
6.7.4.7 vertexBuffer . . . 70
6.8 Renderer Class Reference . . . 71
6.8.1 Detailed Description . . . 71
6.8.2 Constructor & Destructor Documentation . . . 71
6.8.2.1 Renderer() . . . 71
6.8.3 Member Function Documentation . . . 72
6.8.3.1 initResources() . . . 72
6.8.3.2 initSwapChainResources() . . . 72
6.8.3.3 releaseResources() . . . 72
6.8.3.4 releaseSwapChainResources() . . . 72
6.8.3.5 startNextFrame() . . . 73
6.9 Shader Class Reference . . . 73
6.9.1 Detailed Description . . . 73
6.9.2 Constructor & Destructor Documentation . . . 73
6.9.2.1 Shader() . . . 73
6.9.3 Member Function Documentation . . . 74
6.9.3.1 isValid() . . . 74
6.9.3.2 load() . . . 74
6.9.4 Member Data Documentation . . . 74
6.9.4.1 shaderModule . . . 74
6.10 math::Vector2 Class Reference . . . 75
6.10.1 Detailed Description . . . 76
6.10.2 Constructor & Destructor Documentation . . . 76
6.10.2.1 Vector2() [1/2] . . . 76

6.10.2.2 Vector2() [2/2] . . . 76
6.10.2.3 ~Vector2() . . . 76
6.10.3 Member Function Documentation . . . 76
6.10.3.1 dot() . . . 76
6.10.3.2 magnitude() . . . 77
6.10.3.3 magnitudeSquared() . . . 77
6.10.3.4 normal() . . . 77
6.10.3.5 normalise() . . . 77
6.10.3.6 operator*() . . . 77
6.10.3.7 operator*=() . . . 78
6.10.3.8 operator+() . . . 78
6.10.3.9 operator+=() . . . 78
6.10.3.10 operator-() . . . 79
6.10.3.11 operator-=() . . . 79
6.10.3.12 operator/() . . . 79
6.10.3.13 operator/=() . . . 80
6.10.3.14 operator=() . . . 80
6.10.3.15 operator==() . . . 80
6.10.4 Member Data Documentation . . . 81
6.10.4.1 x . . . 81
6.10.4.2 y . . . 81
6.11 math::Vector3 Class Reference . . . 81
6.11.1 Detailed Description . . . 82
6.11.2 Constructor & Destructor Documentation . . . 82
6.11.2.1 Vector3() [1/2] . . . 82
6.11.2.2 Vector3() [2/2] . . . 82
6.11.2.3 ~Vector3() . . . 83
6.11.3 Member Function Documentation . . . 83
6.11.3.1 cross() . . . 83
6.11.3.2 dot() . . . 83

6.11.3.3 magnitude() . . . 84
6.11.3.4 magnitudeSquared() . . . 84
6.11.3.5 normal() . . . 84
6.11.3.6 normalise() . . . 84
6.11.3.7 operator∗() . . . 84
6.11.3.8 operator∗=() . . . 85
6.11.3.9 operator+() . . . 85
6.11.3.10 operator+=() . . . 85
6.11.3.11 operator-() . . . 86
6.11.3.12 operator-=() . . . 86
6.11.3.13 operator/() . . . 86
6.11.3.14 operator/=() . . . 87
6.11.3.15 operator=() . . . 87
6.11.3.16 operator==() . . . 87
6.11.4 Member Data Documentation . . . 88
6.11.4.1 x . . . 88
6.11.4.2 y . . . 88
6.11.4.3 z . . . 88
6.12 VkDescriptions Struct Reference . . . 88
6.12.1 Detailed Description . . . 95
6.12.2 Member Data Documentation . . . 96
6.12.2.1 bufferImageGranularity . . . 96
6.12.2.2 discreteQueuePriorities . . . 96
6.12.2.3 framebufferColorSampleCounts . . . 96
6.12.2.4 framebufferDepthSampleCounts . . . 96
6.12.2.5 framebufferNoAttachmentsSampleCounts . . . 96
6.12.2.6 framebufferStencilSampleCounts . . . 97
6.12.2.7 lineWidthGranularity . . . 97
6.12.2.8 lineWidthRangeX . . . 97
6.12.2.9 lineWidthRangeY . . . 97

6.12.2.10 maxBoundDescriptorSets . . . 97
6.12.2.11 maxClipDistances . . . 97
6.12.2.12 maxColorAttachments . . . 98
6.12.2.13 maxCombinedClipAndCullDistances . . . 98
6.12.2.14 maxComputeSharedMemorySize . . . 98
6.12.2.15 maxComputeWorkGroupCountX . . . 98
6.12.2.16 maxComputeWorkGroupCountY . . . 98
6.12.2.17 maxComputeWorkGroupCountZ . . . 99
6.12.2.18 maxComputeWorkGroupInvocations . . . 99
6.12.2.19 maxComputeWorkGroupSizeX . . . 99
6.12.2.20 maxComputeWorkGroupSizeY . . . 99
6.12.2.21 maxComputeWorkGroupSizeZ . . . 99
6.12.2.22 maxCullDistances . . . 100
6.12.2.23 maxDescriptorSetInputAttachments . . . 100
6.12.2.24 maxDescriptorSetSampledImages . . . 100
6.12.2.25 maxDescriptorSetSamplers . . . 100
6.12.2.26 maxDescriptorSetStorageBuffers . . . 100
6.12.2.27 maxDescriptorSetStorageBuffersDynamic . . . 101
6.12.2.28 maxDescriptorSetStorageImages . . . 101
6.12.2.29 maxDescriptorSetUniformBuffers . . . 101
6.12.2.30 maxDescriptorSetUniformBuffersDynamic . . . 101
6.12.2.31 maxDrawIndexedIndexValue . . . 101
6.12.2.32 maxDrawIndirectCount . . . 102
6.12.2.33 maxFragmentCombinedOutputResources . . . 102
6.12.2.34 maxFragmentDualSrcAttachments . . . 102
6.12.2.35 maxFragmentInputComponents . . . 102
6.12.2.36 maxFragmentOutputAttachments . . . 102
6.12.2.37 maxFramebufferHeight . . . 102
6.12.2.38 maxFramebufferLayers . . . 103
6.12.2.39 maxFramebufferWidth . . . 103

6.12.2.40 maxGeometryInputComponents . . . 103
6.12.2.41 maxGeometryOutputComponents . . . 103
6.12.2.42 maxGeometryOutputVertices . . . 103
6.12.2.43 maxGeometryShaderInvocations . . . 103
6.12.2.44 maxGeometryTotalOutputComponents . . . 104
6.12.2.45 maxImageArrayLayers . . . 104
6.12.2.46 maxImageDimension1D . . . 104
6.12.2.47 maxImageDimension2D . . . 104
6.12.2.48 maxImageDimension3D . . . 104
6.12.2.49 maxImageDimensionCube . . . 104
6.12.2.50 maxInterpolationOffset . . . 105
6.12.2.51 maxMemoryAllocationCount . . . 105
6.12.2.52 maxPerStageDescriptorInputAttachments . . . 105
6.12.2.53 maxPerStageDescriptorSampledImages . . . 105
6.12.2.54 maxPerStageDescriptorSamplers . . . 105
6.12.2.55 maxPerStageDescriptorStorageBuffers . . . 106
6.12.2.56 maxPerStageDescriptorStorageImages . . . 106
6.12.2.57 maxPerStageDescriptorUniformBuffers . . . 106
6.12.2.58 maxPerStageResources . . . 106
6.12.2.59 maxPushConstantsSize . . . 107
6.12.2.60 maxSampleMaskWords . . . 107
6.12.2.61 maxSamplerAllocationCount . . . 107
6.12.2.62 maxSamplerAnisotropy . . . 107
6.12.2.63 maxSamplerLodBias . . . 107
6.12.2.64 maxStorageBufferRange . . . 107
6.12.2.65 maxTessellationControlPerPatchOutputComponents . . . 108
6.12.2.66 maxTessellationControlPerVertexInputComponents . . . 108
6.12.2.67 maxTessellationControlPerVertexOutputComponents . . . 108
6.12.2.68 maxTessellationControlTotalOutputComponents . . . 108
6.12.2.69 maxTessellationEvaluationInputComponents . . . 108

6.12.2.70 maxTessellationEvaluationOutputComponents . . . 108
6.12.2.71 maxTessellationGenerationLevel . . . 109
6.12.2.72 maxTessellationPatchSize . . . 109
6.12.2.73 maxTexelBufferElements . . . 109
6.12.2.74 maxTexelGatherOffset . . . 109
6.12.2.75 maxTexelOffset . . . 109
6.12.2.76 maxUniformBufferRange . . . 109
6.12.2.77 maxVertexInputAttributeOffset . . . 110
6.12.2.78 maxVertexInputAttributes . . . 110
6.12.2.79 maxVertexInputBindings . . . 110
6.12.2.80 maxVertexInputBindingStride . . . 110
6.12.2.81 maxVertexOutputComponents . . . 110
6.12.2.82 maxViewportDimensionsX . . . 111
6.12.2.83 maxViewportDimensionsY . . . 111
6.12.2.84 maxViewports . . . 111
6.12.2.85 minInterpolationOffset . . . 111
6.12.2.86 minMemoryMapAlignment . . . 111
6.12.2.87 minStorageBufferOffsetAlignment . . . 111
6.12.2.88 minTexelBufferOffsetAlignment . . . 112
6.12.2.89 minTexelGatherOffset . . . 112
6.12.2.90 minTexelOffset . . . 112
6.12.2.91 minUniformBufferOffsetAlignment . . . 112
6.12.2.92 mipmapPrecisionBits . . . 112
6.12.2.93 nonCoherentAtomSize . . . 112
6.12.2.94 optimalBufferCopyOffsetAlignment . . . 113
6.12.2.95 optimalBufferCopyRowPitchAlignment . . . 113
6.12.2.96 pointSizeGranularity . . . 113
6.12.2.97 pointSizeRangeX . . . 113
6.12.2.98 pointSizeRangeY . . . 113
6.12.2.99 sampledImageColorSampleCounts . . . 113

6.12.2.100 sampledImageDepthSampleCounts . . . 114
6.12.2.101 sampledImageIntegerSampleCounts . . . 114
6.12.2.102 sampledImageStencilSampleCounts . . . 114
6.12.2.103 sparseAddressSpaceSize . . . 114
6.12.2.104 standardSampleLocations . . . 114
6.12.2.105 storageImageSampleCounts . . . 114
6.12.2.106 strictLines . . . 115
6.12.2.107 subPixelInterpolationOffsetBits . . . 115
6.12.2.108 subPixelPrecisionBits . . . 115
6.12.2.109 subTexelPrecisionBits . . . 115
6.12.2.110 timestampComputeAndGraphics . . . 115
6.12.2.111 timestampPeriod . . . 115
6.12.2.112 viewportBoundsRangeX . . . 116
6.12.2.113 viewportBoundsRangeY . . . 116
6.12.2.114 viewportSubPixelBits . . . 116
6.13 VulkanWindow Class Reference . . . 116
6.13.1 Detailed Description . . . 117
6.13.2 Constructor & Destructor Documentation . . . 117
6.13.2.1 VulkanWindow() . . . 117
6.13.3 Member Function Documentation . . . 117
6.13.3.1 createRenderer() . . . 117

7 File Documentation **119**

7.1 libs/opengl-engine/libs/math-library/src/MathLibrary.hpp File Reference . . . 119
7.2 libs/opengl-engine/libs/math-library/src/Vector2.hpp File Reference . . . 119
7.3 libs/opengl-engine/libs/math-library/src/Vector3.hpp File Reference . . . 119
7.4 libs/opengl-engine/src/getgpuinfo.cpp File Reference . . . 120
7.5 libs/opengl-engine/src/glwidget.cpp File Reference . . . 120
7.5.1 Macro Definition Documentation . . . 120
7.5.1.1 QT_APPLICATION . . . 120
7.6 libs/opengl-engine/src/glwidget.h File Reference . . . 120
7.7 libs/opengl-engine/src/glwindow.cpp File Reference . . . 121
7.8 libs/opengl-engine/src/glwindow.h File Reference . . . 121
7.9 libs/opengl-engine/src/model.hpp File Reference . . . 121
7.10 libs/vulkan-engine/src/mesh.cpp File Reference . . . 121
7.11 libs/vulkan-engine/src/mesh.h File Reference . . . 122
7.12 libs/vulkan-engine/src/renderer.cpp File Reference . . . 122
7.13 libs/vulkan-engine/src/renderer.h File Reference . . . 122
7.14 libs/vulkan-engine/src/setvkinfo.cpp File Reference . . . 122
7.15 libs/vulkan-engine/src/shader.cpp File Reference . . . 122
7.16 libs/vulkan-engine/src/shader.h File Reference . . . 123
7.17 libs/vulkan-engine/src/vulkanwindow.cpp File Reference . . . 123
7.18 libs/vulkan-engine/src/vulkanwindow.h File Reference . . . 123
7.19 src/gldescriptions.hpp File Reference . . . 123
7.20 src/main.cpp File Reference . . . 123
7.20.1 Function Documentation . . . 124
7.20.1.1 main() . . . 124
7.21 libs/opengl-engine/src/main.cpp File Reference . . . 124
7.21.1 Function Documentation . . . 124
7.21.1.1 main() . . . 124
7.22 libs/vulkan-engine/src/main.cpp File Reference . . . 124
7.22.1 Function Documentation . . . 125
7.22.1.1 main() . . . 125
7.23 src/mainwindow.cpp File Reference . . . 125
7.24 src/mainwindow.h File Reference . . . 125
7.25 src/runners.cpp File Reference . . . 125
7.26 src/updateGLresults.cpp File Reference . . . 126
7.26.1 Macro Definition Documentation . . . 126
7.26.1.1 DISPLAY . . . 126
7.27 src/updateVKresults.cpp File Reference . . . 126
7.27.1 Macro Definition Documentation . . . 126
7.27.1.1 DISPLAY . . . 126
7.28 src/vkdescriptions.hpp File Reference . . . 126

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

gpu . . . . . ??
math 9
obj . . . . . ??
Ui 9

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chart . . . 11
obj::Face . . . ??
GLDescriptions . . . 12
gpu::GLInfo . . . ??
Mesh . . . 66
Model . . . 68
obj::OBJ . . . ??
gpu::Profiler . . . ??
QMainWindow
    GLWindow . . . 63
    MainWindow . . . 65
QOpenGLFunctions
    GLWidget . . . 60
QOpenGLWidget
    GLWidget . . . 60
QVulkanWindow
    VulkanWindow . . . 116
QVulkanWindowRenderer
    Renderer . . . 71
Shader . . . 73
math::Vector2 . . . 75
math::Vector3 . . . 81
VkDescriptions . . . 88
gpu::VkInfo . . . ??

Chapter 3

Class Index

3.1 Class List

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

Chart
: Struct used to represent a chart. Contains QChartView, QChart, QValueAxis and QLineSeries 11

obj::Face
: Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index . . . **??**

GLDescriptions
: OpenGL variables descriptions. Structure containing the descriptions of the OpenGL variables 12

gpu::GLInfo
: GLInfo struct used to store the GPU and OpenGL data. Structure containing the OpenGL information . . . **??**

GLWidget
: Class for gl widget. OpenGL Widget class based on QGLWidget and QOpenGLFunctions to implement the OpenGL API . . . 60

GLWindow
: Class for gl window. OpenGL window based on QMainWindow widget . . . 63

MainWindow
: Class for main window. Main Window based on QMainWindow Widget. Contains the profilers and the charts . . . 65

Mesh
: Class for mesh. Loads the 3D model (mesh). Keeps track of vertex count and saves data in a QByteArray . . . 66

Model
: Class for model. 3D model loaded as an OBJ. Has a reference to the OBJ instance, the buffers (vertex and UVs) buffers. Contains model array object, shader program and texture . . . 68

obj::OBJ
: Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index . . . **??**

gpu::Profiler
: Class for profiler. Saves the GPU information: vendor, renderer. OpenGL version, shading version, extensions. Vulkan device name, API version, driver version, vendor ID, device ID, extensions. Std::vector stores Profiling data (polycount per time) . . . **??**

Renderer
: Class for renderer. Renderer implementation based on the QVulkanWindowRenderer widget . 71

Shader
: Class for shader. Loads a shader for file so that it can be used by the Vulkan Engine . . . 73

math::Vector2
2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.↩ normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare() 75
math::Vector3
Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.↩ normalise(), Vec.magnitude(), Vec.magnitudeSquare() 81
VkDescriptions
Vulkan variables descriptions. Structure containing the descriptions of the Vulkan variables . . 88
gpu::VkInfo
VkInfo struct used to store the GPU and OpenGL data. Structure containing the Vulkan information . . . . . **??**
VulkanWindow
Class for vulkan window. Vulkan Window class based on QVulkanWindow widget 116

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

libs/opengl-engine/libs/math-library/src/MathLibrary.hpp . . . 119
libs/opengl-engine/libs/math-library/src/Vector2.hpp . . . 119
libs/opengl-engine/libs/math-library/src/Vector3.hpp . . . 119
libs/opengl-engine/libs/obj-loader/src/Face.hpp . . . ??
libs/opengl-engine/libs/obj-loader/src/OBJ.cpp . . . ??
libs/opengl-engine/libs/obj-loader/src/OBJ.h . . . ??
libs/opengl-engine/libs/profiler/src/GLInfo.hpp . . . ??
libs/opengl-engine/libs/profiler/src/Profiler.cpp . . . ??
libs/opengl-engine/libs/profiler/src/Profiler.h . . . ??
libs/opengl-engine/libs/profiler/src/VkInfo.hpp . . . ??
libs/opengl-engine/libs/triangulate/src/Main.cpp . . . ??
libs/opengl-engine/libs/triangulate/src/Triangulate.hpp . . . ??
libs/opengl-engine/src/getgpuinfo.cpp . . . 120
libs/opengl-engine/src/glwidget.cpp . . . 120
libs/opengl-engine/src/glwidget.h . . . 120
libs/opengl-engine/src/glwindow.cpp . . . 121
libs/opengl-engine/src/glwindow.h . . . 121
libs/opengl-engine/src/main.cpp . . . 124
libs/opengl-engine/src/model.hpp . . . 121
libs/vulkan-engine/src/main.cpp . . . 124
libs/vulkan-engine/src/mesh.cpp . . . 121
libs/vulkan-engine/src/mesh.h . . . 122
libs/vulkan-engine/src/renderer.cpp . . . 122
libs/vulkan-engine/src/renderer.h . . . 122
libs/vulkan-engine/src/setvkinfo.cpp . . . 122
libs/vulkan-engine/src/shader.cpp . . . 122
libs/vulkan-engine/src/shader.h . . . 123
libs/vulkan-engine/src/vulkanwindow.cpp . . . 123
libs/vulkan-engine/src/vulkanwindow.h . . . 123
src/gldescriptions.hpp . . . 123
src/main.cpp . . . 123
src/mainwindow.cpp . . . 125
src/mainwindow.h . . . 125
src/runners.cpp . . . 125
src/updateGLresults.cpp . . . 126
src/updateVKresults.cpp . . . 126
src/vkdescriptions.hpp . . . 126

Chapter 5

Namespace Documentation

5.1 gpu Namespace Reference

Classes

- struct GLInfo

  GLInfo struct used to store the GPU and OpenGL data. Structure containing the OpenGL information.
- class Profiler

  Class for profiler. Saves the GPU information: vendor, renderer. OpenGL version, shading version, extensions. Vulkan device name, API version, driver version, vendor ID, device ID, extensions. Std::vector stores Profiling data (polycount per time).
- struct VkInfo

  VkInfo struct used to store the GPU and OpenGL data. Structure containing the Vulkan information.

Enumerations

- enum Count {
  VK_SAMPLE_COUNT_1_BIT = 0x00000001, VK_SAMPLE_COUNT_2_BIT = 0x00000002, VK_SAMPLE_COUNT_4_BIT = 0x00000004, VK_SAMPLE_COUNT_8_BIT = 0x00000008,
  VK_SAMPLE_COUNT_16_BIT = 0x00000010, VK_SAMPLE_COUNT_32_BIT = 0x00000020, VK_SAMPLE_COUNT_64_BIT = 0x00000040, VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM = 0x7FFFFFFF }

5.1.1 Enumeration Type Documentation

5.1.1.1 Count

`enum gpu::Count`

Sample count.

Enumerator

VK_SAMPLE_COUNT_1_BIT	
VK_SAMPLE_COUNT_2_BIT	
VK_SAMPLE_COUNT_4_BIT	
VK_SAMPLE_COUNT_8_BIT	
VK_SAMPLE_COUNT_16_BIT	
VK_SAMPLE_COUNT_32_BIT	
VK_SAMPLE_COUNT_64_BIT	
VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM	

5.2 math Namespace Reference

Classes

- class Vector2

  2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()
- class Vector3

  Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()

5.3 obj Namespace Reference

Classes

- class Face

  Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index.
- class OBJ

  Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index.

Functions

- void triangulate (OBJ &model)

  Triangulate and subdivide an OBJ file. Uses a 3D model loaded and loops through the faces ABC. For each face, computes 3 new points (D, E and F). D = (A + B)/2, E = (B + C)/2 and F = (C + A)/2. Adds D, E and F to the model and retrieved their indices. Delete the current face and create 4 faces: ADF, DBE, DEF and ECF.

5.3.1 Function Documentation

5.3.1.1 triangulate()

```
void obj::triangulate (
            OBJ & model )
```

Triangulate and subdivide an OBJ file. Uses a 3D model loaded and loops through the faces ABC. For each face, computes 3 new points (D, E and F). D = (A + B)/2, E = (B + C)/2 and F = (C + A)/2. Adds D, E and F to the model and retrieved their indices. Delete the current face and create 4 faces: ADF, DBE, DEF and ECF.

Parameters

model	The model to be subdivided

5.4 Ui Namespace Reference

5.4.1 Detailed Description

UI namespace.

Chapter 6

Class Documentation

6.1 Chart Struct Reference

Struct used to represent a chart. Contains QChartView, QChart, QValueAxis and QLineSeries.

`#include <mainwindow.h>`

Public Attributes

- QChartView * chartView
- QChart * chart
- QValueAxis * axisX
- QValueAxis * axisY
- QLineSeries * series

6.1.1 Detailed Description

Struct used to represent a chart. Contains QChartView, QChart, QValueAxis and QLineSeries.

6.1.2 Member Data Documentation

6.1.2.1 axisX

`QValueAxis* Chart::axisX`

6.1.2.2 axisY

```
QValueAxis * Chart::axisY
```

6.1.2.3 chart

```
QChart* Chart::chart
```

6.1.2.4 chartView

```
QChartView* Chart::chartView
```

6.1.2.5 series

```
QLineSeries* Chart::series
```

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

- src/mainwindow.h

6.2 obj::Face Class Reference

Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index.

```
#include <Face.hpp>
```

Public Member Functions

- Face ()

  Constructs the object. Empty constructor sets vertices, texture coordinates and normals indices to 0.
- Face (unsigned v0, unsigned tc0, unsigned n0, unsigned v1, unsigned tc1, unsigned n1, unsigned v2, unsigned tc2, unsigned n2)

  Constructs the object.

Public Attributes

- unsigned v0
- unsigned v1
- unsigned v2
- unsigned tc0
- unsigned tc1
- unsigned tc2
- unsigned n0
- unsigned n1
- unsigned n2

6.2.1 Detailed Description

Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 Face() [1/2]

```
obj::Face::Face ( )  [inline]
```

Constructs the object. Empty constructor sets vertices, texture coordinates and normals indices to 0.

6.2.2.2 Face() [2/2]

```
obj::Face::Face (
            unsigned v0,
            unsigned tc0,
            unsigned n0,
            unsigned v1,
            unsigned tc1,
            unsigned n1,
            unsigned v2,
            unsigned tc2,
            unsigned n2 )  [inline]
```

Constructs the object.

Parameters

in	*v0*	First vertex index
in	*tc0*	First vertex texture coordinate index
in	*n0*	First vertex normal index
in	*v1*	First vertex index
in	*tc1*	First vertex texture coordinate index
in	*n1*	First vertex normal index
in	*v2*	First vertex index
in	*tc2*	First vertex texture coordinate index
in	*n2*	First vertex normal index

6.2.3 Member Data Documentation

6.2.3.1 n0

```
unsigned obj::Face::n0
```

6.2.3.2 n1

```
unsigned obj::Face::n1
```

6.2.3.3 n2

```
unsigned obj::Face::n2
```

6.2.3.4 tc0

```
unsigned obj::Face::tc0
```

6.2.3.5 tc1

```
unsigned obj::Face::tc1
```

6.2.3.6 tc2

```
unsigned obj::Face::tc2
```

6.2.3.7 v0

```
unsigned obj::Face::v0
```

6.2.3.8 v1

```
unsigned obj::Face::v1
```

6.2.3.9 v2

```
unsigned obj::Face::v2
```

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

- libs/opengl-engine/libs/obj-loader/src/Face.hpp

6.3 GLDescriptions Struct Reference

OpenGL variables descriptions. Structure containing the descriptions of the OpenGL variables.

```
#include <gldescriptions.hpp>
```

Public Attributes

- QString gl_active_texture = "Data returns a single value indicating the active multitexture unit. The initial value is GL_TEXTURE0. See glActiveTexture."
- QString gl_aliased_line_width_range = "Data returns a pair of values indicating the range of widths supported for aliased lines. See glLineWidth."
- QString gl_array_buffer_binding = "Data returns a single value, the name of the buffer object currently bound to the target GL_ARRAY_BUFFER. If no buffer object is bound to this target, 0 is returned. The initial value is 0. See glBindBuffer."
- QString gl_blend = "Data returns a single boolean value indicating whether blending is enabled. The initial value is GL_FALSE. See glBlendFunc."
- QString gl_blend_color = "Data returns four values, the red, green, blue, and alpha values which are the components of the blend color. See glBlendColor."
- QString gl_blend_dst_alpha = "Data returns one value, the symbolic constant identifying the alpha destination blend function. The initial value is GL_ZERO. See glBlendFunc and glBlendFuncSeparate."
- QString gl_blend_dst_rgb = "Data returns one value, the symbolic constant identifying the RGB destination blend function. The initial value is GL_ZERO. See glBlendFunc and glBlendFuncSeparate."
- QString gl_blend_equation_rgb = "Data returns one value, a symbolic constant indicating whether the RGB blend equation is GL_FUNC_ADD, GL_FUNC_SUBTRACT, GL_FUNC_REVERSE_SUBTRACT, GL_MIN or GL_MAX. See glBlendEquationSeparate."
- QString gl_blend_equation_alpha = "Data returns one value, a symbolic constant indicating whether the Alpha blend equation is GL_FUNC_ADD, GL_FUNC_SUBTRACT, GL_FUNC_REVERSE_SUBTRACT, GL_MIN or GL_MAX. See glBlendEquationSeparate."
- QString gl_blend_src_alpha = "Data returns one value, the symbolic constant identifying the alpha source blend function. The initial value is GL_ONE. See glBlendFunc and glBlendFuncSeparate."
- QString gl_blend_src_rgb = "Data returns one value, the symbolic constant identifying the RGB source blend function. The initial value is GL_ONE. See glBlendFunc and glBlendFuncSeparate."
- QString gl_color_clear_value = "Data returns four values: the red, green, blue, and alpha values used to clear the color buffers. Integer values, if requested, are linearly mapped from the internal floating-point representation such that 1.0 returns the most positive representable integer value, and 1.0 returns the most negative representable integer value. The initial value is (0, 0, 0, 0). See glClearColor."

- QString gl_color_logic_op = "Data returns a single boolean value indicating whether a fragment's RGBA color values are merged into the framebuffer using a logical operation. The initial value is GL_FALSE. See glLogicOp."
- QString gl_color_writemask = "Data returns four boolean values: the red, green, blue, and alpha write enables for the color buffers. The initial value is (GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE). See glColorMask."
- QString gl_compressed_texture_formats = "Data returns a list of symbolic constants of length GL_NUM↩ _COMPRESSED_TEXTURE_FORMATS indicating which compressed texture formats are available. See glCompressedTexImage2D."
- QString gl_max_compute_shader_storage_blocks = "Data returns one value, the maximum number of active shader storage blocks that may be accessed by a compute shader."
- QString gl_max_combined_shader_storage_blocks = "Data returns one value, the maximum total number of active shader storage blocks that may be accessed by all active shaders."
- QString gl_max_compute_uniform_blocks = "Data returns one value, the maximum number of uniform blocks per compute shader. The value must be at least 14. See glUniformBlockBinding."
- QString gl_max_compute_texture_image_units = "Data returns one value, the maximum supported texture image units that can be used to access texture maps from the compute shader. The value may be at least 16. See glActiveTexture."
- QString gl_max_compute_uniform_components = "Data returns one value, the maximum number of individual floating-point, integer, or boolean values that can be held in uniform variable storage for a compute shader. The value must be at least 1024. See glUniform."
- QString gl_max_compute_atomic_counters = "Data returns a single value, the maximum number of atomic counters available to compute shaders."
- QString gl_max_compute_atomic_counter_buffers = "Data returns a single value, the maximum number of atomic counter buffers that may be accessed by a compute shader."
- QString gl_max_combined_compute_uniform_components = "Data returns one value, the number of words for compute shader uniform variables in all uniform blocks (including default). The value must be at least 1. See glUniform."
- QString gl_max_compute_work_group_count = "Accepted by the indexed versions of glGet. data the maximum number of work groups that may be dispatched to a compute shader. Indices 0, 1, and 2 correspond to the X, Y and Z dimensions, respectively."
- QString gl_max_compute_work_group_size = "Accepted by the indexed versions of glGet. data the maximum size of a work groups that may be used during compilation of a compute shader. Indices 0, 1, and 2 correspond to the X, Y and Z dimensions, respectively."
- QString gl_dispatch_indirect_buffer_binding = "Data returns a single value, the name of the buffer object currently bound to the target GL_DISPATCH_INDIRECT_BUFFER. If no buffer object is bound to this target, 0 is returned. The initial value is 0. See glBindBuffer."
- QString gl_max_debug_group_stack_depth = "Data returns a single value, the maximum depth of the debug message group stack."
- QString gl_debug_group_stack_depth = "Data returns a single value, the current depth of the debug message group stack."
- QString gl_context_flags = "Data returns one value, the flags with which the context was created (such as debugging functionality)."
- QString gl_cull_face = "Data returns a single boolean value indicating whether polygon culling is enabled. The initial value is GL_FALSE. See glCullFace."
- QString gl_cull_face_mode = "Data returns a single value indicating the mode of polygon culling. The initial value is GL_BACK. See glCullFace."
- QString gl_current_program = "Data returns one value, the name of the program object that is currently active, or 0 if no program object is active. See glUseProgram."
- QString gl_depth_clear_value = "Data returns one value, the value that is used to clear the depth buffer. Integer values, if requested, are linearly mapped from the internal floating-point representation such that 1.0 returns the most positive representable integer value, and 1.0 returns the most negative representable integer value. The initial value is 1. See glClearDepth."
- QString gl_depth_func = "Data returns one value, the symbolic constant that indicates the depth comparison function. The initial value is GL_LESS. See glDepthFunc."
- QString gl_depth_range = "Data returns two values: the near and far mapping limits for the depth buffer. Integer values, if requested, are linearly mapped from the internal floating-point representation such that 1.0 returns the most positive representable integer value, and 1.0 returns the most negative representable integer value. The initial value is (0, 1). See glDepthRange."

- QString gl_depth_test = "Data returns a single boolean value indicating whether depth testing of fragments is enabled. The initial value is GL_FALSE. See glDepthFunc and glDepthRange."
- QString gl_depth_writemask = "Data returns a single boolean value indicating if the depth buffer is enabled for writing. The initial value is GL_TRUE. See glDepthMask."
- QString gl_dither = "Data returns a single boolean value indicating whether dithering of fragment colors and indices is enabled. The initial value is GL_TRUE."
- QString gl_doublebuffer = "Data returns a single boolean value indicating whether double buffering is supported."
- QString gl_draw_buffer = "Data returns one value, a symbolic constant indicating which buffers are being drawn to. See glDrawBuffer. The initial value is GL_BACK if there are back buffers, otherwise it is GL_FR↩ ONT."
- QString gl_draw_framebuffer_binding = "Data returns one value, the name of the framebuffer object currently bound to the GL_DRAW_FRAMEBUFFER target. If the default framebuffer is bound, this value will be zero. The initial value is zero. See glBindFramebuffer."
- QString gl_read_framebuffer_binding = "Data returns one value, the name of the framebuffer object currently bound to the GL_READ_FRAMEBUFFER target. If the default framebuffer is bound, this value will be zero. The initial value is zero. See glBindFramebuffer."
- QString gl_element_array_buffer_binding = "Data returns a single value, the name of the buffer object currently bound to the target GL_ELEMENT_ARRAY_BUFFER. If no buffer object is bound to this target, 0 is returned. The initial value is 0. See glBindBuffer."
- QString gl_fragment_shader_derivative_hint = "Data returns one value, a symbolic constant indicating the mode of the derivative accuracy hint for fragment shaders. The initial value is GL_DONT_CARE. See glHint."
- QString gl_implementation_color_read_format = "Data returns a single GLenum value indicating the implementation's preferred pixel data format. See glReadPixels."
- QString gl_implementation_color_read_type = "Data returns a single GLenum value indicating the implementation's preferred pixel data type. See glReadPixels."
- QString gl_line_smooth = "Data returns a single boolean value indicating whether antialiasing of lines is enabled. The initial value is GL_FALSE. See glLineWidth."
- QString gl_line_smooth_hint = "Data returns one value, a symbolic constant indicating the mode of the line antialiasing hint. The initial value is GL_DONT_CARE. See glHint."
- QString gl_line_width = "Data returns one value, the line width as specified with glLineWidth. The initial value is 1."
- QString gl_layer_provoking_vertex = "Data returns one value, the implementation dependent specifc vertex of a primitive that is used to select the rendering layer. If the value returned is equivalent to GL_PROVOK↩ ING_VERTEX, then the vertex selection follows the convention specified by glProvokingVertex. If the value returned is equivalent to GL_FIRST_VERTEX_CONVENTION, then the selection is always taken from the first vertex in the primitive. If the value returned is equivalent to GL_LAST_VERTEX_CONVENTION, then the selection is always taken from the last vertex in the primitive. If the value returned is equivalent to G↩ L_UNDEFINED_VERTEX, then the selection is not guaranteed to be taken from any specific vertex in the primitive."
- QString gl_logic_op_mode = "Data returns one value, a symbolic constant indicating the selected logic operation mode. The initial value is GL_COPY. See glLogicOp."
- QString gl_major_version = "Data returns one value, the major version number of the OpenGL API supported by the current context."
- QString gl_max_3d_texture_size = "Data returns one value, a rough estimate of the largest 3D texture that the GL can handle. The value must be at least 64. Use GL_PROXY_TEXTURE_3D to determine if a texture is too large. See glTexImage3D."
- QString gl_max_array_texture_layers = "Data returns one value. The value indicates the maximum number of layers allowed in an array texture, and must be at least 256. See glTexImage2D."
- QString gl_max_clip_distances = "Data returns one value, the maximum number of application-defined clipping distances. The value must be at least 8."
- QString gl_max_color_texture_samples = "Data returns one value, the maximum number of samples in a color multisample texture."
- QString gl_max_combined_atomic_counters = "Data returns a single value, the maximum number of atomic counters available to all active shaders."

- QString gl_max_combined_fragment_uniform_components = "Data returns one value, the number of words for fragment shader uniform variables in all uniform blocks (including default). The value must be at least 1. See glUniform."
- QString gl_max_combined_geometry_uniform_components = "Data returns one value, the number of words for geometry shader uniform variables in all uniform blocks (including default). The value must be at least 1. See glUniform."
- QString gl_max_combined_texture_image_units = "Data returns one value, the maximum supported texture image units that can be used to access texture maps from the vertex shader and the fragment processor combined. If both the vertex shader and the fragment processing stage access the same texture image unit, then that counts as using two texture image units against this limit. The value must be at least 48. See glActiveTexture."
- QString gl_max_combined_uniform_blocks = "Data returns one value, the maximum number of uniform blocks per program. The value must be at least 70. See glUniformBlockBinding."
- QString gl_max_combined_vertex_uniform_components = "Data returns one value, the number of words for vertex shader uniform variables in all uniform blocks (including default). The value must be at least 1. See glUniform."
- QString gl_max_cube_map_texture_size = "Data returns one value. The value gives a rough estimate of the largest cube-map texture that the GL can handle. The value must be at least 1024. Use GL_PROXY_TEX↩ TURE_CUBE_MAP to determine if a texture is too large. See glTexImage2D."
- QString gl_max_depth_texture_samples = "Data returns one value, the maximum number of samples in a multisample depth or depth-stencil texture."
- QString gl_max_draw_buffers = "Data returns one value, the maximum number of simultaneous outputs that may be written in a fragment shader. The value must be at least 8. See glDrawBuffers."
- QString gl_max_dual_source_draw_buffers = "Data returns one value, the maximum number of active draw buffers when using dual-source blending. The value must be at least 1. See glBlendFunc and glBlendFunc↩ Separate."
- QString gl_max_elements_indices = "Data returns one value, the recommended maximum number of vertex array indices. See glDrawRangeElements."
- QString gl_max_elements_vertices = "Data returns one value, the recommended maximum number of vertex array vertices. See glDrawRangeElements."
- QString gl_max_fragment_atomic_counters = "Data returns a single value, the maximum number of atomic counters available to fragment shaders."
- QString gl_max_fragment_shader_storage_blocks = "Data returns one value, the maximum number of active shader storage blocks that may be accessed by a fragment shader."
- QString gl_max_fragment_input_components = "Data returns one value, the maximum number of components of the inputs read by the fragment shader, which must be at least 128."
- QString gl_max_fragment_uniform_components = "Data returns one value, the maximum number of individual floating-point, integer, or boolean values that can be held in uniform variable storage for a fragment shader. The value must be at least 1024. See glUniform."
- QString gl_max_fragment_uniform_vectors = "Data returns one value, the maximum number of individual 4-vectors of floating-point, integer, or boolean values that can be held in uniform variable storage for a fragment shader. The value is equal to the value of GL_MAX_FRAGMENT_UNIFORM_COMPONENTS divided by 4 and must be at least 256. See glUniform."
- QString gl_max_fragment_uniform_blocks = "Data returns one value, the maximum number of uniform blocks per fragment shader. The value must be at least 12. See glUniformBlockBinding."
- QString gl_max_framebuffer_width = "Data returns one value, the maximum width for a framebuffer that has no attachments, which must be at least 16384. See glFramebufferParameter."
- QString gl_max_framebuffer_height = "Data returns one value, the maximum height for a framebuffer that has no attachments, which must be at least 16384. See glFramebufferParameter."
- QString gl_max_framebuffer_layers = "Data returns one value, the maximum number of layers for a framebuffer that has no attachments, which must be at least 2048. See glFramebufferParameter."
- QString gl_max_framebuffer_samples = "Data returns one value, the maximum samples in a framebuffer that has no attachments, which must be at least 4. See glFramebufferParameter."
- QString gl_max_geometry_atomic_counters = "Data returns a single value, the maximum number of atomic counters available to geometry shaders."
- QString gl_max_geometry_shader_storage_blocks = "Data returns one value, the maximum number of active shader storage blocks that may be accessed by a geometry shader."

- QString gl_max_geometry_input_components = "Data returns one value, the maximum number of components of inputs read by a geometry shader, which must be at least 64."
- QString gl_max_geometry_output_components = "Data returns one value, the maximum number of components of outputs written by a geometry shader, which must be at least 128."
- QString gl_max_geometry_texture_image_units = "Data returns one value, the maximum supported texture image units that can be used to access texture maps from the geometry shader. The value must be at least 16. See glActiveTexture."
- QString gl_max_geometry_uniform_blocks = "Data returns one value, the maximum number of uniform blocks per geometry shader. The value must be at least 12. See glUniformBlockBinding."
- QString gl_max_geometry_uniform_components = "Data returns one value, the maximum number of individual floating-point, integer, or boolean values that can be held in uniform variable storage for a geometry shader. The value must be at least 1024. See glUniform."
- QString gl_max_integer_samples = "Data returns one value, the maximum number of samples supported in integer format multisample buffers."
- QString gl_min_map_buffer_alignment = "Data returns one value, the minimum alignment in basic machine units of pointers returned fromglMapBuffer and glMapBufferRange. This value must be a power of two and must be at least 64."
- QString gl_max_label_length = "Data returns one value, the maximum length of a label that may be assigned to an object. See glObjectLabel and glObjectPtrLabel."
- QString gl_max_program_texel_offset = "Data returns one value, the maximum texel offset allowed in a texture lookup, which must be at least 7."
- QString gl_min_program_texel_offset = "Data returns one value, the minimum texel offset allowed in a texture lookup, which must be at most -8."
- QString gl_max_rectangle_texture_size = "Data returns one value. The value gives a rough estimate of the largest rectangular texture that the GL can handle. The value must be at least 1024. Use GL_PROXY_TE↩ XTURE_RECTANGLE to determine if a texture is too large. See glTexImage2D."
- QString gl_max_renderbuffer_size = "Data returns one value. The value indicates the maximum supported size for renderbuffers. See glFramebufferRenderbuffer."
- QString gl_max_sample_mask_words = "Data returns one value, the maximum number of sample mask words."
- QString gl_max_server_wait_timeout = "Data returns one value, the maximum glWaitSync timeout interval."
- QString gl_max_shader_storage_buffer_bindings = "Data returns one value, the maximum number of shader storage buffer binding points on the context, which must be at least 8."
- QString gl_max_tess_control_atomic_counters = "Data returns a single value, the maximum number of atomic counters available to tessellation control shaders."
- QString gl_max_tess_evaluation_atomic_counters = "Data returns a single value, the maximum number of atomic counters available to tessellation evaluation shaders."
- QString gl_max_tess_control_shader_storage_blocks = "Data returns one value, the maximum number of active shader storage blocks that may be accessed by a tessellation control shader."
- QString gl_max_tess_evaluation_shader_storage_blocks = "Data returns one value, the maximum number of active shader storage blocks that may be accessed by a tessellation evaluation shader."
- QString gl_max_texture_buffer_size = "Data returns one value. The value gives the maximum number of texels allowed in the texel array of a texture buffer object. Value must be at least 65536."
- QString gl_max_texture_image_units = "Data returns one value, the maximum supported texture image units that can be used to access texture maps from the fragment shader. The value must be at least 16. See glActiveTexture."
- QString gl_max_texture_lod_bias = "Data returns one value, the maximum, absolute value of the texture level-of-detail bias. The value must be at least 2.0."
- QString gl_max_texture_size = "Data returns one value. The value gives a rough estimate of the largest texture that the GL can handle. The value must be at least 1024. Use a proxy texture target such as GL_PR↩ OXY_TEXTURE_1D or GL_PROXY_TEXTURE_2D to determine if a texture is too large. See glTexImage1D and glTexImage2D."
- QString gl_max_uniform_buffer_bindings = "Data returns one value, the maximum number of uniform buffer binding points on the context, which must be at least 36."
- QString gl_max_uniform_block_size = "Data returns one value, the maximum size in basic machine units of a uniform block, which must be at least 16384."

- QString gl_max_uniform_locations = "Data returns one value, the maximum number of explicitly assignable uniform locations, which must be at least 1024."
- QString gl_max_varying_components = "Data returns one value, the number components for varying variables, which must be at least 60."
- QString gl_max_varying_vectors = "Data returns one value, the number 4-vectors for varying variables, which is equal to the value of GL_MAX_VARYING_COMPONENTS and must be at least 15."
- QString gl_max_varying_floats = "Data returns one value, the maximum number of interpolators available for processing varying variables used by vertex and fragment shaders. This value represents the number of individual floating-point values that can be interpolated; varying variables declared as vectors, matrices, and arrays will all consume multiple interpolators. The value must be at least 32."
- QString gl_max_vertex_atomic_counters = "Data returns a single value, the maximum number of atomic counters available to vertex shaders."
- QString gl_max_vertex_attribs = "Data returns one value, the maximum number of 4-component generic vertex attributes accessible to a vertex shader. The value must be at least 16. See glVertexAttrib."
- QString gl_max_vertex_shader_storage_blocks = "Data returns one value, the maximum number of active shader storage blocks that may be accessed by a vertex shader."
- QString gl_max_vertex_texture_image_units = "Data returns one value, the maximum supported texture image units that can be used to access texture maps from the vertex shader. The value may be at least 16. See glActiveTexture."
- QString gl_max_vertex_uniform_components = "Data returns one value, the maximum number of individual floating-point, integer, or boolean values that can be held in uniform variable storage for a vertex shader. The value must be at least 1024. See glUniform."
- QString gl_max_vertex_uniform_vectors = "Data returns one value, the maximum number of 4-vectors that may be held in uniform variable storage for the vertex shader. The value of GL_MAX_VERTEX_UNIFOR↩ M_VECTORS is equal to the value of GL_MAX_VERTEX_UNIFORM_COMPONENTS and must be at least 256."
- QString gl_max_vertex_output_components = "Data returns one value, the maximum number of components of output written by a vertex shader, which must be at least 64."
- QString gl_max_vertex_uniform_blocks = "Data returns one value, the maximum number of uniform blocks per vertex shader. The value must be at least 12. See glUniformBlockBinding."
- QString gl_max_viewport_dims = "Data returns two values: the maximum supported width and height of the viewport. These must be at least as large as the visible dimensions of the display being rendered to. See glViewport."
- QString gl_max_viewports = "Data returns one value, the maximum number of simultaneous viewports that are supported. The value must be at least 16. See glViewportIndexed."
- QString gl_minor_version = "Data returns one value, the minor version number of the OpenGL API supported by the current context."
- QString gl_num_compressed_texture_formats = "Data returns a single integer value indicating the number of available compressed texture formats. The minimum value is 4. See glCompressedTexImage2D."
- QString gl_num_extensions = "Data returns one value, the number of extensions supported by the GL implementation for the current context. See glGetString."
- QString gl_num_program_binary_formats = "Data returns one value, the number of program binary formats supported by the implementation."
- QString gl_num_shader_binary_formats = "Data returns one value, the number of binary shader formats supported by the implementation. If this value is greater than zero, then the implementation supports loading binary shaders. If it is zero, then the loading of binary shaders by the implementation is not supported."
- QString gl_pack_alignment = "Data returns one value, the byte alignment used for writing pixel data to memory. The initial value is 4. See glPixelStore."
- QString gl_pack_image_height = "Data returns one value, the image height used for writing pixel data to memory. The initial value is 0. See glPixelStore."
- QString gl_pack_lsb_first = "Data returns a single boolean value indicating whether single-bit pixels being written to memory are written first to the least significant bit of each unsigned byte. The initial value is GL_↩ FALSE. See glPixelStore."
- QString gl_pack_row_length = "Data returns one value, the row length used for writing pixel data to memory. The initial value is 0. See glPixelStore."
- QString gl_pack_skip_images = "Data returns one value, the number of pixel images skipped before the first pixel is written into memory. The initial value is 0. See glPixelStore."

- QString gl_pack_skip_pixels = "Data returns one value, the number of pixel locations skipped before the first pixel is written into memory. The initial value is 0. See glPixelStore."
- QString gl_pack_skip_rows = "Data returns one value, the number of rows of pixel locations skipped before the first pixel is written into memory. The initial value is 0. See glPixelStore."
- QString gl_pack_swap_bytes = "Data returns a single boolean value indicating whether the bytes of two-byte and four-byte pixel indices and components are swapped before being written to memory. The initial value is GL_FALSE. See glPixelStore."
- QString gl_pixel_pack_buffer_binding = "Data returns a single value, the name of the buffer object currently bound to the target GL_PIXEL_PACK_BUFFER. If no buffer object is bound to this target, 0 is returned. The initial value is 0. See glBindBuffer."
- QString gl_pixel_unpack_buffer_binding = "Data returns a single value, the name of the buffer object currently bound to the target GL_PIXEL_UNPACK_BUFFER. If no buffer object is bound to this target, 0 is returned. The initial value is 0. See glBindBuffer."
- QString gl_point_fade_threshold_size = "Data returns one value, the point size threshold for determining the point size. See glPointParameter."
- QString gl_primitive_restart_index = "Data returns one value, the current primitive restart index. The initial value is 0. See glPrimitiveRestartIndex."
- QString gl_program_binary_formats = "Data an array of GL_NUM_PROGRAM_BINARY_FORMATS values, indicating the proram binary formats supported by the implementation."
- QString gl_program_pipeline_binding = "Data a single value, the name of the currently bound program pipeline object, or zero if no program pipeline object is bound. See glBindProgramPipeline."
- QString gl_program_point_size = "Data returns a single boolean value indicating whether vertex program point size mode is enabled. If enabled, then the point size is taken from the shader built-in gl_PointSize. If disabled, then the point size is taken from the point state as specified by glPointSize. The initial value is GL_FALSE."
- QString gl_provoking_vertex = "Data returns one value, the currently selected provoking vertex convention. The initial value is GL_LAST_VERTEX_CONVENTION. See glProvokingVertex."
- QString gl_point_size = "Data returns one value, the point size as specified by glPointSize. The initial value is 1."
- QString gl_point_size_granularity = "Data returns one value, the size difference between adjacent supported sizes for antialiased points. See glPointSize."
- QString gl_point_size_range = "Data returns two values: the smallest and largest supported sizes for antialiased points. The smallest size must be at most 1, and the largest size must be at least 1. See glPoint↩ Size."
- QString gl_polygon_offset_factor = "Data returns one value, the scaling factor used to determine the variable offset that is added to the depth value of each fragment generated when a polygon is rasterized. The initial value is 0. See glPolygonOffset."
- QString gl_polygon_offset_units = "Data returns one value. This value is multiplied by an implementation-specific value and then added to the depth value of each fragment generated when a polygon is rasterized. The initial value is 0. See glPolygonOffset."
- QString gl_polygon_offset_fill = "Data returns a single boolean value indicating whether polygon offset is enabled for polygons in fill mode. The initial value is GL_FALSE. See glPolygonOffset."
- QString gl_polygon_offset_line = "Data returns a single boolean value indicating whether polygon offset is enabled for polygons in line mode. The initial value is GL_FALSE. See glPolygonOffset."
- QString gl_polygon_offset_point = "Data returns a single boolean value indicating whether polygon offset is enabled for polygons in point mode. The initial value is GL_FALSE. See glPolygonOffset."
- QString gl_polygon_smooth = "Data returns a single boolean value indicating whether antialiasing of polygons is enabled. The initial value is GL_FALSE. See glPolygonMode."
- QString gl_polygon_smooth_hint = "Data returns one value, a symbolic constant indicating the mode of the polygon antialiasing hint. The initial value is GL_DONT_CARE. See glHint."
- QString gl_read_buffer = "Data returns one value, a symbolic constant indicating which color buffer is selected for reading. The initial value is GL_BACK if there is a back buffer, otherwise it is GL_FRONT. See glRead↩ Pixels."
- QString gl_renderbuffer_binding = "Data returns a single value, the name of the renderbuffer object currently bound to the target GL_RENDERBUFFER. If no renderbuffer object is bound to this target, 0 is returned. The initial value is 0. See glBindRenderbuffer."

- QString gl_sample_buffers = "Data returns a single integer value indicating the number of sample buffers associated with the framebuffer. See glSampleCoverage."
- QString gl_sample_coverage_value = "Data returns a single positive floating-point value indicating the current sample coverage value. See glSampleCoverage."
- QString gl_sample_coverage_invert = "Data returns a single boolean value indicating if the temporary coverage value should be inverted. See glSampleCoverage."
- QString gl_sampler_binding = "Data returns a single value, the name of the sampler object currently bound to the active texture unit. The initial value is 0. See glBindSampler."
- QString gl_samples = "Data returns a single integer value indicating the coverage mask size. See glSample↩ Coverage."
- QString gl_scissor_box = "Data returns four values: the x and y window coordinates of the scissor box, followed by its width and height. Initially the x and y window coordinates are both 0 and the width and height are set to the size of the window. See glScissor."
- QString gl_scissor_test = "Data returns a single boolean value indicating whether scissoring is enabled. The initial value is GL_FALSE. See glScissor."
- QString gl_shader_compiler = "Data returns a single boolean value indicating whether an online shader compiler is present in the implementation. All desktop OpenGL implementations must support online shader compilations, and therefore the value of GL_SHADER_COMPILER will always be GL_TRUE."
- QString gl_shader_storage_buffer_binding = "When used with non-indexed variants of glGet (such as glGet↩ Integerv), data returns a single value, the name of the buffer object currently bound to the target GL_SHA↩ DER_STORAGE_BUFFER. If no buffer object is bound to this target, 0 is returned. When used with indexed variants of glGet (such as glGetIntegeri_v), data returns a single value, the name of the buffer object bound to the indexed shader storage buffer binding points. The initial value is 0 for all targets. See glBindBuffer, glBindBufferBase, and glBindBufferRange."
- QString gl_shader_storage_buffer_offset_alignment = "Data returns a single value, the minimum required alignment for shader storage buffer sizes and offset. The initial value is 1. See glShaderStorageBlock↩ Binding."
- QString gl_shader_storage_buffer_start = "When used with indexed variants of glGet (such as glGet↩ Integer64i_v), data returns a single value, the start offset of the binding range for each indexed shader storage buffer binding. The initial value is 0 for all bindings. See glBindBufferRange."
- QString gl_shader_storage_buffer_size = "When used with indexed variants of glGet (such as glGet↩ Integer64i_v), data returns a single value, the size of the binding range for each indexed shader storage buffer binding. The initial value is 0 for all bindings. See glBindBufferRange."
- QString gl_smooth_line_width_range = "Data returns a pair of values indicating the range of widths supported for smooth (antialiased) lines. See glLineWidth."
- QString gl_smooth_line_width_granularity = "Data returns a single value indicating the level of quantization applied to smooth line width parameters."
- QString gl_stencil_back_fail = "Data returns one value, a symbolic constant indicating what action is taken for back-facing polygons when the stencil test fails. The initial value is GL_KEEP. See glStencilOpSeparate."
- QString gl_stencil_back_func = "Data returns one value, a symbolic constant indicating what function is used for back-facing polygons to compare the stencil reference value with the stencil buffer value. The initial value is GL_ALWAYS. See glStencilFuncSeparate."
- QString gl_stencil_back_pass_depth_fail = "Data returns one value, a symbolic constant indicating what action is taken for back-facing polygons when the stencil test passes, but the depth test fails. The initial value is GL_KEEP. See glStencilOpSeparate."
- QString gl_stencil_back_pass_depth_pass = "Data returns one value, a symbolic constant indicating what action is taken for back-facing polygons when the stencil test passes and the depth test passes. The initial value is GL_KEEP. See glStencilOpSeparate."
- QString gl_stencil_back_ref = "Data returns one value, the reference value that is compared with the contents of the stencil buffer for back-facing polygons. The initial value is 0. See glStencilFuncSeparate."
- QString gl_stencil_back_value_mask = "Data returns one value, the mask that is used for back-facing polygons to mask both the stencil reference value and the stencil buffer value before they are compared. The initial value is all 1's. See glStencilFuncSeparate."
- QString gl_stencil_back_writemask = "Data returns one value, the mask that controls writing of the stencil bitplanes for back-facing polygons. The initial value is all 1's. See glStencilMaskSeparate."
- QString gl_stencil_clear_value = "Data returns one value, the index to which the stencil bitplanes are cleared. The initial value is 0. See glClearStencil."

- QString gl_stencil_fail = "Data returns one value, a symbolic constant indicating what action is taken when the stencil test fails. The initial value is GL_KEEP. See glStencilOp. This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See glStencilOpSeparate."
- QString gl_stencil_func = "Data returns one value, a symbolic constant indicating what function is used to compare the stencil reference value with the stencil buffer value. The initial value is GL_ALWAYS. See glStencilFunc. This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See glStencilFuncSeparate."
- QString gl_stencil_pass_depth_fail = "Data returns one value, a symbolic constant indicating what action is taken when the stencil test passes, but the depth test fails. The initial value is GL_KEEP. See glStencilOp. This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See glStencilOpSeparate."
- QString gl_stencil_pass_depth_pass = "Data returns one value, a symbolic constant indicating what action is taken when the stencil test passes and the depth test passes. The initial value is GL_KEEP. See glStencilOp. This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See glStencilOpSeparate."
- QString gl_stencil_ref = "Data returns one value, the reference value that is compared with the contents of the stencil buffer. The initial value is 0. See glStencilFunc. This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See glStencilFuncSeparate."
- QString gl_stencil_test = "Data returns a single boolean value indicating whether stencil testing of fragments is enabled. The initial value is GL_FALSE. See glStencilFunc and glStencilOp."
- QString gl_stencil_value_mask = "Data returns one value, the mask that is used to mask both the stencil reference value and the stencil buffer value before they are compared. The initial value is all 1's. See glStencilFunc. This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See glStencilFuncSeparate."
- QString gl_stencil_writemask = "Data returns one value, the mask that controls writing of the stencil bitplanes. The initial value is all 1's. See glStencilMask. This stencil state only affects non-polygons and front-facing polygons. Back-facing polygons use separate stencil state. See glStencilMaskSeparate."
- QString gl_stereo = "Data returns a single boolean value indicating whether stereo buffers (left and right) are supported."
- QString gl_subpixel_bits = "Data returns one value, an estimate of the number of bits of subpixel resolution that are used to position rasterized geometry in window coordinates. The value must be at least 4."
- QString gl_texture_binding_1d = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_1D. The initial value is 0. See glBindTexture."
- QString gl_texture_binding_1d_array = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_1D_ARRAY. The initial value is 0. See glBindTexture."
- QString gl_texture_binding_2d = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_2D. The initial value is 0. See glBindTexture."
- QString gl_texture_binding_2d_array = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_2D_ARRAY. The initial value is 0. See glBindTexture."
- QString gl_texture_binding_2d_multisample = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_2D_MULTISAMPLE. The initial value is 0. See glBindTexture."
- QString gl_texture_binding_2d_multisample_array = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_2D_MULTISAMPLE_ARRAY. The initial value is 0. See glBindTexture."
- QString gl_texture_binding_3d = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_3D. The initial value is 0. See glBindTexture."
- QString gl_texture_binding_buffer = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_BUFFER. The initial value is 0. See glBindTexture."
- QString gl_texture_binding_cube_map = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_CUBE_MAP. The initial value is 0. See glBindTexture."
- QString gl_texture_binding_rectangle = "Data returns a single value, the name of the texture currently bound to the target GL_TEXTURE_RECTANGLE. The initial value is 0. See glBindTexture."
- QString gl_texture_compression_hint = "Data returns a single value indicating the mode of the texture compression hint. The initial value is GL_DONT_CARE."
- QString gl_texture_buffer_offset_alignment = "Data returns a single value, the minimum required alignment for texture buffer sizes and offset. The initial value is 1. See glUniformBlockBinding."

- QString gl_timestamp = "Data returns a single value, the 64-bit value of the current GL time. See glQuery↩ Counter."
- QString gl_transform_feedback_buffer_binding = "When used with non-indexed variants of glGet (such as glGetIntegerv), data returns a single value, the name of the buffer object currently bound to the target GL↩ _TRANSFORM_FEEDBACK_BUFFER. If no buffer object is bound to this target, 0 is returned. When used with indexed variants of glGet (such as glGetIntegeri_v), data returns a single value, the name of the buffer object bound to the indexed transform feedback attribute stream. The initial value is 0 for all targets. See glBindBuffer, glBindBufferBase, and glBindBufferRange."
- QString gl_transform_feedback_buffer_start = "When used with indexed variants of glGet (such as glGet↩ Integer64i_v), data returns a single value, the start offset of the binding range for each transform feedback attribute stream. The initial value is 0 for all streams. See glBindBufferRange."
- QString gl_transform_feedback_buffer_size = "When used with indexed variants of glGet (such as glGet↩ Integer64i_v), data returns a single value, the size of the binding range for each transform feedback attribute stream. The initial value is 0 for all streams. See glBindBufferRange."
- QString gl_uniform_buffer_binding = "When used with non-indexed variants of glGet (such as glGetIntegerv), data returns a single value, the name of the buffer object currently bound to the target GL_UNIFORM_B↩ UFFER. If no buffer object is bound to this target, 0 is returned. When used with indexed variants of glGet (such as glGetIntegeri_v), data returns a single value, the name of the buffer object bound to the indexed uniform buffer binding point. The initial value is 0 for all targets. See glBindBuffer, glBindBufferBase, and glBindBufferRange."
- QString gl_uniform_buffer_offset_alignment = "Data returns a single value, the minimum required alignment for uniform buffer sizes and offset. The initial value is 1. See glUniformBlockBinding."
- QString gl_uniform_buffer_size = "When used with indexed variants of glGet (such as glGetInteger64i_v), data returns a single value, the size of the binding range for each indexed uniform buffer binding. The initial value is 0 for all bindings. See glBindBufferRange."
- QString gl_uniform_buffer_start = "When used with indexed variants of glGet (such as glGetInteger64i_v), data returns a single value, the start offset of the binding range for each indexed uniform buffer binding. The initial value is 0 for all bindings. See glBindBufferRange."
- QString gl_unpack_alignment = "Data returns one value, the byte alignment used for reading pixel data from memory. The initial value is 4. See glPixelStore."
- QString gl_unpack_image_height = "Data returns one value, the image height used for reading pixel data from memory. The initial is 0. See glPixelStore."
- QString gl_unpack_lsb_first = "Data returns a single boolean value indicating whether single-bit pixels being read from memory are read first from the least significant bit of each unsigned byte. The initial value is GL_FALSE. See glPixelStore."
- QString gl_unpack_row_length = "Data returns one value, the row length used for reading pixel data from memory. The initial value is 0. See glPixelStore."
- QString gl_unpack_skip_images = "Data returns one value, the number of pixel images skipped before the first pixel is read from memory. The initial value is 0. See glPixelStore."
- QString gl_unpack_skip_pixels = "Data returns one value, the number of pixel locations skipped before the first pixel is read from memory. The initial value is 0. See glPixelStore."
- QString gl_unpack_skip_rows = "Data returns one value, the number of rows of pixel locations skipped before the first pixel is read from memory. The initial value is 0. See glPixelStore."
- QString gl_unpack_swap_bytes = "Data returns a single boolean value indicating whether the bytes of two-byte and four-byte pixel indices and components are swapped after being read from memory. The initial value is GL_FALSE. See glPixelStore."
- QString gl_vertex_array_binding = "Data returns a single value, the name of the vertex array object currently bound to the context. If no vertex array object is bound to the context, 0 is returned. The initial value is 0. See glBindVertexArray."
- QString gl_vertex_binding_divisor = "Accepted by the indexed forms. data returns a single integer value representing the instance step divisor of the first element in the bound buffer's data store for vertex attribute bound to index."
- QString gl_vertex_binding_offset = "Accepted by the indexed forms. data returns a single integer value representing the byte offset of the first element in the bound buffer's data store for vertex attribute bound to index."
- QString gl_vertex_binding_stride = "Accepted by the indexed forms. data returns a single integer value representing the byte offset between the start of each element in the bound buffer's data store for vertex attribute bound to index."

- QString gl_max_vertex_attrib_relative_offset = "Data returns a single integer value containing the maximum offset that may be added to a vertex binding offset."
- QString gl_max_vertex_attrib_bindings = "Data returns a single integer value containing the maximum number of vertex buffers that may be bound."
- QString gl_viewport = "When used with non-indexed variants of glGet (such as glGetIntegerv), data returns four values: the x and y window coordinates of the viewport, followed by its width and height. Initially the x and y window coordinates are both set to 0, and the width and height are set to the width and height of the window into which the GL will do its rendering. See glViewport."
- QString gl_viewport_bounds_range = "Data returns two values, the minimum and maximum viewport bounds range. The minimum range should be at least [-32768, 32767]."
- QString gl_viewport_index_provoking_vertex = "Data returns one value, the implementation dependent specifc vertex of a primitive that is used to select the viewport index. If the value returned is equivalent to GL_PROVOKING_VERTEX, then the vertex selection follows the convention specified by glProvokingVertex. If the value returned is equivalent to GL_FIRST_VERTEX_CONVENTION, then the selection is always taken from the first vertex in the primitive. If the value returned is equivalent to GL_LAST_VERTEX_CONVENTION, then the selection is always taken from the last vertex in the primitive. If the value returned is equivalent to GL_UNDEFINED_VERTEX, then the selection is not guaranteed to be taken from any specific vertex in the primitive."
- QString gl_viewport_subpixel_bits = "Data returns a single value, the number of bits of sub-pixel precision which the GL uses to interpret the floating point viewport bounds. The minimum value is 0."
- QString gl_max_element_index = "Data returns a single value, the maximum index that may be specified during the transfer of generic vertex attributes to the GL."

6.3.1 Detailed Description

OpenGL variables descriptions. Structure containing the descriptions of the OpenGL variables.

6.3.2 Member Data Documentation

6.3.2.1 gl_active_texture

```
QString GLDescriptions::gl_active_texture = "Data returns a single value indicating the active
multitexture unit.  The initial value is GL_TEXTURE0.  See glActiveTexture."
```

6.3.2.2 gl_aliased_line_width_range

```
QString GLDescriptions::gl_aliased_line_width_range = "Data returns a pair of values indicating
the range of widths supported for aliased lines.  See glLineWidth."
```

6.3.2.3 gl_array_buffer_binding

```
QString GLDescriptions::gl_array_buffer_binding = "Data returns a single value, the name of
the buffer object currently bound to the target GL_ARRAY_BUFFER. If no buffer object is bound
to this target, 0 is returned.  The initial value is 0.  See glBindBuffer."
```

6.3.2.4 gl_blend

```
QString GLDescriptions::gl_blend = "Data returns a single boolean value indicating whether
blending is enabled.  The initial value is GL_FALSE. See glBlendFunc."
```

6.3.2.5 gl_blend_color

```
QString GLDescriptions::gl_blend_color = "Data returns four values, the red, green, blue, and
alpha values which are the components of the blend color.  See glBlendColor."
```

6.3.2.6 gl_blend_dst_alpha

```
QString GLDescriptions::gl_blend_dst_alpha = "Data returns one value, the symbolic constant
identifying the alpha destination blend function.  The initial value is GL_ZERO. See glBlend↩
Func and glBlendFuncSeparate."
```

6.3.2.7 gl_blend_dst_rgb

```
QString GLDescriptions::gl_blend_dst_rgb = "Data returns one value, the symbolic constant
identifying the RGB destination blend function.  The initial value is GL_ZERO. See glBlendFunc
and glBlendFuncSeparate."
```

6.3.2.8 gl_blend_equation_alpha

```
QString GLDescriptions::gl_blend_equation_alpha = "Data returns one value, a symbolic constant
indicating whether the Alpha blend equation is GL_FUNC_ADD, GL_FUNC_SUBTRACT, GL_FUNC_REVERS↩
E_SUBTRACT, GL_MIN or GL_MAX. See glBlendEquationSeparate."
```

6.3.2.9 gl_blend_equation_rgb

```
QString GLDescriptions::gl_blend_equation_rgb = "Data returns one value, a symbolic constant
indicating whether the RGB blend equation is GL_FUNC_ADD, GL_FUNC_SUBTRACT, GL_FUNC_REVERSE_↩
SUBTRACT, GL_MIN or GL_MAX. See glBlendEquationSeparate."
```

6.3.2.10 gl_blend_src_alpha

```
QString GLDescriptions::gl_blend_src_alpha = "Data returns one value, the symbolic constant
identifying the alpha source blend function.  The initial value is GL_ONE. See glBlendFunc and
glBlendFuncSeparate."
```

6.3.2.11 gl_blend_src_rgb

```
QString GLDescriptions::gl_blend_src_rgb = "Data returns one value, the symbolic constant
identifying the RGB source blend function.  The initial value is GL_ONE. See glBlendFunc and
glBlendFuncSeparate."
```

6.3.2.12 gl_color_clear_value

```
QString GLDescriptions::gl_color_clear_value = "Data returns four values:  the red, green,
blue, and alpha values used to clear the color buffers.  Integer values, if requested, are
linearly mapped from the internal floating-point representation such that 1.0 returns the most
positive representable integer value, and 1.0 returns the most negative representable integer
value.  The initial value is (0, 0, 0, 0).  See glClearColor."
```

6.3.2.13 gl_color_logic_op

```
QString GLDescriptions::gl_color_logic_op = "Data returns a single boolean value indicating
whether a fragment's RGBA color values are merged into the framebuffer using a logical operation.
The initial value is GL_FALSE. See glLogicOp."
```

6.3.2.14 gl_color_writemask

```
QString GLDescriptions::gl_color_writemask = "Data returns four boolean values:  the red,
green, blue, and alpha write enables for the color buffers.  The initial value is (GL_TRUE,
GL_TRUE, GL_TRUE, GL_TRUE). See glColorMask."
```

6.3.2.15 gl_compressed_texture_formats

```
QString GLDescriptions::gl_compressed_texture_formats = "Data returns a list of symbolic constants
of length GL_NUM_COMPRESSED_TEXTURE_FORMATS indicating which compressed texture formats are
available.  See glCompressedTexImage2D."
```

6.3.2.16 gl_context_flags

```
QString GLDescriptions::gl_context_flags = "Data returns one value, the flags with which the
context was created (such as debugging functionality)."
```

6.3.2.17 gl_cull_face

```
QString GLDescriptions::gl_cull_face = "Data returns a single boolean value indicating whether
polygon culling is enabled.  The initial value is GL_FALSE. See glCullFace."
```

6.3.2.18 gl_cull_face_mode

```
QString GLDescriptions::gl_cull_face_mode = "Data returns a single value indicating the mode
of polygon culling.  The initial value is GL_BACK. See glCullFace."
```

6.3.2.19 gl_current_program

```
QString GLDescriptions::gl_current_program = "Data returns one value, the name of the program
object that is currently active, or 0 if no program object is active.  See glUseProgram."
```

6.3.2.20 gl_debug_group_stack_depth

```
QString GLDescriptions::gl_debug_group_stack_depth = "Data returns a single value, the current
depth of the debug message group stack."
```

6.3.2.21 gl_depth_clear_value

```
QString GLDescriptions::gl_depth_clear_value = "Data returns one value, the value that is
used to clear the depth buffer.  Integer values, if requested, are linearly mapped from the
internal floating-point representation such that 1.0 returns the most positive representable
integer value, and 1.0 returns the most negative representable integer value.  The initial
value is 1.  See glClearDepth."
```

6.3.2.22 gl_depth_func

```
QString GLDescriptions::gl_depth_func = "Data returns one value, the symbolic constant that
indicates the depth comparison function.  The initial value is GL_LESS. See glDepthFunc."
```

6.3.2.23 gl_depth_range

```
QString GLDescriptions::gl_depth_range = "Data returns two values:  the near and far mapping
limits for the depth buffer.  Integer values, if requested, are linearly mapped from the internal
floating-point representation such that 1.0 returns the most positive representable integer
value, and 1.0 returns the most negative representable integer value.  The initial value is
(0, 1).  See glDepthRange."
```

6.3.2.24 gl_depth_test

```
QString GLDescriptions::gl_depth_test = "Data returns a single boolean value indicating whether
depth testing of fragments is enabled.  The initial value is GL_FALSE. See glDepthFunc and
glDepthRange."
```

6.3.2.25 gl_depth_writemask

```
QString GLDescriptions::gl_depth_writemask = "Data returns a single boolean value indicating
if the depth buffer is enabled for writing.  The initial value is GL_TRUE. See glDepthMask."
```

6.3.2.26 gl_dispatch_indirect_buffer_binding

```
QString GLDescriptions::gl_dispatch_indirect_buffer_binding = "Data returns a single value,
the name of the buffer object currently bound to the target GL_DISPATCH_INDIRECT_BUFFER. If no
buffer object is bound to this target, 0 is returned.  The initial value is 0.  See glBind↩
Buffer."
```

6.3.2.27 gl_dither

```
QString GLDescriptions::gl_dither = "Data returns a single boolean value indicating whether
dithering of fragment colors and indices is enabled.  The initial value is GL_TRUE."
```

6.3.2.28 gl_doublebuffer

```
QString GLDescriptions::gl_doublebuffer = "Data returns a single boolean value indicating
whether double buffering is supported."
```

6.3.2.29 gl_draw_buffer

```
QString GLDescriptions::gl_draw_buffer = "Data returns one value, a symbolic constant indicating
which buffers are being drawn to.  See glDrawBuffer.  The initial value is GL_BACK if there
are back buffers, otherwise it is GL_FRONT."
```

6.3.2.30 gl_draw_framebuffer_binding

```
QString GLDescriptions::gl_draw_framebuffer_binding = "Data returns one value, the name of
the framebuffer object currently bound to the GL_DRAW_FRAMEBUFFER target.  If the default
framebuffer is bound, this value will be zero.  The initial value is zero.  See glBindFramebuffer."
```

6.3.2.31 gl_element_array_buffer_binding

```
QString GLDescriptions::gl_element_array_buffer_binding = "Data returns a single value, the
name of the buffer object currently bound to the target GL_ELEMENT_ARRAY_BUFFER. If no buffer
object is bound to this target, 0 is returned.  The initial value is 0.  See glBindBuffer."
```

6.3.2.32 gl_fragment_shader_derivative_hint

```
QString GLDescriptions::gl_fragment_shader_derivative_hint = "Data returns one value, a symbolic
constant indicating the mode of the derivative accuracy hint for fragment shaders.  The initial
value is GL_DONT_CARE. See glHint."
```

6.3.2.33 gl_implementation_color_read_format

```
QString GLDescriptions::gl_implementation_color_read_format = "Data returns a single GLenum
value indicating the implementation's preferred pixel data format.  See glReadPixels."
```

6.3.2.34 gl_implementation_color_read_type

```
QString GLDescriptions::gl_implementation_color_read_type = "Data returns a single GLenum
value indicating the implementation's preferred pixel data type.  See glReadPixels."
```

6.3.2.35 gl_layer_provoking_vertex

```
QString GLDescriptions::gl_layer_provoking_vertex = "Data returns one value, the implementation
dependent specifc vertex of a primitive that is used to select the rendering layer.  If the
value returned is equivalent to GL_PROVOKING_VERTEX, then the vertex selection follows the
convention specified by glProvokingVertex.  If the value returned is equivalent to GL_FIRST↩
_VERTEX_CONVENTION, then the selection is always taken from the first vertex in the primitive.
If the value returned is equivalent to GL_LAST_VERTEX_CONVENTION, then the selection is always
taken from the last vertex in the primitive.  If the value returned is equivalent to GL_UNDE↩
FINED_VERTEX, then the selection is not guaranteed to be taken from any specific vertex in the
primitive."
```

6.3.2.36 gl_line_smooth

```
QString GLDescriptions::gl_line_smooth = "Data returns a single boolean value indicating whether
antialiasing of lines is enabled.  The initial value is GL_FALSE. See glLineWidth."
```

6.3.2.37 gl_line_smooth_hint

```
QString GLDescriptions::gl_line_smooth_hint = "Data returns one value, a symbolic constant
indicating the mode of the line antialiasing hint.  The initial value is GL_DONT_CARE. See
glHint."
```

6.3.2.38 gl_line_width

```
QString GLDescriptions::gl_line_width = "Data returns one value, the line width as specified
with glLineWidth.  The initial value is 1."
```

6.3.2.39 gl_logic_op_mode

```
QString GLDescriptions::gl_logic_op_mode = "Data returns one value, a symbolic constant indicating
the selected logic operation mode.  The initial value is GL_COPY. See glLogicOp."
```

6.3.2.40 gl_major_version

```
QString GLDescriptions::gl_major_version = "Data returns one value, the major version number
of the OpenGL API supported by the current context."
```

6.3.2.41 gl_max_3d_texture_size

```
QString GLDescriptions::gl_max_3d_texture_size = "Data returns one value, a rough estimate of
the largest 3D texture that the GL can handle.  The value must be at least 64.  Use GL_PROXY↩
_TEXTURE_3D to determine if a texture is too large.  See glTexImage3D."
```

6.3.2.42 gl_max_array_texture_layers

```
QString GLDescriptions::gl_max_array_texture_layers = "Data returns one value.  The value
indicates the maximum number of layers allowed in an array texture, and must be at least 256.
See glTexImage2D."
```

6.3.2.43 gl_max_clip_distances

```
QString GLDescriptions::gl_max_clip_distances = "Data returns one value, the maximum number of
application-defined clipping distances.  The value must be at least 8."
```

6.3.2.44 gl_max_color_texture_samples

```
QString GLDescriptions::gl_max_color_texture_samples = "Data returns one value, the maximum
number of samples in a color multisample texture."
```

6.3.2.45 gl_max_combined_atomic_counters

```
QString GLDescriptions::gl_max_combined_atomic_counters = "Data returns a single value, the
maximum number of atomic counters available to all active shaders."
```

6.3.2.46 gl_max_combined_compute_uniform_components

```
QString GLDescriptions::gl_max_combined_compute_uniform_components = "Data returns one value,
the number of words for compute shader uniform variables in all uniform blocks (including
default).  The value must be at least 1.  See glUniform."
```

6.3.2.47 gl_max_combined_fragment_uniform_components

```
QString GLDescriptions::gl_max_combined_fragment_uniform_components = "Data returns one value,
the number of words for fragment shader uniform variables in all uniform blocks (including
default).  The value must be at least 1.  See glUniform."
```

6.3.2.48 gl_max_combined_geometry_uniform_components

```
QString GLDescriptions::gl_max_combined_geometry_uniform_components = "Data returns one value,
the number of words for geometry shader uniform variables in all uniform blocks (including
default).  The value must be at least 1.  See glUniform."
```

6.3.2.49 gl_max_combined_shader_storage_blocks

```
QString GLDescriptions::gl_max_combined_shader_storage_blocks = "Data returns one value, the
maximum total number of active shader storage blocks that may be accessed by all active shaders."
```

6.3.2.50 gl_max_combined_texture_image_units

```
QString GLDescriptions::gl_max_combined_texture_image_units = "Data returns one value, the
maximum supported texture image units that can be used to access texture maps from the vertex
shader and the fragment processor combined.  If both the vertex shader and the fragment processing
stage access the same texture image unit, then that counts as using two texture image units
against this limit.  The value must be at least 48.  See glActiveTexture."
```

6.3.2.51 gl_max_combined_uniform_blocks

```
QString GLDescriptions::gl_max_combined_uniform_blocks = "Data returns one value, the maximum
number of uniform blocks per program.  The value must be at least 70.  See glUniformBlock↩
Binding."
```

6.3.2.52 gl_max_combined_vertex_uniform_components

```
QString GLDescriptions::gl_max_combined_vertex_uniform_components = "Data returns one value,
the number of words for vertex shader uniform variables in all uniform blocks (including default).
The value must be at least 1.  See glUniform."
```

6.3.2.53 gl_max_compute_atomic_counter_buffers

```
QString GLDescriptions::gl_max_compute_atomic_counter_buffers = "Data returns a single value,
the maximum number of atomic counter buffers that may be accessed by a compute shader."
```

6.3.2.54 gl_max_compute_atomic_counters

```
QString GLDescriptions::gl_max_compute_atomic_counters = "Data returns a single value, the
maximum number of atomic counters available to compute shaders."
```

6.3.2.55 gl_max_compute_shader_storage_blocks

```
QString GLDescriptions::gl_max_compute_shader_storage_blocks = "Data returns one value, the
maximum number of active shader storage blocks that may be accessed by a compute shader."
```

6.3.2.56 gl_max_compute_texture_image_units

```
QString GLDescriptions::gl_max_compute_texture_image_units = "Data returns one value, the
maximum supported texture image units that can be used to access texture maps from the compute
shader.  The value may be at least 16.  See glActiveTexture."
```

6.3.2.57 gl_max_compute_uniform_blocks

```
QString GLDescriptions::gl_max_compute_uniform_blocks = "Data returns one value, the maximum
number of uniform blocks per compute shader.  The value must be at least 14.  See glUniform↩
BlockBinding."
```

6.3.2.58 gl_max_compute_uniform_components

```
QString GLDescriptions::gl_max_compute_uniform_components = "Data returns one value, the maximum
number of individual floating-point, integer, or boolean values that can be held in uniform
variable storage for a compute shader.  The value must be at least 1024.  See glUniform."
```

6.3.2.59 gl_max_compute_work_group_count

```
QString GLDescriptions::gl_max_compute_work_group_count = "Accepted by the indexed versions
of glGet.  data the maximum number of work groups that may be dispatched to a compute shader.
Indices 0, 1, and 2 correspond to the X, Y and Z dimensions, respectively."
```

6.3.2.60 gl_max_compute_work_group_size

```
QString GLDescriptions::gl_max_compute_work_group_size = "Accepted by the indexed versions
of glGet.  data the maximum size of a work groups that may be used during compilation of a
compute shader.  Indices 0, 1, and 2 correspond to the X, Y and Z dimensions, respectively."
```

6.3.2.61 gl_max_cube_map_texture_size

```
QString GLDescriptions::gl_max_cube_map_texture_size = "Data returns one value.  The value
gives a rough estimate of the largest cube-map texture that the GL can handle.  The value must
be at least 1024.  Use GL_PROXY_TEXTURE_CUBE_MAP to determine if a texture is too large.  See
glTexImage2D."
```

6.3.2.62 gl_max_debug_group_stack_depth

```
QString GLDescriptions::gl_max_debug_group_stack_depth = "Data returns a single value, the
maximum depth of the debug message group stack."
```

6.3.2.63 gl_max_depth_texture_samples

```
QString GLDescriptions::gl_max_depth_texture_samples = "Data returns one value, the maximum
number of samples in a multisample depth or depth-stencil texture."
```

6.3.2.64 gl_max_draw_buffers

```
QString GLDescriptions::gl_max_draw_buffers = "Data returns one value, the maximum number of
simultaneous outputs that may be written in a fragment shader.  The value must be at least 8.
See glDrawBuffers."
```

6.3.2.65 gl_max_dual_source_draw_buffers

```
QString GLDescriptions::gl_max_dual_source_draw_buffers = "Data returns one value, the maximum
number of active draw buffers when using dual-source blending.  The value must be at least 1.
See glBlendFunc and glBlendFuncSeparate."
```

6.3.2.66 gl_max_element_index

```
QString GLDescriptions::gl_max_element_index = "Data returns a single value, the maximum index
that may be specified during the transfer of generic vertex attributes to the GL."
```

6.3.2.67 gl_max_elements_indices

```
QString GLDescriptions::gl_max_elements_indices = "Data returns one value, the recommended
maximum number of vertex array indices.  See glDrawRangeElements."
```

6.3.2.68 gl_max_elements_vertices

```
QString GLDescriptions::gl_max_elements_vertices = "Data returns one value, the recommended
maximum number of vertex array vertices.  See glDrawRangeElements."
```

6.3.2.69 gl_max_fragment_atomic_counters

```
QString GLDescriptions::gl_max_fragment_atomic_counters = "Data returns a single value, the
maximum number of atomic counters available to fragment shaders."
```

6.3.2.70 gl_max_fragment_input_components

```
QString GLDescriptions::gl_max_fragment_input_components = "Data returns one value, the maximum
number of components of the inputs read by the fragment shader, which must be at least 128."
```

6.3.2.71 gl_max_fragment_shader_storage_blocks

```
QString GLDescriptions::gl_max_fragment_shader_storage_blocks = "Data returns one value, the
maximum number of active shader storage blocks that may be accessed by a fragment shader."
```

6.3.2.72 gl_max_fragment_uniform_blocks

```
QString GLDescriptions::gl_max_fragment_uniform_blocks = "Data returns one value, the maximum
number of uniform blocks per fragment shader.  The value must be at least 12.  See glUniform↩
BlockBinding."
```

6.3.2.73 gl_max_fragment_uniform_components

```
QString GLDescriptions::gl_max_fragment_uniform_components = "Data returns one value, the
maximum number of individual floating-point, integer, or boolean values that can be held in
uniform variable storage for a fragment shader.  The value must be at least 1024.  See gl↩
Uniform."
```

6.3.2.74 gl_max_fragment_uniform_vectors

```
QString GLDescriptions::gl_max_fragment_uniform_vectors = "Data returns one value, the maximum
number of individual 4-vectors of floating-point, integer, or boolean values that can be held
in uniform variable storage for a fragment shader.  The value is equal to the value of GL_MA↩
X_FRAGMENT_UNIFORM_COMPONENTS divided by 4 and must be at least 256.  See glUniform."
```

6.3.2.75 gl_max_framebuffer_height

```
QString GLDescriptions::gl_max_framebuffer_height = "Data returns one value, the maximum height
for a framebuffer that has no attachments, which must be at least 16384.  See glFramebuffer↩
Parameter."
```

6.3.2.76 gl_max_framebuffer_layers

```
QString GLDescriptions::gl_max_framebuffer_layers = "Data returns one value, the maximum number
of layers for a framebuffer that has no attachments, which must be at least 2048.  See gl↩
FramebufferParameter."
```

6.3.2.77 gl_max_framebuffer_samples

```
QString GLDescriptions::gl_max_framebuffer_samples = "Data returns one value, the maximum
samples in a framebuffer that has no attachments, which must be at least 4.  See glFramebuffer↩
Parameter."
```

6.3.2.78 gl_max_framebuffer_width

```
QString GLDescriptions::gl_max_framebuffer_width = "Data returns one value, the maximum width
for a framebuffer that has no attachments, which must be at least 16384.  See glFramebuffer↩
Parameter."
```

6.3.2.79 gl_max_geometry_atomic_counters

```
QString GLDescriptions::gl_max_geometry_atomic_counters = "Data returns a single value, the
maximum number of atomic counters available to geometry shaders."
```

6.3.2.80 gl_max_geometry_input_components

```
QString GLDescriptions::gl_max_geometry_input_components = "Data returns one value, the maximum
number of components of inputs read by a geometry shader, which must be at least 64."
```

6.3.2.81 gl_max_geometry_output_components

```
QString GLDescriptions::gl_max_geometry_output_components = "Data returns one value, the maximum
number of components of outputs written by a geometry shader, which must be at least 128."
```

6.3.2.82 gl_max_geometry_shader_storage_blocks

```
QString GLDescriptions::gl_max_geometry_shader_storage_blocks = "Data returns one value, the
maximum number of active shader storage blocks that may be accessed by a geometry shader."
```

6.3.2.83 gl_max_geometry_texture_image_units

```
QString GLDescriptions::gl_max_geometry_texture_image_units = "Data returns one value, the
maximum supported texture image units that can be used to access texture maps from the geometry
shader.  The value must be at least 16.  See glActiveTexture."
```

6.3.2.84 gl_max_geometry_uniform_blocks

```
QString GLDescriptions::gl_max_geometry_uniform_blocks = "Data returns one value, the maximum
number of uniform blocks per geometry shader.  The value must be at least 12.  See glUniform↩
BlockBinding."
```

6.3.2.85 gl_max_geometry_uniform_components

```
QString GLDescriptions::gl_max_geometry_uniform_components = "Data returns one value, the
maximum number of individual floating-point, integer, or boolean values that can be held in
uniform variable storage for a geometry shader.  The value must be at least 1024.  See gl↩
Uniform."
```

6.3.2.86 gl_max_integer_samples

```
QString GLDescriptions::gl_max_integer_samples = "Data returns one value, the maximum number
of samples supported in integer format multisample buffers."
```

6.3.2.87 gl_max_label_length

```
QString GLDescriptions::gl_max_label_length = "Data returns one value, the maximum length of a
label that may be assigned to an object.  See glObjectLabel and glObjectPtrLabel."
```

6.3.2.88 gl_max_program_texel_offset

```
QString GLDescriptions::gl_max_program_texel_offset = "Data returns one value, the maximum
texel offset allowed in a texture lookup, which must be at least 7."
```

6.3.2.89 gl_max_rectangle_texture_size

```
QString GLDescriptions::gl_max_rectangle_texture_size = "Data returns one value.  The value
gives a rough estimate of the largest rectangular texture that the GL can handle.  The value
must be at least 1024.  Use GL_PROXY_TEXTURE_RECTANGLE to determine if a texture is too large.
See glTexImage2D."
```

6.3.2.90 gl_max_renderbuffer_size

```
QString GLDescriptions::gl_max_renderbuffer_size = "Data returns one value.  The value indicates
the maximum supported size for renderbuffers.  See glFramebufferRenderbuffer."
```

6.3.2.91 gl_max_sample_mask_words

```
QString GLDescriptions::gl_max_sample_mask_words = "Data returns one value, the maximum number
of sample mask words."
```

6.3.2.92 gl_max_server_wait_timeout

```
QString GLDescriptions::gl_max_server_wait_timeout = "Data returns one value, the maximum gl↩
WaitSync timeout interval."
```

6.3.2.93 gl_max_shader_storage_buffer_bindings

```
QString GLDescriptions::gl_max_shader_storage_buffer_bindings = "Data returns one value, the
maximum number of shader storage buffer binding points on the context, which must be at least
8."
```

6.3.2.94 gl_max_tess_control_atomic_counters

```
QString GLDescriptions::gl_max_tess_control_atomic_counters = "Data returns a single value,
the maximum number of atomic counters available to tessellation control shaders."
```

6.3.2.95 gl_max_tess_control_shader_storage_blocks

```
QString GLDescriptions::gl_max_tess_control_shader_storage_blocks = "Data returns one value,
the maximum number of active shader storage blocks that may be accessed by a tessellation
control shader."
```

6.3.2.96 gl_max_tess_evaluation_atomic_counters

```
QString GLDescriptions::gl_max_tess_evaluation_atomic_counters = "Data returns a single value,
the maximum number of atomic counters available to tessellation evaluation shaders."
```

6.3.2.97 gl_max_tess_evaluation_shader_storage_blocks

```
QString GLDescriptions::gl_max_tess_evaluation_shader_storage_blocks = "Data returns one value,
the maximum number of active shader storage blocks that may be accessed by a tessellation
evaluation shader."
```

6.3.2.98 gl_max_texture_buffer_size

```
QString GLDescriptions::gl_max_texture_buffer_size = "Data returns one value.  The value gives
the maximum number of texels allowed in the texel array of a texture buffer object.  Value
must be at least 65536."
```

6.3.2.99 gl_max_texture_image_units

```
QString GLDescriptions::gl_max_texture_image_units = "Data returns one value, the maximum
supported texture image units that can be used to access texture maps from the fragment shader.
The value must be at least 16.  See glActiveTexture."
```

6.3.2.100 gl_max_texture_lod_bias

```
QString GLDescriptions::gl_max_texture_lod_bias = "Data returns one value, the maximum, absolute
value of the texture level-of-detail bias.  The value must be at least 2.0."
```

6.3.2.101 gl_max_texture_size

```
QString GLDescriptions::gl_max_texture_size = "Data returns one value.  The value gives a
rough estimate of the largest texture that the GL can handle.  The value must be at least
1024.  Use a proxy texture target such as GL_PROXY_TEXTURE_1D or GL_PROXY_TEXTURE_2D to determine
if a texture is too large.  See glTexImage1D and glTexImage2D."
```

6.3.2.102 gl_max_uniform_block_size

```
QString GLDescriptions::gl_max_uniform_block_size = "Data returns one value, the maximum size
in basic machine units of a uniform block, which must be at least 16384."
```

6.3.2.103 gl_max_uniform_buffer_bindings

```
QString GLDescriptions::gl_max_uniform_buffer_bindings = "Data returns one value, the maximum
number of uniform buffer binding points on the context, which must be at least 36."
```

6.3.2.104 gl_max_uniform_locations

```
QString GLDescriptions::gl_max_uniform_locations = "Data returns one value, the maximum number
of explicitly assignable uniform locations, which must be at least 1024."
```

6.3.2.105 gl_max_varying_components

```
QString GLDescriptions::gl_max_varying_components = "Data returns one value, the number components
for varying variables, which must be at least 60."
```

6.3.2.106 gl_max_varying_floats

```
QString GLDescriptions::gl_max_varying_floats = "Data returns one value, the maximum number of
interpolators available for processing varying variables used by vertex and fragment shaders.
This value represents the number of individual floating-point values that can be interpolated;
varying variables declared as vectors, matrices, and arrays will all consume multiple interpolators.
The value must be at least 32."
```

6.3.2.107 gl_max_varying_vectors

```
QString GLDescriptions::gl_max_varying_vectors = "Data returns one value, the number 4-vectors
for varying variables, which is equal to the value of GL_MAX_VARYING_COMPONENTS and must be at
least 15."
```

6.3.2.108 gl_max_vertex_atomic_counters

```
QString GLDescriptions::gl_max_vertex_atomic_counters = "Data returns a single value, the
maximum number of atomic counters available to vertex shaders."
```

6.3.2.109 gl_max_vertex_attrib_bindings

```
QString GLDescriptions::gl_max_vertex_attrib_bindings = "Data returns a single integer value
containing the maximum number of vertex buffers that may be bound."
```

6.3.2.110 gl_max_vertex_attrib_relative_offset

```
QString GLDescriptions::gl_max_vertex_attrib_relative_offset = "Data returns a single integer
value containing the maximum offset that may be added to a vertex binding offset."
```

6.3.2.111 gl_max_vertex_attribs

```
QString GLDescriptions::gl_max_vertex_attribs = "Data returns one value, the maximum number
of 4-component generic vertex attributes accessible to a vertex shader.  The value must be at
least 16.  See glVertexAttrib."
```

6.3.2.112 gl_max_vertex_output_components

```
QString GLDescriptions::gl_max_vertex_output_components = "Data returns one value, the maximum
number of components of output written by a vertex shader, which must be at least 64."
```

6.3.2.113 gl_max_vertex_shader_storage_blocks

```
QString GLDescriptions::gl_max_vertex_shader_storage_blocks = "Data returns one value, the
maximum number of active shader storage blocks that may be accessed by a vertex shader."
```

6.3.2.114 gl_max_vertex_texture_image_units

```
QString GLDescriptions::gl_max_vertex_texture_image_units = "Data returns one value, the maximum
supported texture image units that can be used to access texture maps from the vertex shader.
The value may be at least 16.  See glActiveTexture."
```

6.3.2.115 gl_max_vertex_uniform_blocks

```
QString GLDescriptions::gl_max_vertex_uniform_blocks = "Data returns one value, the maximum
number of uniform blocks per vertex shader.  The value must be at least 12.  See glUniform↩
BlockBinding."
```

6.3.2.116 gl_max_vertex_uniform_components

```
QString GLDescriptions::gl_max_vertex_uniform_components = "Data returns one value, the maximum
number of individual floating-point, integer, or boolean values that can be held in uniform
variable storage for a vertex shader.  The value must be at least 1024.  See glUniform."
```

6.3.2.117 gl_max_vertex_uniform_vectors

```
QString GLDescriptions::gl_max_vertex_uniform_vectors = "Data returns one value, the maximum
number of 4-vectors that may be held in uniform variable storage for the vertex shader.  The
value of GL_MAX_VERTEX_UNIFORM_VECTORS is equal to the value of GL_MAX_VERTEX_UNIFORM_COMPON↩
ENTS and must be at least 256."
```

6.3.2.118 gl_max_viewport_dims

```
QString GLDescriptions::gl_max_viewport_dims = "Data returns two values:  the maximum supported
width and height of the viewport.  These must be at least as large as the visible dimensions
of the display being rendered to.  See glViewport."
```

6.3.2.119 gl_max_viewports

```
QString GLDescriptions::gl_max_viewports = "Data returns one value, the maximum number of
simultaneous viewports that are supported.  The value must be at least 16.  See glViewport↩
Indexed."
```

6.3.2.120 gl_min_map_buffer_alignment

```
QString GLDescriptions::gl_min_map_buffer_alignment = "Data returns one value, the minimum
alignment in basic machine units of pointers returned fromglMapBuffer and glMapBufferRange.
This value must be a power of two and must be at least 64."
```

6.3.2.121 gl_min_program_texel_offset

```
QString GLDescriptions::gl_min_program_texel_offset = "Data returns one value, the minimum
texel offset allowed in a texture lookup, which must be at most -8."
```

6.3.2.122 gl_minor_version

```
QString GLDescriptions::gl_minor_version = "Data returns one value, the minor version number
of the OpenGL API supported by the current context."
```

6.3.2.123 gl_num_compressed_texture_formats

```
QString GLDescriptions::gl_num_compressed_texture_formats = "Data returns a single integer
value indicating the number of available compressed texture formats.  The minimum value is 4.
See glCompressedTexImage2D."
```

6.3.2.124 gl_num_extensions

```
QString GLDescriptions::gl_num_extensions = "Data returns one value, the number of extensions
supported by the GL implementation for the current context.  See glGetString."
```

6.3.2.125 gl_num_program_binary_formats

```
QString GLDescriptions::gl_num_program_binary_formats = "Data returns one value, the number of
program binary formats supported by the implementation."
```

6.3.2.126 gl_num_shader_binary_formats

```
QString GLDescriptions::gl_num_shader_binary_formats = "Data returns one value, the number of
binary shader formats supported by the implementation.  If this value is greater than zero,
then the implementation supports loading binary shaders.  If it is zero, then the loading of
binary shaders by the implementation is not supported."
```

6.3.2.127 gl_pack_alignment

```
QString GLDescriptions::gl_pack_alignment = "Data returns one value, the byte alignment used
for writing pixel data to memory.  The initial value is 4.  See glPixelStore."
```

6.3.2.128 gl_pack_image_height

```
QString GLDescriptions::gl_pack_image_height = "Data returns one value, the image height used
for writing pixel data to memory.  The initial value is 0.  See glPixelStore."
```

6.3.2.129 gl_pack_lsb_first

```
QString GLDescriptions::gl_pack_lsb_first = "Data returns a single boolean value indicating
whether single-bit pixels being written to memory are written first to the least significant
bit of each unsigned byte.  The initial value is GL_FALSE. See glPixelStore."
```

6.3.2.130 gl_pack_row_length

```
QString GLDescriptions::gl_pack_row_length = "Data returns one value, the row length used for
writing pixel data to memory.  The initial value is 0.  See glPixelStore."
```

6.3.2.131 gl_pack_skip_images

```
QString GLDescriptions::gl_pack_skip_images = "Data returns one value, the number of pixel
images skipped before the first pixel is written into memory.  The initial value is 0.  See
glPixelStore."
```

6.3.2.132 gl_pack_skip_pixels

```
QString GLDescriptions::gl_pack_skip_pixels = "Data returns one value, the number of pixel
locations skipped before the first pixel is written into memory.  The initial value is 0.  See
glPixelStore."
```

6.3.2.133 gl_pack_skip_rows

```
QString GLDescriptions::gl_pack_skip_rows = "Data returns one value, the number of rows of
pixel locations skipped before the first pixel is written into memory.  The initial value is
0.  See glPixelStore."
```

6.3.2.134 gl_pack_swap_bytes

```
QString GLDescriptions::gl_pack_swap_bytes = "Data returns a single boolean value indicating
whether the bytes of two-byte and four-byte pixel indices and components are swapped before
being written to memory.  The initial value is GL_FALSE. See glPixelStore."
```

6.3.2.135 gl_pixel_pack_buffer_binding

```
QString GLDescriptions::gl_pixel_pack_buffer_binding = "Data returns a single value, the name
of the buffer object currently bound to the target GL_PIXEL_PACK_BUFFER. If no buffer object
is bound to this target, 0 is returned.  The initial value is 0.  See glBindBuffer."
```

6.3.2.136 gl_pixel_unpack_buffer_binding

```
QString GLDescriptions::gl_pixel_unpack_buffer_binding = "Data returns a single value, the
name of the buffer object currently bound to the target GL_PIXEL_UNPACK_BUFFER. If no buffer
object is bound to this target, 0 is returned.  The initial value is 0.  See glBindBuffer."
```

6.3.2.137 gl_point_fade_threshold_size

```
QString GLDescriptions::gl_point_fade_threshold_size = "Data returns one value, the point size
threshold for determining the point size.  See glPointParameter."
```

6.3.2.138 gl_point_size

```
QString GLDescriptions::gl_point_size = "Data returns one value, the point size as specified
by glPointSize.  The initial value is 1."
```

6.3.2.139 gl_point_size_granularity

```
QString GLDescriptions::gl_point_size_granularity = "Data returns one value, the size difference
between adjacent supported sizes for antialiased points.  See glPointSize."
```

6.3.2.140 gl_point_size_range

```
QString GLDescriptions::gl_point_size_range = "Data returns two values:  the smallest and
largest supported sizes for antialiased points.  The smallest size must be at most 1, and the
largest size must be at least 1.  See glPointSize."
```

6.3.2.141 gl_polygon_offset_factor

```
QString GLDescriptions::gl_polygon_offset_factor = "Data returns one value, the scaling factor
used to determine the variable offset that is added to the depth value of each fragment generated
when a polygon is rasterized.  The initial value is 0.  See glPolygonOffset."
```

6.3.2.142 gl_polygon_offset_fill

```
QString GLDescriptions::gl_polygon_offset_fill = "Data returns a single boolean value indicating
whether polygon offset is enabled for polygons in fill mode.  The initial value is GL_FALSE.
See glPolygonOffset."
```

6.3.2.143 gl_polygon_offset_line

```
QString GLDescriptions::gl_polygon_offset_line = "Data returns a single boolean value indicating
whether polygon offset is enabled for polygons in line mode.  The initial value is GL_FALSE.
See glPolygonOffset."
```

6.3.2.144 gl_polygon_offset_point

```
QString GLDescriptions::gl_polygon_offset_point = "Data returns a single boolean value indicating
whether polygon offset is enabled for polygons in point mode.  The initial value is GL_FALSE.
See glPolygonOffset."
```

6.3.2.145 gl_polygon_offset_units

```
QString GLDescriptions::gl_polygon_offset_units = "Data returns one value.  This value is
multiplied by an implementation-specific value and then added to the depth value of each fragment
generated when a polygon is rasterized.  The initial value is 0.  See glPolygonOffset."
```

6.3.2.146 gl_polygon_smooth

```
QString GLDescriptions::gl_polygon_smooth = "Data returns a single boolean value indicating
whether antialiasing of polygons is enabled.  The initial value is GL_FALSE. See glPolygon↩
Mode."
```

6.3.2.147 gl_polygon_smooth_hint

```
QString GLDescriptions::gl_polygon_smooth_hint = "Data returns one value, a symbolic constant
indicating the mode of the polygon antialiasing hint.  The initial value is GL_DONT_CARE. See
glHint."
```

6.3.2.148 gl_primitive_restart_index

```
QString GLDescriptions::gl_primitive_restart_index = "Data returns one value, the current
primitive restart index.  The initial value is 0.  See glPrimitiveRestartIndex."
```

6.3.2.149 gl_program_binary_formats

```
QString GLDescriptions::gl_program_binary_formats = "Data an array of GL_NUM_PROGRAM_BINARY_↩
FORMATS values, indicating the proram binary formats supported by the implementation."
```

6.3.2.150 gl_program_pipeline_binding

```
QString GLDescriptions::gl_program_pipeline_binding = "Data a single value, the name of the
currently bound program pipeline object, or zero if no program pipeline object is bound.  See
glBindProgramPipeline."
```

6.3.2.151 gl_program_point_size

```
QString GLDescriptions::gl_program_point_size = "Data returns a single boolean value indicating
whether vertex program point size mode is enabled.  If enabled, then the point size is taken
from the shader built-in gl_PointSize.  If disabled, then the point size is taken from the
point state as specified by glPointSize.  The initial value is GL_FALSE."
```

6.3.2.152 gl_provoking_vertex

```
QString GLDescriptions::gl_provoking_vertex = "Data returns one value, the currently selected
provoking vertex convention.  The initial value is GL_LAST_VERTEX_CONVENTION. See glProvoking↩
Vertex."
```

6.3.2.153 gl_read_buffer

```
QString GLDescriptions::gl_read_buffer = "Data returns one value, a symbolic constant indicating
which color buffer is selected for reading.  The initial value is GL_BACK if there is a back
buffer, otherwise it is GL_FRONT. See glReadPixels."
```

6.3.2.154 gl_read_framebuffer_binding

```
QString GLDescriptions::gl_read_framebuffer_binding = "Data returns one value, the name of
the framebuffer object currently bound to the GL_READ_FRAMEBUFFER target.  If the default
framebuffer is bound, this value will be zero.  The initial value is zero.  See glBindFramebuffer."
```

6.3.2.155 gl_renderbuffer_binding

```
QString GLDescriptions::gl_renderbuffer_binding = "Data returns a single value, the name of
the renderbuffer object currently bound to the target GL_RENDERBUFFER. If no renderbuffer
object is bound to this target, 0 is returned.  The initial value is 0.  See glBindRenderbuffer."
```

6.3.2.156 gl_sample_buffers

```
QString GLDescriptions::gl_sample_buffers = "Data returns a single integer value indicating
the number of sample buffers associated with the framebuffer.  See glSampleCoverage."
```

6.3.2.157 gl_sample_coverage_invert

```
QString GLDescriptions::gl_sample_coverage_invert = "Data returns a single boolean value indicating
if the temporary coverage value should be inverted.  See glSampleCoverage."
```

6.3.2.158 gl_sample_coverage_value

```
QString GLDescriptions::gl_sample_coverage_value = "Data returns a single positive floating-point
value indicating the current sample coverage value.  See glSampleCoverage."
```

6.3.2.159 gl_sampler_binding

```
QString GLDescriptions::gl_sampler_binding = "Data returns a single value, the name of the
sampler object currently bound to the active texture unit.  The initial value is 0.  See gl↩
BindSampler."
```

6.3.2.160 gl_samples

```
QString GLDescriptions::gl_samples = "Data returns a single integer value indicating the coverage
mask size.  See glSampleCoverage."
```

6.3.2.161 gl_scissor_box

```
QString GLDescriptions::gl_scissor_box = "Data returns four values:  the x and y window coordinates
of the scissor box, followed by its width and height.  Initially the x and y window coordinates
are both 0 and the width and height are set to the size of the window.  See glScissor."
```

6.3.2.162 gl_scissor_test

```
QString GLDescriptions::gl_scissor_test = "Data returns a single boolean value indicating
whether scissoring is enabled.  The initial value is GL_FALSE. See glScissor."
```

6.3.2.163 gl_shader_compiler

```
QString GLDescriptions::gl_shader_compiler = "Data returns a single boolean value indicating
whether an online shader compiler is present in the implementation.  All desktop OpenGL implementations
must support online shader compilations, and therefore the value of GL_SHADER_COMPILER will
always be GL_TRUE."
```

6.3.2.164 gl_shader_storage_buffer_binding

```
QString GLDescriptions::gl_shader_storage_buffer_binding = "When used with non-indexed variants
of glGet (such as glGetIntegerv), data returns a single value, the name of the buffer object
currently bound to the target GL_SHADER_STORAGE_BUFFER. If no buffer object is bound to this
target, 0 is returned.  When used with indexed variants of glGet (such as glGetIntegeri_v),
data returns a single value, the name of the buffer object bound to the indexed shader storage
buffer binding points.  The initial value is 0 for all targets.  See glBindBuffer, glBind↩
BufferBase, and glBindBufferRange."
```

6.3.2.165 gl_shader_storage_buffer_offset_alignment

```
QString GLDescriptions::gl_shader_storage_buffer_offset_alignment = "Data returns a single
value, the minimum required alignment for shader storage buffer sizes and offset.  The initial
value is 1.  See glShaderStorageBlockBinding."
```

6.3.2.166 gl_shader_storage_buffer_size

```
QString GLDescriptions::gl_shader_storage_buffer_size = "When used with indexed variants of
glGet (such as glGetInteger64i_v), data returns a single value, the size of the binding range
for each indexed shader storage buffer binding.  The initial value is 0 for all bindings.  See
glBindBufferRange."
```

6.3.2.167 gl_shader_storage_buffer_start

```
QString GLDescriptions::gl_shader_storage_buffer_start = "When used with indexed variants
of glGet (such as glGetInteger64i_v), data returns a single value, the start offset of the
binding range for each indexed shader storage buffer binding.  The initial value is 0 for all
bindings.  See glBindBufferRange."
```

6.3.2.168 gl_smooth_line_width_granularity

```
QString GLDescriptions::gl_smooth_line_width_granularity = "Data returns a single value indicating
the level of quantization applied to smooth line width parameters."
```

6.3.2.169 gl_smooth_line_width_range

```
QString GLDescriptions::gl_smooth_line_width_range = "Data returns a pair of values indicating
the range of widths supported for smooth (antialiased) lines.  See glLineWidth."
```

6.3.2.170 gl_stencil_back_fail

```
QString GLDescriptions::gl_stencil_back_fail = "Data returns one value, a symbolic constant
indicating what action is taken for back-facing polygons when the stencil test fails.  The
initial value is GL_KEEP. See glStencilOpSeparate."
```

6.3.2.171 gl_stencil_back_func

```
QString GLDescriptions::gl_stencil_back_func = "Data returns one value, a symbolic constant
indicating what function is used for back-facing polygons to compare the stencil reference
value with the stencil buffer value.  The initial value is GL_ALWAYS. See glStencilFunc↩
Separate."
```

6.3.2.172 gl_stencil_back_pass_depth_fail

```
QString GLDescriptions::gl_stencil_back_pass_depth_fail = "Data returns one value, a symbolic
constant indicating what action is taken for back-facing polygons when the stencil test passes,
but the depth test fails.  The initial value is GL_KEEP. See glStencilOpSeparate."
```

6.3.2.173 gl_stencil_back_pass_depth_pass

```
QString GLDescriptions::gl_stencil_back_pass_depth_pass = "Data returns one value, a symbolic
constant indicating what action is taken for back-facing polygons when the stencil test passes
and the depth test passes.  The initial value is GL_KEEP. See glStencilOpSeparate."
```

6.3.2.174 gl_stencil_back_ref

```
QString GLDescriptions::gl_stencil_back_ref = "Data returns one value, the reference value
that is compared with the contents of the stencil buffer for back-facing polygons.  The initial
value is 0.  See glStencilFuncSeparate."
```

6.3.2.175 gl_stencil_back_value_mask

```
QString GLDescriptions::gl_stencil_back_value_mask = "Data returns one value, the mask that is
used for back-facing polygons to mask both the stencil reference value and the stencil buffer
value before they are compared.  The initial value is all 1's.  See glStencilFuncSeparate."
```

6.3.2.176 gl_stencil_back_writemask

```
QString GLDescriptions::gl_stencil_back_writemask = "Data returns one value, the mask that
controls writing of the stencil bitplanes for back-facing polygons.  The initial value is all
1's.  See glStencilMaskSeparate."
```

6.3.2.177 gl_stencil_clear_value

```
QString GLDescriptions::gl_stencil_clear_value = "Data returns one value, the index to which
the stencil bitplanes are cleared.  The initial value is 0.  See glClearStencil."
```

6.3.2.178 gl_stencil_fail

```
QString GLDescriptions::gl_stencil_fail = "Data returns one value, a symbolic constant indicating
what action is taken when the stencil test fails.  The initial value is GL_KEEP. See gl↩
StencilOp.  This stencil state only affects non-polygons and front-facing polygons.  Back-facing
polygons use separate stencil state.  See glStencilOpSeparate."
```

6.3.2.179 gl_stencil_func

```
QString GLDescriptions::gl_stencil_func = "Data returns one value, a symbolic constant indicating
what function is used to compare the stencil reference value with the stencil buffer value.
The initial value is GL_ALWAYS. See glStencilFunc.  This stencil state only affects non-polygons
and front-facing polygons.  Back-facing polygons use separate stencil state.  See glStencil↩
FuncSeparate."
```

6.3.2.180 gl_stencil_pass_depth_fail

```
QString GLDescriptions::gl_stencil_pass_depth_fail = "Data returns one value, a symbolic constant
indicating what action is taken when the stencil test passes, but the depth test fails.  The
initial value is GL_KEEP. See glStencilOp.  This stencil state only affects non-polygons and
front-facing polygons.  Back-facing polygons use separate stencil state.  See glStencilOp↩
Separate."
```

6.3.2.181 gl_stencil_pass_depth_pass

```
QString GLDescriptions::gl_stencil_pass_depth_pass = "Data returns one value, a symbolic constant
indicating what action is taken when the stencil test passes and the depth test passes.  The
initial value is GL_KEEP. See glStencilOp.  This stencil state only affects non-polygons and
front-facing polygons.  Back-facing polygons use separate stencil state.  See glStencilOp↩
Separate."
```

6.3.2.182 gl_stencil_ref

```
QString GLDescriptions::gl_stencil_ref = "Data returns one value, the reference value that is
compared with the contents of the stencil buffer.  The initial value is 0.  See glStencilFunc.
This stencil state only affects non-polygons and front-facing polygons.  Back-facing polygons
use separate stencil state.  See glStencilFuncSeparate."
```

6.3.2.183 gl_stencil_test

```
QString GLDescriptions::gl_stencil_test = "Data returns a single boolean value indicating
whether stencil testing of fragments is enabled.  The initial value is GL_FALSE. See gl↩
StencilFunc and glStencilOp."
```

6.3.2.184 gl_stencil_value_mask

```
QString GLDescriptions::gl_stencil_value_mask = "Data returns one value, the mask that is
used to mask both the stencil reference value and the stencil buffer value before they are
compared.  The initial value is all 1's.  See glStencilFunc.  This stencil state only affects
non-polygons and front-facing polygons.  Back-facing polygons use separate stencil state.  See
glStencilFuncSeparate."
```

6.3.2.185 gl_stencil_writemask

```
QString GLDescriptions::gl_stencil_writemask = "Data returns one value, the mask that controls
writing of the stencil bitplanes.  The initial value is all 1's.  See glStencilMask.  This
stencil state only affects non-polygons and front-facing polygons.  Back-facing polygons use
separate stencil state.  See glStencilMaskSeparate."
```

6.3.2.186 gl_stereo

```
QString GLDescriptions::gl_stereo = "Data returns a single boolean value indicating whether
stereo buffers (left and right) are supported."
```

6.3.2.187 gl_subpixel_bits

```
QString GLDescriptions::gl_subpixel_bits = "Data returns one value, an estimate of the number
of bits of subpixel resolution that are used to position rasterized geometry in window coordinates.
The value must be at least 4."
```

6.3.2.188 gl_texture_binding_1d

```
QString GLDescriptions::gl_texture_binding_1d = "Data returns a single value, the name of the
texture currently bound to the target GL_TEXTURE_1D. The initial value is 0.  See glBind↩
Texture."
```

6.3.2.189 gl_texture_binding_1d_array

```
QString GLDescriptions::gl_texture_binding_1d_array = "Data returns a single value, the name
of the texture currently bound to the target GL_TEXTURE_1D_ARRAY. The initial value is 0.  See
glBindTexture."
```

6.3.2.190 gl_texture_binding_2d

```
QString GLDescriptions::gl_texture_binding_2d = "Data returns a single value, the name of the
texture currently bound to the target GL_TEXTURE_2D. The initial value is 0.  See glBind↩
Texture."
```

6.3.2.191 gl_texture_binding_2d_array

```
QString GLDescriptions::gl_texture_binding_2d_array = "Data returns a single value, the name
of the texture currently bound to the target GL_TEXTURE_2D_ARRAY. The initial value is 0.  See
glBindTexture."
```

6.3.2.192 gl_texture_binding_2d_multisample

```
QString GLDescriptions::gl_texture_binding_2d_multisample = "Data returns a single value, the
name of the texture currently bound to the target GL_TEXTURE_2D_MULTISAMPLE. The initial value
is 0.  See glBindTexture."
```

6.3.2.193 gl_texture_binding_2d_multisample_array

```
QString GLDescriptions::gl_texture_binding_2d_multisample_array = "Data returns a single value,
the name of the texture currently bound to the target GL_TEXTURE_2D_MULTISAMPLE_ARRAY. The
initial value is 0.  See glBindTexture."
```

6.3.2.194 gl_texture_binding_3d

```
QString GLDescriptions::gl_texture_binding_3d = "Data returns a single value, the name of the
texture currently bound to the target GL_TEXTURE_3D. The initial value is 0.  See glBind↩
Texture."
```

6.3.2.195 gl_texture_binding_buffer

```
QString GLDescriptions::gl_texture_binding_buffer = "Data returns a single value, the name of
the texture currently bound to the target GL_TEXTURE_BUFFER. The initial value is 0.  See gl↩
BindTexture."
```

6.3.2.196 gl_texture_binding_cube_map

```
QString GLDescriptions::gl_texture_binding_cube_map = "Data returns a single value, the name
of the texture currently bound to the target GL_TEXTURE_CUBE_MAP. The initial value is 0.  See
glBindTexture."
```

6.3.2.197 gl_texture_binding_rectangle

```
QString GLDescriptions::gl_texture_binding_rectangle = "Data returns a single value, the name
of the texture currently bound to the target GL_TEXTURE_RECTANGLE. The initial value is 0.
See glBindTexture."
```

6.3.2.198 gl_texture_buffer_offset_alignment

```
QString GLDescriptions::gl_texture_buffer_offset_alignment = "Data returns a single value, the
minimum required alignment for texture buffer sizes and offset.  The initial value is 1.  See
glUniformBlockBinding."
```

6.3.2.199 gl_texture_compression_hint

```
QString GLDescriptions::gl_texture_compression_hint = "Data returns a single value indicating
the mode of the texture compression hint.  The initial value is GL_DONT_CARE."
```

6.3.2.200 gl_timestamp

```
QString GLDescriptions::gl_timestamp = "Data returns a single value, the 64-bit value of the
current GL time.  See glQueryCounter."
```

6.3.2.201 gl_transform_feedback_buffer_binding

```
QString GLDescriptions::gl_transform_feedback_buffer_binding = "When used with non-indexed
variants of glGet (such as glGetIntegerv), data returns a single value, the name of the buffer
object currently bound to the target GL_TRANSFORM_FEEDBACK_BUFFER. If no buffer object is
bound to this target, 0 is returned.  When used with indexed variants of glGet (such as gl↩
GetIntegeri_v), data returns a single value, the name of the buffer object bound to the indexed
transform feedback attribute stream.  The initial value is 0 for all targets.  See glBind↩
Buffer, glBindBufferBase, and glBindBufferRange."
```

6.3.2.202 gl_transform_feedback_buffer_size

```
QString GLDescriptions::gl_transform_feedback_buffer_size = "When used with indexed variants
of glGet (such as glGetInteger64i_v), data returns a single value, the size of the binding
range for each transform feedback attribute stream.  The initial value is 0 for all streams.
See glBindBufferRange."
```

6.3.2.203 gl_transform_feedback_buffer_start

```
QString GLDescriptions::gl_transform_feedback_buffer_start = "When used with indexed variants
of glGet (such as glGetInteger64i_v), data returns a single value, the start offset of the
binding range for each transform feedback attribute stream.  The initial value is 0 for all
streams.  See glBindBufferRange."
```

6.3.2.204 gl_uniform_buffer_binding

```
QString GLDescriptions::gl_uniform_buffer_binding = "When used with non-indexed variants of
glGet (such as glGetIntegerv), data returns a single value, the name of the buffer object
currently bound to the target GL_UNIFORM_BUFFER. If no buffer object is bound to this target,
0 is returned.  When used with indexed variants of glGet (such as glGetIntegeri_v), data returns
a single value, the name of the buffer object bound to the indexed uniform buffer binding
point.  The initial value is 0 for all targets.  See glBindBuffer, glBindBufferBase, and gl↩
BindBufferRange."
```

6.3.2.205 gl_uniform_buffer_offset_alignment

```
QString GLDescriptions::gl_uniform_buffer_offset_alignment = "Data returns a single value, the
minimum required alignment for uniform buffer sizes and offset.  The initial value is 1.  See
glUniformBlockBinding."
```

6.3.2.206 gl_uniform_buffer_size

```
QString GLDescriptions::gl_uniform_buffer_size = "When used with indexed variants of glGet
(such as glGetInteger64i_v), data returns a single value, the size of the binding range for
each indexed uniform buffer binding.  The initial value is 0 for all bindings.  See glBind↩
BufferRange."
```

6.3.2.207 gl_uniform_buffer_start

```
QString GLDescriptions::gl_uniform_buffer_start = "When used with indexed variants of gl↩
Get (such as glGetInteger64i_v), data returns a single value, the start offset of the binding
range for each indexed uniform buffer binding.  The initial value is 0 for all bindings.  See
glBindBufferRange."
```

6.3.2.208 gl_unpack_alignment

```
QString GLDescriptions::gl_unpack_alignment = "Data returns one value, the byte alignment used
for reading pixel data from memory.  The initial value is 4.  See glPixelStore."
```

6.3.2.209 gl_unpack_image_height

```
QString GLDescriptions::gl_unpack_image_height = "Data returns one value, the image height
used for reading pixel data from memory.  The initial is 0.  See glPixelStore."
```

6.3.2.210 gl_unpack_lsb_first

```
QString GLDescriptions::gl_unpack_lsb_first = "Data returns a single boolean value indicating
whether single-bit pixels being read from memory are read first from the least significant bit
of each unsigned byte.  The initial value is GL_FALSE. See glPixelStore."
```

6.3.2.211 gl_unpack_row_length

```
QString GLDescriptions::gl_unpack_row_length = "Data returns one value, the row length used
for reading pixel data from memory.  The initial value is 0.  See glPixelStore."
```

6.3.2.212 gl_unpack_skip_images

```
QString GLDescriptions::gl_unpack_skip_images = "Data returns one value, the number of pixel
images skipped before the first pixel is read from memory.  The initial value is 0.  See gl↩
PixelStore."
```

6.3.2.213 gl_unpack_skip_pixels

```
QString GLDescriptions::gl_unpack_skip_pixels = "Data returns one value, the number of pixel
locations skipped before the first pixel is read from memory.  The initial value is 0.  See
glPixelStore."
```

6.3.2.214 gl_unpack_skip_rows

```
QString GLDescriptions::gl_unpack_skip_rows = "Data returns one value, the number of rows of
pixel locations skipped before the first pixel is read from memory.  The initial value is 0.
See glPixelStore."
```

6.3.2.215 gl_unpack_swap_bytes

```
QString GLDescriptions::gl_unpack_swap_bytes = "Data returns a single boolean value indicating
whether the bytes of two-byte and four-byte pixel indices and components are swapped after
being read from memory.  The initial value is GL_FALSE. See glPixelStore."
```

6.3.2.216 gl_vertex_array_binding

```
QString GLDescriptions::gl_vertex_array_binding = "Data returns a single value, the name of
the vertex array object currently bound to the context.  If no vertex array object is bound to
the context, 0 is returned.  The initial value is 0.  See glBindVertexArray."
```

6.3.2.217 gl_vertex_binding_divisor

```
QString GLDescriptions::gl_vertex_binding_divisor = "Accepted by the indexed forms.  data
returns a single integer value representing the instance step divisor of the first element in
the bound buffer's data store for vertex attribute bound to index."
```

6.3.2.218 gl_vertex_binding_offset

```
QString GLDescriptions::gl_vertex_binding_offset = "Accepted by the indexed forms.  data returns
a single integer value representing the byte offset of the first element in the bound buffer's
data store for vertex attribute bound to index."
```

6.3.2.219 gl_vertex_binding_stride

```
QString GLDescriptions::gl_vertex_binding_stride = "Accepted by the indexed forms.  data returns
a single integer value representing the byte offset between the start of each element in the
bound buffer's data store for vertex attribute bound to index."
```

6.3.2.220 gl_viewport

```
QString GLDescriptions::gl_viewport = "When used with non-indexed variants of glGet (such as
glGetIntegerv), data returns four values:  the x and y window coordinates of the viewport,
followed by its width and height.  Initially the x and y window coordinates are both set to 0,
and the width and height are set to the width and height of the window into which the GL will
do its rendering.  See glViewport."
```

6.3.2.221 gl_viewport_bounds_range

```
QString GLDescriptions::gl_viewport_bounds_range = "Data returns two values, the minimum and
maximum viewport bounds range.  The minimum range should be at least [-32768, 32767]."
```

6.3.2.222 gl_viewport_index_provoking_vertex

```
QString GLDescriptions::gl_viewport_index_provoking_vertex = "Data returns one value, the
implementation dependent specifc vertex of a primitive that is used to select the viewport
index.  If the value returned is equivalent to GL_PROVOKING_VERTEX, then the vertex selection
follows the convention specified by glProvokingVertex.  If the value returned is equivalent
to GL_FIRST_VERTEX_CONVENTION, then the selection is always taken from the first vertex in
the primitive.  If the value returned is equivalent to GL_LAST_VERTEX_CONVENTION, then the
selection is always taken from the last vertex in the primitive.  If the value returned is
equivalent to GL_UNDEFINED_VERTEX, then the selection is not guaranteed to be taken from any
specific vertex in the primitive."
```

6.3.2.223 gl_viewport_subpixel_bits

```
QString GLDescriptions::gl_viewport_subpixel_bits = "Data returns a single value, the number
of bits of sub-pixel precision which the GL uses to interpret the floating point viewport
bounds.  The minimum value is 0."
```

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

- src/gldescriptions.hpp

6.4 gpu::GLInfo Struct Reference

GLInfo struct used to store the GPU and OpenGL data. Structure containing the OpenGL information.

```
#include <GLInfo.hpp>
```

Public Attributes

- int gl_active_texture = -1
- int gl_aliased_line_width_range = -1
- int gl_array_buffer_binding = -1
- int gl_blend = -1
- int gl_blend_color = -1
- int gl_blend_dst_alpha = -1
- int gl_blend_dst_rgb = -1
- int gl_blend_equation_rgb = -1
- int gl_blend_equation_alpha = -1
- int gl_blend_src_alpha = -1
- int gl_blend_src_rgb = -1
- int gl_color_clear_value = -1
- int gl_color_logic_op = -1
- int gl_color_writemask = -1
- int gl_compressed_texture_formats = -1
- int gl_max_compute_shader_storage_blocks = -1
- int gl_max_combined_shader_storage_blocks = -1
- int gl_max_compute_uniform_blocks = -1
- int gl_max_compute_texture_image_units = -1
- int gl_max_compute_uniform_components = -1
- int gl_max_compute_atomic_counters = -1
- int gl_max_compute_atomic_counter_buffers = -1
- int gl_max_combined_compute_uniform_components = -1
- int gl_max_compute_work_group_count = -1
- int gl_max_compute_work_group_size = -1
- int gl_dispatch_indirect_buffer_binding = -1
- int gl_max_debug_group_stack_depth = -1
- int gl_debug_group_stack_depth = -1
- int gl_context_flags = -1
- int gl_cull_face = -1
- int gl_cull_face_mode = -1
- int gl_current_program = -1
- int gl_depth_clear_value = -1
- int gl_depth_func = -1
- int gl_depth_range = -1
- int gl_depth_test = -1
- int gl_depth_writemask = -1
- int gl_dither = -1
- int gl_doublebuffer = -1
- int gl_draw_buffer = -1
- int gl_draw_framebuffer_binding = -1
- int gl_read_framebuffer_binding = -1
- int gl_element_array_buffer_binding = -1
- int gl_fragment_shader_derivative_hint = -1
- int gl_implementation_color_read_format = -1
- int gl_implementation_color_read_type = -1
- int gl_line_smooth = -1
- int gl_line_smooth_hint = -1
- int gl_line_width = -1
- int gl_layer_provoking_vertex = -1
- int gl_logic_op_mode = -1
- int gl_major_version = -1
- int gl_max_3d_texture_size = -1

- int gl_max_array_texture_layers = -1
- int gl_max_clip_distances = -1
- int gl_max_color_texture_samples = -1
- int gl_max_combined_atomic_counters = -1
- int gl_max_combined_fragment_uniform_components = -1
- int gl_max_combined_geometry_uniform_components = -1
- int gl_max_combined_texture_image_units = -1
- int gl_max_combined_uniform_blocks = -1
- int gl_max_combined_vertex_uniform_components = -1
- int gl_max_cube_map_texture_size = -1
- int gl_max_depth_texture_samples = -1
- int gl_max_draw_buffers = -1
- int gl_max_dual_source_draw_buffers = -1
- int gl_max_elements_indices = -1
- int gl_max_elements_vertices = -1
- int gl_max_fragment_atomic_counters = -1
- int gl_max_fragment_shader_storage_blocks = -1
- int gl_max_fragment_input_components = -1
- int gl_max_fragment_uniform_components = -1
- int gl_max_fragment_uniform_vectors = -1
- int gl_max_fragment_uniform_blocks = -1
- int gl_max_framebuffer_width = -1
- int gl_max_framebuffer_height = -1
- int gl_max_framebuffer_layers = -1
- int gl_max_framebuffer_samples = -1
- int gl_max_geometry_atomic_counters = -1
- int gl_max_geometry_shader_storage_blocks = -1
- int gl_max_geometry_input_components = -1
- int gl_max_geometry_output_components = -1
- int gl_max_geometry_texture_image_units = -1
- int gl_max_geometry_uniform_blocks = -1
- int gl_max_geometry_uniform_components = -1
- int gl_max_integer_samples = -1
- int gl_min_map_buffer_alignment = -1
- int gl_max_label_length = -1
- int gl_max_program_texel_offset = -1
- int gl_min_program_texel_offset = -1
- int gl_max_rectangle_texture_size = -1
- int gl_max_renderbuffer_size = -1
- int gl_max_sample_mask_words = -1
- int gl_max_server_wait_timeout = -1
- int gl_max_shader_storage_buffer_bindings = -1
- int gl_max_tess_control_atomic_counters = -1
- int gl_max_tess_evaluation_atomic_counters = -1
- int gl_max_tess_control_shader_storage_blocks = -1
- int gl_max_tess_evaluation_shader_storage_blocks = -1
- int gl_max_texture_buffer_size = -1
- int gl_max_texture_image_units = -1
- int gl_max_texture_lod_bias = -1
- int gl_max_texture_size = -1
- int gl_max_uniform_buffer_bindings = -1
- int gl_max_uniform_block_size = -1
- int gl_max_uniform_locations = -1
- int gl_max_varying_components = -1
- int gl_max_varying_vectors = -1

- int gl_max_varying_floats = -1
- int gl_max_vertex_atomic_counters = -1
- int gl_max_vertex_attribs = -1
- int gl_max_vertex_shader_storage_blocks = -1
- int gl_max_vertex_texture_image_units = -1
- int gl_max_vertex_uniform_components = -1
- int gl_max_vertex_uniform_vectors = -1
- int gl_max_vertex_output_components = -1
- int gl_max_vertex_uniform_blocks = -1
- int gl_max_viewport_dims = -1
- int gl_max_viewports = -1
- int gl_minor_version = -1
- int gl_num_compressed_texture_formats = -1
- int gl_num_extensions = -1
- int gl_num_program_binary_formats = -1
- int gl_num_shader_binary_formats = -1
- int gl_pack_alignment = -1
- int gl_pack_image_height = -1
- int gl_pack_lsb_first = -1
- int gl_pack_row_length = -1
- int gl_pack_skip_images = -1
- int gl_pack_skip_pixels = -1
- int gl_pack_skip_rows = -1
- int gl_pack_swap_bytes = -1
- int gl_pixel_pack_buffer_binding = -1
- int gl_pixel_unpack_buffer_binding = -1
- int gl_point_fade_threshold_size = -1
- int gl_primitive_restart_index = -1
- int gl_program_binary_formats = -1
- int gl_program_pipeline_binding = -1
- int gl_program_point_size = -1
- int gl_provoking_vertex = -1
- int gl_point_size = -1
- int gl_point_size_granularity = -1
- int gl_point_size_range = -1
- int gl_polygon_offset_factor = -1
- int gl_polygon_offset_units = -1
- int gl_polygon_offset_fill = -1
- int gl_polygon_offset_line = -1
- int gl_polygon_offset_point = -1
- int gl_polygon_smooth = -1
- int gl_polygon_smooth_hint = -1
- int gl_read_buffer = -1
- int gl_renderbuffer_binding = -1
- int gl_sample_buffers = -1
- int gl_sample_coverage_value = -1
- int gl_sample_coverage_invert = -1
- int gl_sampler_binding = -1
- int gl_samples = -1
- int gl_scissor_box = -1
- int gl_scissor_test = -1
- int gl_shader_compiler = -1
- int gl_shader_storage_buffer_binding = -1
- int gl_shader_storage_buffer_offset_alignment = -1
- int gl_shader_storage_buffer_start = -1

- int gl_shader_storage_buffer_size = -1
- int gl_smooth_line_width_range = -1
- int gl_smooth_line_width_granularity = -1
- int gl_stencil_back_fail = -1
- int gl_stencil_back_func = -1
- int gl_stencil_back_pass_depth_fail = -1
- int gl_stencil_back_pass_depth_pass = -1
- int gl_stencil_back_ref = -1
- int gl_stencil_back_value_mask = -1
- int gl_stencil_back_writemask = -1
- int gl_stencil_clear_value = -1
- int gl_stencil_fail = -1
- int gl_stencil_func = -1
- int gl_stencil_pass_depth_fail = -1
- int gl_stencil_pass_depth_pass = -1
- int gl_stencil_ref = -1
- int gl_stencil_test = -1
- int gl_stencil_value_mask = -1
- int gl_stencil_writemask = -1
- int gl_stereo = -1
- int gl_subpixel_bits = -1
- int gl_texture_binding_1d = -1
- int gl_texture_binding_1d_array = -1
- int gl_texture_binding_2d = -1
- int gl_texture_binding_2d_array = -1
- int gl_texture_binding_2d_multisample = -1
- int gl_texture_binding_2d_multisample_array = -1
- int gl_texture_binding_3d = -1
- int gl_texture_binding_buffer = -1
- int gl_texture_binding_cube_map = -1
- int gl_texture_binding_rectangle = -1
- int gl_texture_compression_hint = -1
- int gl_texture_buffer_offset_alignment = -1
- int gl_timestamp = -1
- int gl_transform_feedback_buffer_binding = -1
- int gl_transform_feedback_buffer_start = -1
- int gl_transform_feedback_buffer_size = -1
- int gl_uniform_buffer_binding = -1
- int gl_uniform_buffer_offset_alignment = -1
- int gl_uniform_buffer_size = -1
- int gl_uniform_buffer_start = -1
- int gl_unpack_alignment = -1
- int gl_unpack_image_height = -1
- int gl_unpack_lsb_first = -1
- int gl_unpack_row_length = -1
- int gl_unpack_skip_images = -1
- int gl_unpack_skip_pixels = -1
- int gl_unpack_skip_rows = -1
- int gl_unpack_swap_bytes = -1
- int gl_vertex_array_binding = -1
- int gl_vertex_binding_divisor = -1
- int gl_vertex_binding_offset = -1
- int gl_vertex_binding_stride = -1
- int gl_max_vertex_attrib_relative_offset = -1
- int gl_max_vertex_attrib_bindings = -1

- int gl_viewport = -1
- int gl_viewport_bounds_range = -1
- int gl_viewport_index_provoking_vertex = -1
- int gl_viewport_subpixel_bits = -1
- int gl_max_element_index = -1

6.4.1 Detailed Description

GLInfo struct used to store the GPU and OpenGL data. Structure containing the OpenGL information.

6.4.2 Member Data Documentation

6.4.2.1 gl_active_texture

```
int gpu::GLInfo::gl_active_texture = -1
```

6.4.2.2 gl_aliased_line_width_range

```
int gpu::GLInfo::gl_aliased_line_width_range = -1
```

6.4.2.3 gl_array_buffer_binding

```
int gpu::GLInfo::gl_array_buffer_binding = -1
```

6.4.2.4 gl_blend

```
int gpu::GLInfo::gl_blend = -1
```

6.4.2.5 gl_blend_color

```
int gpu::GLInfo::gl_blend_color = -1
```

6.4.2.6 gl_blend_dst_alpha

```
int gpu::GLInfo::gl_blend_dst_alpha = -1
```

6.4.2.7 gl_blend_dst_rgb

```
int gpu::GLInfo::gl_blend_dst_rgb = -1
```

6.4.2.8 gl_blend_equation_alpha

```
int gpu::GLInfo::gl_blend_equation_alpha = -1
```

6.4.2.9 gl_blend_equation_rgb

```
int gpu::GLInfo::gl_blend_equation_rgb = -1
```

6.4.2.10 gl_blend_src_alpha

```
int gpu::GLInfo::gl_blend_src_alpha = -1
```

6.4.2.11 gl_blend_src_rgb

```
int gpu::GLInfo::gl_blend_src_rgb = -1
```

6.4.2.12 gl_color_clear_value

```
int gpu::GLInfo::gl_color_clear_value = -1
```

6.4.2.13 gl_color_logic_op

```
int gpu::GLInfo::gl_color_logic_op = -1
```

6.4.2.14 gl_color_writemask

```
int gpu::GLInfo::gl_color_writemask = -1
```

6.4.2.15 gl_compressed_texture_formats

```
int gpu::GLInfo::gl_compressed_texture_formats = -1
```

6.4.2.16 gl_context_flags

```
int gpu::GLInfo::gl_context_flags = -1
```

6.4.2.17 gl_cull_face

```
int gpu::GLInfo::gl_cull_face = -1
```

6.4.2.18 gl_cull_face_mode

```
int gpu::GLInfo::gl_cull_face_mode = -1
```

6.4.2.19 gl_current_program

```
int gpu::GLInfo::gl_current_program = -1
```

6.4.2.20 gl_debug_group_stack_depth

```
int gpu::GLInfo::gl_debug_group_stack_depth = -1
```

6.4.2.21 gl_depth_clear_value

```
int gpu::GLInfo::gl_depth_clear_value = -1
```

6.4.2.22 gl_depth_func

```
int gpu::GLInfo::gl_depth_func = -1
```

6.4.2.23 gl_depth_range

```
int gpu::GLInfo::gl_depth_range = -1
```

6.4.2.24 gl_depth_test

```
int gpu::GLInfo::gl_depth_test = -1
```

6.4.2.25 gl_depth_writemask

```
int gpu::GLInfo::gl_depth_writemask = -1
```

6.4.2.26 gl_dispatch_indirect_buffer_binding

```
int gpu::GLInfo::gl_dispatch_indirect_buffer_binding = -1
```

6.4.2.27 gl_dither

```
int gpu::GLInfo::gl_dither = -1
```

6.4.2.28 gl_doublebuffer

```
int gpu::GLInfo::gl_doublebuffer = -1
```

6.4.2.29 gl_draw_buffer

```
int gpu::GLInfo::gl_draw_buffer = -1
```

6.4.2.30 gl_draw_framebuffer_binding

```
int gpu::GLInfo::gl_draw_framebuffer_binding = -1
```

6.4.2.31 gl_element_array_buffer_binding

```
int gpu::GLInfo::gl_element_array_buffer_binding = -1
```

6.4.2.32 gl_fragment_shader_derivative_hint

```
int gpu::GLInfo::gl_fragment_shader_derivative_hint = -1
```

6.4.2.33 gl_implementation_color_read_format

```
int gpu::GLInfo::gl_implementation_color_read_format = -1
```

6.4.2.34 gl_implementation_color_read_type

```
int gpu::GLInfo::gl_implementation_color_read_type = -1
```

6.4.2.35 gl_layer_provoking_vertex

```
int gpu::GLInfo::gl_layer_provoking_vertex = -1
```

6.4.2.36 gl_line_smooth

```
int gpu::GLInfo::gl_line_smooth = -1
```

6.4.2.37 gl_line_smooth_hint

```
int gpu::GLInfo::gl_line_smooth_hint = -1
```

6.4.2.38 gl_line_width

```
int gpu::GLInfo::gl_line_width = -1
```

6.4.2.39 gl_logic_op_mode

```
int gpu::GLInfo::gl_logic_op_mode = -1
```

6.4.2.40 gl_major_version

```
int gpu::GLInfo::gl_major_version = -1
```

6.4.2.41 gl_max_3d_texture_size

```
int gpu::GLInfo::gl_max_3d_texture_size = -1
```

6.4.2.42 gl_max_array_texture_layers

```
int gpu::GLInfo::gl_max_array_texture_layers = -1
```

6.4.2.43 gl_max_clip_distances

```
int gpu::GLInfo::gl_max_clip_distances = -1
```

6.4.2.44 gl_max_color_texture_samples

```
int gpu::GLInfo::gl_max_color_texture_samples = -1
```

6.4.2.45 gl_max_combined_atomic_counters

```
int gpu::GLInfo::gl_max_combined_atomic_counters = -1
```

6.4.2.46 gl_max_combined_compute_uniform_components

```
int gpu::GLInfo::gl_max_combined_compute_uniform_components = -1
```

6.4.2.47 gl_max_combined_fragment_uniform_components

```
int gpu::GLInfo::gl_max_combined_fragment_uniform_components = -1
```

6.4.2.48 gl_max_combined_geometry_uniform_components

```
int gpu::GLInfo::gl_max_combined_geometry_uniform_components = -1
```

6.4.2.49 gl_max_combined_shader_storage_blocks

```
int gpu::GLInfo::gl_max_combined_shader_storage_blocks = -1
```

6.4.2.50 gl_max_combined_texture_image_units

```
int gpu::GLInfo::gl_max_combined_texture_image_units = -1
```

6.4.2.51 gl_max_combined_uniform_blocks

```
int gpu::GLInfo::gl_max_combined_uniform_blocks = -1
```

6.4.2.52 gl_max_combined_vertex_uniform_components

```
int gpu::GLInfo::gl_max_combined_vertex_uniform_components = -1
```

6.4.2.53 gl_max_compute_atomic_counter_buffers

```
int gpu::GLInfo::gl_max_compute_atomic_counter_buffers = -1
```

6.4.2.54 gl_max_compute_atomic_counters

```
int gpu::GLInfo::gl_max_compute_atomic_counters = -1
```

6.4.2.55 gl_max_compute_shader_storage_blocks

```
int gpu::GLInfo::gl_max_compute_shader_storage_blocks = -1
```

6.4.2.56 gl_max_compute_texture_image_units

```
int gpu::GLInfo::gl_max_compute_texture_image_units = -1
```

6.4.2.57 gl_max_compute_uniform_blocks

```
int gpu::GLInfo::gl_max_compute_uniform_blocks = -1
```

6.4.2.58 gl_max_compute_uniform_components

```
int gpu::GLInfo::gl_max_compute_uniform_components = -1
```

6.4.2.59 gl_max_compute_work_group_count

```
int gpu::GLInfo::gl_max_compute_work_group_count = -1
```

6.4.2.60 gl_max_compute_work_group_size

```
int gpu::GLInfo::gl_max_compute_work_group_size = -1
```

6.4.2.61 gl_max_cube_map_texture_size

```
int gpu::GLInfo::gl_max_cube_map_texture_size = -1
```

6.4.2.62 gl_max_debug_group_stack_depth

```
int gpu::GLInfo::gl_max_debug_group_stack_depth = -1
```

6.4.2.63 gl_max_depth_texture_samples

```
int gpu::GLInfo::gl_max_depth_texture_samples = -1
```

6.4.2.64 gl_max_draw_buffers

```
int gpu::GLInfo::gl_max_draw_buffers = -1
```

6.4.2.65 gl_max_dual_source_draw_buffers

```
int gpu::GLInfo::gl_max_dual_source_draw_buffers = -1
```

6.4.2.66 gl_max_element_index

```
int gpu::GLInfo::gl_max_element_index = -1
```

6.4.2.67 gl_max_elements_indices

```
int gpu::GLInfo::gl_max_elements_indices = -1
```

6.4.2.68 gl_max_elements_vertices

```
int gpu::GLInfo::gl_max_elements_vertices = -1
```

6.4.2.69 gl_max_fragment_atomic_counters

```
int gpu::GLInfo::gl_max_fragment_atomic_counters = -1
```

6.4.2.70 gl_max_fragment_input_components

```
int gpu::GLInfo::gl_max_fragment_input_components = -1
```

6.4.2.71 gl_max_fragment_shader_storage_blocks

```
int gpu::GLInfo::gl_max_fragment_shader_storage_blocks = -1
```

6.4.2.72 gl_max_fragment_uniform_blocks

```
int gpu::GLInfo::gl_max_fragment_uniform_blocks = -1
```

6.4.2.73 gl_max_fragment_uniform_components

```
int gpu::GLInfo::gl_max_fragment_uniform_components = -1
```

6.4.2.74 gl_max_fragment_uniform_vectors

```
int gpu::GLInfo::gl_max_fragment_uniform_vectors = -1
```

6.4.2.75 gl_max_framebuffer_height

```
int gpu::GLInfo::gl_max_framebuffer_height = -1
```

6.4.2.76 gl_max_framebuffer_layers

```
int gpu::GLInfo::gl_max_framebuffer_layers = -1
```

6.4.2.77 gl_max_framebuffer_samples

```
int gpu::GLInfo::gl_max_framebuffer_samples = -1
```

6.4.2.78 gl_max_framebuffer_width

```
int gpu::GLInfo::gl_max_framebuffer_width = -1
```

6.4.2.79 gl_max_geometry_atomic_counters

```
int gpu::GLInfo::gl_max_geometry_atomic_counters = -1
```

6.4.2.80 gl_max_geometry_input_components

```
int gpu::GLInfo::gl_max_geometry_input_components = -1
```

6.4.2.81 gl_max_geometry_output_components

```
int gpu::GLInfo::gl_max_geometry_output_components = -1
```

6.4.2.82 gl_max_geometry_shader_storage_blocks

```
int gpu::GLInfo::gl_max_geometry_shader_storage_blocks = -1
```

6.4.2.83 gl_max_geometry_texture_image_units

```
int gpu::GLInfo::gl_max_geometry_texture_image_units = -1
```

6.4.2.84 gl_max_geometry_uniform_blocks

```
int gpu::GLInfo::gl_max_geometry_uniform_blocks = -1
```

6.4.2.85 gl_max_geometry_uniform_components

```
int gpu::GLInfo::gl_max_geometry_uniform_components = -1
```

6.4.2.86 gl_max_integer_samples

```
int gpu::GLInfo::gl_max_integer_samples = -1
```

6.4.2.87 gl_max_label_length

```
int gpu::GLInfo::gl_max_label_length = -1
```

6.4.2.88 gl_max_program_texel_offset

```
int gpu::GLInfo::gl_max_program_texel_offset = -1
```

6.4.2.89 gl_max_rectangle_texture_size

```
int gpu::GLInfo::gl_max_rectangle_texture_size = -1
```

6.4.2.90 gl_max_renderbuffer_size

```
int gpu::GLInfo::gl_max_renderbuffer_size = -1
```

6.4.2.91 gl_max_sample_mask_words

```
int gpu::GLInfo::gl_max_sample_mask_words = -1
```

6.4.2.92 gl_max_server_wait_timeout

```
int gpu::GLInfo::gl_max_server_wait_timeout = -1
```

6.4.2.93 gl_max_shader_storage_buffer_bindings

```
int gpu::GLInfo::gl_max_shader_storage_buffer_bindings = -1
```

6.4.2.94 gl_max_tess_control_atomic_counters

```
int gpu::GLInfo::gl_max_tess_control_atomic_counters = -1
```

6.4.2.95 gl_max_tess_control_shader_storage_blocks

```
int gpu::GLInfo::gl_max_tess_control_shader_storage_blocks = -1
```

6.4.2.96 gl_max_tess_evaluation_atomic_counters

```
int gpu::GLInfo::gl_max_tess_evaluation_atomic_counters = -1
```

6.4.2.97 gl_max_tess_evaluation_shader_storage_blocks

```
int gpu::GLInfo::gl_max_tess_evaluation_shader_storage_blocks = -1
```

6.4.2.98 gl_max_texture_buffer_size

```
int gpu::GLInfo::gl_max_texture_buffer_size = -1
```

6.4.2.99 gl_max_texture_image_units

```
int gpu::GLInfo::gl_max_texture_image_units = -1
```

6.4.2.100 gl_max_texture_lod_bias

```
int gpu::GLInfo::gl_max_texture_lod_bias = -1
```

6.4.2.101 gl_max_texture_size

```
int gpu::GLInfo::gl_max_texture_size = -1
```

6.4.2.102 gl_max_uniform_block_size

```
int gpu::GLInfo::gl_max_uniform_block_size = -1
```

6.4.2.103 gl_max_uniform_buffer_bindings

```
int gpu::GLInfo::gl_max_uniform_buffer_bindings = -1
```

6.4.2.104 gl_max_uniform_locations

```
int gpu::GLInfo::gl_max_uniform_locations = -1
```

6.4.2.105 gl_max_varying_components

```
int gpu::GLInfo::gl_max_varying_components = -1
```

6.4.2.106 gl_max_varying_floats

```
int gpu::GLInfo::gl_max_varying_floats = -1
```

6.4.2.107 gl_max_varying_vectors

```
int gpu::GLInfo::gl_max_varying_vectors = -1
```

6.4.2.108 gl_max_vertex_atomic_counters

```
int gpu::GLInfo::gl_max_vertex_atomic_counters = -1
```

6.4.2.109 gl_max_vertex_attrib_bindings

```
int gpu::GLInfo::gl_max_vertex_attrib_bindings = -1
```

6.4.2.110 gl_max_vertex_attrib_relative_offset

```
int gpu::GLInfo::gl_max_vertex_attrib_relative_offset = -1
```

6.4.2.111 gl_max_vertex_attribs

```
int gpu::GLInfo::gl_max_vertex_attribs = -1
```

6.4.2.112 gl_max_vertex_output_components

```
int gpu::GLInfo::gl_max_vertex_output_components = -1
```

6.4.2.113 gl_max_vertex_shader_storage_blocks

```
int gpu::GLInfo::gl_max_vertex_shader_storage_blocks = -1
```

6.4.2.114 gl_max_vertex_texture_image_units

```
int gpu::GLInfo::gl_max_vertex_texture_image_units = -1
```

6.4.2.115 gl_max_vertex_uniform_blocks

```
int gpu::GLInfo::gl_max_vertex_uniform_blocks = -1
```

6.4.2.116 gl_max_vertex_uniform_components

```
int gpu::GLInfo::gl_max_vertex_uniform_components = -1
```

6.4.2.117 gl_max_vertex_uniform_vectors

```
int gpu::GLInfo::gl_max_vertex_uniform_vectors = -1
```

6.4.2.118 gl_max_viewport_dims

```
int gpu::GLInfo::gl_max_viewport_dims = -1
```

6.4.2.119 gl_max_viewports

```
int gpu::GLInfo::gl_max_viewports = -1
```

6.4.2.120 gl_min_map_buffer_alignment

```
int gpu::GLInfo::gl_min_map_buffer_alignment = -1
```

6.4.2.121 gl_min_program_texel_offset

```
int gpu::GLInfo::gl_min_program_texel_offset = -1
```

6.4.2.122 gl_minor_version

```
int gpu::GLInfo::gl_minor_version = -1
```

6.4.2.123 gl_num_compressed_texture_formats

```
int gpu::GLInfo::gl_num_compressed_texture_formats = -1
```

6.4.2.124 gl_num_extensions

```
int gpu::GLInfo::gl_num_extensions = -1
```

6.4.2.125 gl_num_program_binary_formats

```
int gpu::GLInfo::gl_num_program_binary_formats = -1
```

6.4.2.126 gl_num_shader_binary_formats

```
int gpu::GLInfo::gl_num_shader_binary_formats = -1
```

6.4.2.127 gl_pack_alignment

```
int gpu::GLInfo::gl_pack_alignment = -1
```

6.4.2.128 gl_pack_image_height

```
int gpu::GLInfo::gl_pack_image_height = -1
```

6.4.2.129 gl_pack_lsb_first

```
int gpu::GLInfo::gl_pack_lsb_first = -1
```

6.4.2.130 gl_pack_row_length

```
int gpu::GLInfo::gl_pack_row_length = -1
```

6.4.2.131 gl_pack_skip_images

```
int gpu::GLInfo::gl_pack_skip_images = -1
```

6.4.2.132 gl_pack_skip_pixels

```
int gpu::GLInfo::gl_pack_skip_pixels = -1
```

6.4.2.133 gl_pack_skip_rows

```
int gpu::GLInfo::gl_pack_skip_rows = -1
```

6.4.2.134 gl_pack_swap_bytes

```
int gpu::GLInfo::gl_pack_swap_bytes = -1
```

6.4.2.135 gl_pixel_pack_buffer_binding

```
int gpu::GLInfo::gl_pixel_pack_buffer_binding = -1
```

6.4.2.136 gl_pixel_unpack_buffer_binding

```
int gpu::GLInfo::gl_pixel_unpack_buffer_binding = -1
```

6.4.2.137 gl_point_fade_threshold_size

```
int gpu::GLInfo::gl_point_fade_threshold_size = -1
```

6.4.2.138 gl_point_size

```
int gpu::GLInfo::gl_point_size = -1
```

6.4.2.139 gl_point_size_granularity

```
int gpu::GLInfo::gl_point_size_granularity = -1
```

6.4.2.140 gl_point_size_range

```
int gpu::GLInfo::gl_point_size_range = -1
```

6.4.2.141 gl_polygon_offset_factor

```
int gpu::GLInfo::gl_polygon_offset_factor = -1
```

6.4.2.142 gl_polygon_offset_fill

```
int gpu::GLInfo::gl_polygon_offset_fill = -1
```

6.4.2.143 gl_polygon_offset_line

```
int gpu::GLInfo::gl_polygon_offset_line = -1
```

6.4.2.144 gl_polygon_offset_point

```
int gpu::GLInfo::gl_polygon_offset_point = -1
```

6.4.2.145 gl_polygon_offset_units

```
int gpu::GLInfo::gl_polygon_offset_units = -1
```

6.4.2.146 gl_polygon_smooth

```
int gpu::GLInfo::gl_polygon_smooth = -1
```

6.4.2.147 gl_polygon_smooth_hint

```
int gpu::GLInfo::gl_polygon_smooth_hint = -1
```

6.4.2.148 gl_primitive_restart_index

```
int gpu::GLInfo::gl_primitive_restart_index = -1
```

6.4.2.149 gl_program_binary_formats

```
int gpu::GLInfo::gl_program_binary_formats = -1
```

6.4.2.150 gl_program_pipeline_binding

```
int gpu::GLInfo::gl_program_pipeline_binding = -1
```

6.4.2.151 gl_program_point_size

```
int gpu::GLInfo::gl_program_point_size = -1
```

6.4.2.152 gl_provoking_vertex

```
int gpu::GLInfo::gl_provoking_vertex = -1
```

6.4.2.153 gl_read_buffer

```
int gpu::GLInfo::gl_read_buffer = -1
```

6.4.2.154 gl_read_framebuffer_binding

```
int gpu::GLInfo::gl_read_framebuffer_binding = -1
```

6.4.2.155 gl_renderbuffer_binding

```
int gpu::GLInfo::gl_renderbuffer_binding = -1
```

6.4.2.156 gl_sample_buffers

```
int gpu::GLInfo::gl_sample_buffers = -1
```

6.4.2.157 gl_sample_coverage_invert

```
int gpu::GLInfo::gl_sample_coverage_invert = -1
```

6.4.2.158 gl_sample_coverage_value

```
int gpu::GLInfo::gl_sample_coverage_value = -1
```

6.4.2.159 gl_sampler_binding

```
int gpu::GLInfo::gl_sampler_binding = -1
```

6.4.2.160 gl_samples

```
int gpu::GLInfo::gl_samples = -1
```

6.4.2.161 gl_scissor_box

```
int gpu::GLInfo::gl_scissor_box = -1
```

6.4.2.162 gl_scissor_test

```
int gpu::GLInfo::gl_scissor_test = -1
```

6.4.2.163 gl_shader_compiler

```
int gpu::GLInfo::gl_shader_compiler = -1
```

6.4.2.164 gl_shader_storage_buffer_binding

```
int gpu::GLInfo::gl_shader_storage_buffer_binding = -1
```

6.4.2.165 gl_shader_storage_buffer_offset_alignment

```
int gpu::GLInfo::gl_shader_storage_buffer_offset_alignment = -1
```

6.4.2.166 gl_shader_storage_buffer_size

```
int gpu::GLInfo::gl_shader_storage_buffer_size = -1
```

6.4.2.167 gl_shader_storage_buffer_start

```
int gpu::GLInfo::gl_shader_storage_buffer_start = -1
```

6.4.2.168 gl_smooth_line_width_granularity

```
int gpu::GLInfo::gl_smooth_line_width_granularity = -1
```

6.4.2.169 gl_smooth_line_width_range

```
int gpu::GLInfo::gl_smooth_line_width_range = -1
```

6.4.2.170 gl_stencil_back_fail

```
int gpu::GLInfo::gl_stencil_back_fail = -1
```

6.4.2.171 gl_stencil_back_func

```
int gpu::GLInfo::gl_stencil_back_func = -1
```

6.4.2.172 gl_stencil_back_pass_depth_fail

```
int gpu::GLInfo::gl_stencil_back_pass_depth_fail = -1
```

6.4.2.173 gl_stencil_back_pass_depth_pass

```
int gpu::GLInfo::gl_stencil_back_pass_depth_pass = -1
```

6.4.2.174 gl_stencil_back_ref

```
int gpu::GLInfo::gl_stencil_back_ref = -1
```

6.4.2.175 gl_stencil_back_value_mask

```
int gpu::GLInfo::gl_stencil_back_value_mask = -1
```

6.4.2.176 gl_stencil_back_writemask

```
int gpu::GLInfo::gl_stencil_back_writemask = -1
```

6.4.2.177 gl_stencil_clear_value

```
int gpu::GLInfo::gl_stencil_clear_value = -1
```

6.4.2.178 gl_stencil_fail

```
int gpu::GLInfo::gl_stencil_fail = -1
```

6.4.2.179 gl_stencil_func

```
int gpu::GLInfo::gl_stencil_func = -1
```

6.4.2.180 gl_stencil_pass_depth_fail

```
int gpu::GLInfo::gl_stencil_pass_depth_fail = -1
```

6.4.2.181 gl_stencil_pass_depth_pass

```
int gpu::GLInfo::gl_stencil_pass_depth_pass = -1
```

6.4.2.182 gl_stencil_ref

```
int gpu::GLInfo::gl_stencil_ref = -1
```

6.4.2.183 gl_stencil_test

```
int gpu::GLInfo::gl_stencil_test = -1
```

6.4.2.184 gl_stencil_value_mask

```
int gpu::GLInfo::gl_stencil_value_mask = -1
```

6.4.2.185 gl_stencil_writemask

```
int gpu::GLInfo::gl_stencil_writemask = -1
```

6.4.2.186 gl_stereo

```
int gpu::GLInfo::gl_stereo = -1
```

6.4.2.187 gl_subpixel_bits

```
int gpu::GLInfo::gl_subpixel_bits = -1
```

6.4.2.188 gl_texture_binding_1d

```
int gpu::GLInfo::gl_texture_binding_1d = -1
```

6.4.2.189 gl_texture_binding_1d_array

```
int gpu::GLInfo::gl_texture_binding_1d_array = -1
```

6.4.2.190 gl_texture_binding_2d

```
int gpu::GLInfo::gl_texture_binding_2d = -1
```

6.4.2.191 gl_texture_binding_2d_array

```
int gpu::GLInfo::gl_texture_binding_2d_array = -1
```

6.4.2.192 gl_texture_binding_2d_multisample

```
int gpu::GLInfo::gl_texture_binding_2d_multisample = -1
```

6.4.2.193 gl_texture_binding_2d_multisample_array

```
int gpu::GLInfo::gl_texture_binding_2d_multisample_array = -1
```

6.4.2.194 gl_texture_binding_3d

```
int gpu::GLInfo::gl_texture_binding_3d = -1
```

6.4.2.195 gl_texture_binding_buffer

```
int gpu::GLInfo::gl_texture_binding_buffer = -1
```

6.4.2.196 gl_texture_binding_cube_map

```
int gpu::GLInfo::gl_texture_binding_cube_map = -1
```

6.4.2.197 gl_texture_binding_rectangle

```
int gpu::GLInfo::gl_texture_binding_rectangle = -1
```

6.4.2.198 gl_texture_buffer_offset_alignment

```
int gpu::GLInfo::gl_texture_buffer_offset_alignment = -1
```

6.4.2.199 gl_texture_compression_hint

```
int gpu::GLInfo::gl_texture_compression_hint = -1
```

6.4.2.200 gl_timestamp

```
int gpu::GLInfo::gl_timestamp = -1
```

6.4.2.201 gl_transform_feedback_buffer_binding

```
int gpu::GLInfo::gl_transform_feedback_buffer_binding = -1
```

6.4.2.202 gl_transform_feedback_buffer_size

```
int gpu::GLInfo::gl_transform_feedback_buffer_size = -1
```

6.4.2.203 gl_transform_feedback_buffer_start

```
int gpu::GLInfo::gl_transform_feedback_buffer_start = -1
```

6.4.2.204 gl_uniform_buffer_binding

```
int gpu::GLInfo::gl_uniform_buffer_binding = -1
```

6.4.2.205 gl_uniform_buffer_offset_alignment

```
int gpu::GLInfo::gl_uniform_buffer_offset_alignment = -1
```

6.4.2.206 gl_uniform_buffer_size

```
int gpu::GLInfo::gl_uniform_buffer_size = -1
```

6.4.2.207 gl_uniform_buffer_start

```
int gpu::GLInfo::gl_uniform_buffer_start = -1
```

6.4.2.208 gl_unpack_alignment

```
int gpu::GLInfo::gl_unpack_alignment = -1
```

6.4.2.209 gl_unpack_image_height

```
int gpu::GLInfo::gl_unpack_image_height = -1
```

6.4.2.210 gl_unpack_lsb_first

```
int gpu::GLInfo::gl_unpack_lsb_first = -1
```

6.4.2.211 gl_unpack_row_length

```
int gpu::GLInfo::gl_unpack_row_length = -1
```

6.4.2.212 gl_unpack_skip_images

```
int gpu::GLInfo::gl_unpack_skip_images = -1
```

6.4.2.213 gl_unpack_skip_pixels

```
int gpu::GLInfo::gl_unpack_skip_pixels = -1
```

6.4.2.214 gl_unpack_skip_rows

```
int gpu::GLInfo::gl_unpack_skip_rows = -1
```

6.4.2.215 gl_unpack_swap_bytes

```
int gpu::GLInfo::gl_unpack_swap_bytes = -1
```

6.4.2.216 gl_vertex_array_binding

```
int gpu::GLInfo::gl_vertex_array_binding = -1
```

6.4.2.217 gl_vertex_binding_divisor

```
int gpu::GLInfo::gl_vertex_binding_divisor = -1
```

6.4.2.218 gl_vertex_binding_offset

```
int gpu::GLInfo::gl_vertex_binding_offset = -1
```

6.4.2.219 gl_vertex_binding_stride

```
int gpu::GLInfo::gl_vertex_binding_stride = -1
```

6.4.2.220 gl_viewport

```
int gpu::GLInfo::gl_viewport = -1
```

6.4.2.221 gl_viewport_bounds_range

```
int gpu::GLInfo::gl_viewport_bounds_range = -1
```

6.4.2.222 gl_viewport_index_provoking_vertex

```
int gpu::GLInfo::gl_viewport_index_provoking_vertex = -1
```

6.4.2.223 gl_viewport_subpixel_bits

```
int gpu::GLInfo::gl_viewport_subpixel_bits = -1
```

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

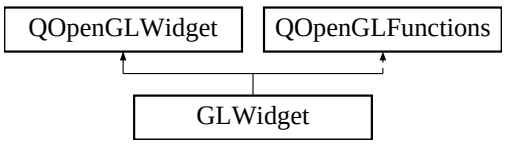
- libs/opengl-engine/libs/profiler/src/GLInfo.hpp

6.5 GLWidget Class Reference

Class for gl widget. OpenGL Widget class based on QGLWidget and QOpenGLFunctions to implement the OpenGL API.

```
#include <glwidget.h>
```

Inheritance diagram for GLWidget:



Public Member Functions

- GLWidget (QWidget *parent=nullptr)

  Constructs the object. Builds the OpenGL widget.
- ~GLWidget ()

  Destroys the object.
- bool setScene ()

  Sets the scene.
- void setProfiler (gpu::Profiler *p)

  Sets the profiler. Profiler passed as a pointer so it can be retrieved by the parent.
- void setTargetFramerate (int framerate)

  Sets target refresh rate.

Protected Member Functions

- void initializeGL ()

  Initialises OpenGL context.
- void paintGL ()

  Rendering function. Binds the model data and renders each cube of the scene.
- void resizeGL (int width, int height)

  Handles widget resize based on the window size.

6.5.1 Detailed Description

Class for gl widget. OpenGL Widget class based on QGLWidget and QOpenGLFunctions to implement the OpenGL API.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 GLWidget()

```
GLWidget::GLWidget (
            QWidget * parent = nullptr )
```

Constructs the object. Builds the OpenGL widget.

Parameters

parent	The parent window

6.5.2.2 ~GLWidget()

```
GLWidget::~GLWidget ( )  [inline]
```

Destroys the object.

6.5.3 Member Function Documentation

6.5.3.1 initializeGL()

```
void GLWidget::initializeGL ( )  [protected]
```

Initialises OpenGL context.

6.5.3.2 paintGL()

```
void GLWidget::paintGL ( )  [protected]
```

Rendering function. Binds the model data and renders each cube of the scene.

6.5.3.3 resizeGL()

```
void GLWidget::resizeGL (
            int width,
            int height )  [protected]
```

Handles widget resize based on the window size.

Parameters

in	*width*	The window width
in	*height*	The window height

6.5.3.4 setProfiler()

```
void GLWidget::setProfiler (
            gpu::Profiler * p )  [inline]
```

Sets the profiler. Profiler passed as a pointer so it can be retrieved by the parent.

Parameters

p	GPU Profiler

6.5.3.5 setScene()

```
bool GLWidget::setScene ( )
```

Sets the scene.

Returns

True if the scene is loaded successfully, False otherwise.

6.5.3.6 setTargetFramerate()

```
void GLWidget::setTargetFramerate (
            int framerate )  [inline]
```

Sets target refresh rate.

Parameters

framerate	Target framerate

The documentation for this class was generated from the following files:

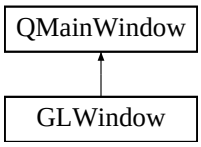
- libs/opengl-engine/src/glwidget.h
- libs/opengl-engine/src/getgpuinfo.cpp
- libs/opengl-engine/src/glwidget.cpp

6.6 GLWindow Class Reference

Class for gl window. OpenGL window based on QMainWindow widget.

`#include <glwindow.h>`

Inheritance diagram for GLWindow:



Public Member Functions

- GLWindow (QWidget *parent=nullptr)

  Constructs the object. Builds the GL Window. Keeps a refernce to the parent window.
- ~GLWindow ()

  Destroys the object.
- void setProfiler (gpu::Profiler *p)

  Sets the profiler. Profiler passed as a pointer so that the data can be retrieved by the parent.
- void setTargetFramerate (int framerate)

  Sets target framerate. Assigns target refresh rate.

6.6.1 Detailed Description

Class for gl window. OpenGL window based on QMainWindow widget.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 GLWindow()

```
GLWindow::GLWindow (
            QWidget * parent = nullptr )  [explicit]
```

Constructs the object. Builds the GL Window. Keeps a refernce to the parent window.

Parameters

parent	The parent window

6.6.2.2 ~GLWindow()

```
GLWindow::~GLWindow ( )
```

Destroys the object.

6.6.3 Member Function Documentation

6.6.3.1 setProfiler()

```
void GLWindow::setProfiler (
            gpu::Profiler * p )  [inline]
```

Sets the profiler. Profiler passed as a pointer so that the data can be retrieved by the parent.

Parameters

p	Profiler used to record the data

6.6.3.2 setTargetFramerate()

```
void GLWindow::setTargetFramerate (
            int framerate )  [inline]
```

Sets target framerate. Assigns target refresh rate.

Parameters

framerate	Target framerate

The documentation for this class was generated from the following files:

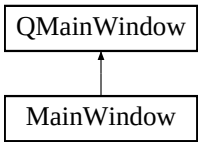
- libs/opengl-engine/src/glwindow.h
- libs/opengl-engine/src/glwindow.cpp

6.7 MainWindow Class Reference

Class for main window. Main Window based on QMainWindow Widget. Contains the profilers and the charts.

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:



Public Member Functions

- MainWindow (QWidget *parent=nullptr)
  Constructs the object. Builds the Main Window.
- ~MainWindow ()
  Destroys the object.

6.7.1 Detailed Description

Class for main window. Main Window based on QMainWindow Widget. Contains the profilers and the charts.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 MainWindow()

```
MainWindow::MainWindow (
            QWidget * parent = nullptr )
```

Constructs the object. Builds the Main Window.

Parameters

parent	The parent window (null in this case)

6.7.2.2 ~MainWindow()

```
MainWindow::~MainWindow ( )
```

Destroys the object.

The documentation for this class was generated from the following files:

- src/mainwindow.h
- src/mainwindow.cpp
- src/runners.cpp
- src/updateGLresults.cpp
- src/updateVKresults.cpp

6.8 Mesh Class Reference

Class for mesh. Loads the 3D model (mesh). Keeps track of vertex count and saves data in a QByteArray.

`#include <mesh.h>`

Public Member Functions

- Mesh ()
  Constructs the object. Empty constructor.
- bool load (const QString &file)
  Loads mesh. Loads mesh from file and saves data in QByteArray.

Public Attributes

- int vertexCount = 0
- float aabb [6]
- QByteArray geom

6.8.1 Detailed Description

Class for mesh. Loads the 3D model (mesh). Keeps track of vertex count and saves data in a QByteArray.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Mesh()

```
Mesh::Mesh ( )  [inline]
```

Constructs the object. Empty constructor.

6.8.3 Member Function Documentation

6.8.3.1 load()

```
bool Mesh::load (
            const QString & file )
```

Loads mesh. Loads mesh from file and saves data in QByteArray.

Parameters

in	*file*	The file to load

Returns

True if load is successful, False otherwise.

6.8.4 Member Data Documentation

6.8.4.1 aabb

```
float Mesh::aabb[6]
```

6.8.4.2 geom

```
QByteArray Mesh::geom
```

6.8.4.3 vertexCount

```
int Mesh::vertexCount = 0
```

The documentation for this class was generated from the following files:

- libs/vulkan-engine/src/mesh.h
- libs/vulkan-engine/src/mesh.cpp

6.9 Model Class Reference

Class for model. 3D model loaded as an OBJ. Has a reference to the OBJ instance, the buffers (vertex and UVs) buffers. Contains model array object, shader program and texture.

```
#include <model.hpp>
```

Public Member Functions

- Model ()

  Constructs the object. Empty constructor.
- Model (const Model &m)

  Constructs the object. Copy constructor.
- ~Model ()

  Destroys the object. Deletes the shader program and the texture.
- void setup ()

  Setups the model. Creates the shader model, creates and binds the buffers and loads the shaders.

Public Attributes

- obj::OBJ obj
- std::vector< unsigned int > indices
- QOpenGLBuffer vertexBuffer
- QOpenGLBuffer uvsBuffer
- QOpenGLVertexArrayObject modelObject
- QOpenGLShaderProgram * shaderProgram
- QOpenGLTexture * texture

6.9.1 Detailed Description

Class for model. 3D model loaded as an OBJ. Has a reference to the OBJ instance, the buffers (vertex and UVs) buffers. Contains model array object, shader program and texture.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Model() [1/2]

```
Model::Model ( )  [inline]
```

Constructs the object. Empty constructor.

6.9.2.2 Model() [2/2]

```
Model::Model (
            const Model & m )  [inline]
```

Constructs the object. Copy constructor.

Parameters

in	*m*	Model to copy

6.9.2.3 ~Model()

```
Model::~Model ( )  [inline]
```

Destroys the object. Deletes the shader program and the texture.

6.9.3 Member Function Documentation

6.9.3.1 setup()

```
void Model::setup ( )  [inline]
```

Setups the model. Creates the shader model, creates and binds the buffers and loads the shaders.

6.9.4 Member Data Documentation

6.9.4.1 indices

```
std::vector<unsigned int> Model::indices
```

6.9.4.2 modelObject

```
QOpenGLVertexArrayObject Model::modelObject
```

6.9.4.3 obj

```
obj::OBJ Model::obj
```

6.9.4.4 shaderProgram

```
QOpenGLShaderProgram* Model::shaderProgram
```

6.9.4.5 texture

```
QOpenGLTexture* Model::texture
```

6.9.4.6 uvsBuffer

```
QOpenGLBuffer Model::uvsBuffer
```

6.9.4.7 vertexBuffer

```
QOpenGLBuffer Model::vertexBuffer
```

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

- libs/opengl-engine/src/model.hpp

6.10 obj::OBJ Class Reference

Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index.

```
#include <OBJ.h>
```

Public Member Functions

- OBJ ()
  Constructs the object. Empty constructor.
- OBJ (const char *modelPath)
  Constructs the object. Loads a 3D model from path.
- ~OBJ ()
  Destroys the object.
- void loadModel (const char *filename)
  Loads a model. Loads a 3D model given a path.
- void saveModel (const char *filename)
  Saves a model. Saves the OBJ to disk given a path.
- bool isLoaded ()
  Determines if loaded. Checks if the OBJ is successfully loaded.
- std::vector< math::Vector3 > getVertices ()
  Gets the vertices. Fetches the list of vertices in the OBJ.
- std::vector< math::Vector3 > getNormals ()
  Gets the normals. Fetches the list of normals in the OBJ.
- std::vector< math::Vector2 > getUVs ()
  Gets the texture coordinates. Fetches the list of UV coordinates in the OBJ.
- std::vector< Face > getFaces ()
  Gets the faces. Fetches the list of faces in the OBJ.
- std::vector< unsigned > getIndices ()
  Gets the indices. Fetches the list of indices in the OBJ.
- unsigned int addVertex (math::Vector3 vertex)
  Adds a vertex. Adds a vertex to the OBJ.
- void updateFaces (std::vector< Face > newFaces)
  Updates the faces. Updates the list of faces.

6.10.1 Detailed Description

Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 OBJ() [1/2]

```
OBJ::OBJ ( )
```

Constructs the object. Empty constructor.

6.10.2.2 OBJ() [2/2]

```
OBJ::OBJ (
            const char * modelPath )
```

Constructs the object. Loads a 3D model from path.

Parameters

in	*modelPath*	The model path to load

6.10.2.3 ~OBJ()

```
obj::OBJ::~OBJ ( )  [inline]
```

Destroys the object.

6.10.3 Member Function Documentation

6.10.3.1 addVertex()

```
unsigned int OBJ::addVertex (
            math::Vector3 vertex )
```

Adds a vertex. Adds a vertex to the OBJ.

Parameters

in	*vertex*	The vertex

Returns

Vertex index.

6.10.3.2 getFaces()

```
std::vector<Face> obj::OBJ::getFaces ( )  [inline]
```

Gets the faces. Fetches the list of faces in the OBJ.

Returns

The faces list.

6.10.3.3 getIndices()

```
std::vector< unsigned > OBJ::getIndices ( )
```

Gets the indices. Fetches the list of indices in the OBJ.

Returns

The indices list.

6.10.3.4 getNormals()

```
std::vector<math::Vector3> obj::OBJ::getNormals ( )  [inline]
```

Gets the normals. Fetches the list of normals in the OBJ.

Returns

The normals list.

6.10.3.5 getUVs()

```
std::vector<math::Vector2> obj::OBJ::getUVs ( )  [inline]
```

Gets the texture coordinates. Fetches the list of UV coordinates in the OBJ.

Returns

The texture coordinates list.

6.10.3.6 getVertices()

```
std::vector<math::Vector3> obj::OBJ::getVertices ( )  [inline]
```

Gets the vertices. Fetches the list of vertices in the OBJ.

Returns

The vertices list.

6.10.3.7 isLoaded()

```
bool obj::OBJ::isLoaded ( )  [inline]
```

Determines if loaded. Checks if the OBJ is successfully loaded.

Returns

True if loaded, False otherwise.

6.10.3.8 loadModel()

```
void OBJ::loadModel (
            const char * filename )
```

Loads a model. Loads a 3D model given a path.

Parameters

in	*filename*	The filename

6.10.3.9 saveModel()

```
void OBJ::saveModel (
            const char * filename )
```

Saves a model. Saves the OBJ to disk given a path.

Parameters

in	*filename*	The filename

6.10.3.10 updateFaces()

```
void obj::OBJ::updateFaces (
            std::vector< Face > newFaces )  [inline]
```

Updates the faces. Updates the list of faces.

Parameters

in	*newFaces*	The new faces

The documentation for this class was generated from the following files:

- libs/opengl-engine/libs/obj-loader/src/OBJ.h
- libs/opengl-engine/libs/obj-loader/src/OBJ.cpp

6.11 gpu::Profiler Class Reference

Class for profiler. Saves the GPU information: vendor, renderer. OpenGL version, shading version, extensions. Vulkan device name, API version, driver version, vendor ID, device ID, extensions. Std::vector stores Profiling data (polycount per time).

```
#include <Profiler.h>
```

Public Types

- enum TYPE { TRIANGLES }

Public Member Functions

- Profiler ()

  Constructs the object. Empty constructor.
- ~Profiler ()

  Destroys the object.
- bool save (std::string filename)

  Save data to csv. Saves the data gathered into a CSV file.
- void clear ()

  Clear profiler. Deletes the data vector.
- void setVendor (std::string data)

  Sets the GPU vendor.
- void setRenderer (std::string data)

  Sets the renderer.
- void setGLVersion (std::string data)

  Sets the gl version.
- void setGLShadingLanguageVersion (std::string data)

  Sets the gl shading language version.
- void setGLExtensions (std::string data)

  Sets the gl extensions.
- void setVkDeviceName (std::string data)

  Sets the vulkan device name.
- void setVkApiVersion (uint32_t data)

  Sets the vulkan api version.
- void setVkDriverVersion (uint32_t data)

  Sets the vulkan driver version.
- void setVkVendorID (uint32_t data)

  Sets the vulkan vendor id.
- void setVkDeviceID (uint32_t data)

  Sets the vulkan device id.
- void setVkExtensions (std::vector< std::string > extensions)

  Sets the vulkan extensions.
- void addTrianglesEntry (unsigned numberOfTriangles, unsigned time)

  Adds an entry to the profiler. Add en entry given a polycount and the time taken to render the frame.
- std::vector< std::vector< unsigned > > getData ()

  Gets the data recorded.
- std::string getVendor ()

  Gets the GPU vendor.
- std::string getRenderer ()

  Gets the renderer.
- std::string getGLVersion ()

  Gets the gl version.
- std::string getGLShadingLanguageVersion ()

  Gets the gl shading language version.
- std::string getGLExtensions ()

  Gets the gl extensions.
- std::string getVkDeviceName ()

  Gets the vulkan device name.
- uint32_t getVkApiVersion ()

  Gets the vulkan api version.
- uint32_t getVkDriverVersion ()

Gets the vulkan driver version.

- uint32_t getVkVendorID ()

  Gets the vulkan vendor id.

- uint32_t getVkDeviceID ()

  Gets the vulkan device id.

- std::vector< std::string > getVkExtensions ()

  Gets the vulkan extensions.

Public Attributes

- GLInfo glInfo
- VkInfo vkInfo

6.11.1 Detailed Description

Class for profiler. Saves the GPU information: vendor, renderer. OpenGL version, shading version, extensions. Vulkan device name, API version, driver version, vendor ID, device ID, extensions. Std::vector stores Profiling data (polycount per time).

6.11.2 Member Enumeration Documentation

6.11.2.1 TYPE

```
enum gpu::Profiler::TYPE
```

Enumerator

TRIANGLES	

6.11.3 Constructor & Destructor Documentation

6.11.3.1 Profiler()

```
gpu::Profiler::Profiler ( )  [inline]
```

Constructs the object. Empty constructor.

6.11.3.2 ~Profiler()

```
gpu::Profiler::~Profiler ( )  [inline]
```

Destroys the object.

6.11.4 Member Function Documentation

6.11.4.1 addTrianglesEntry()

```
void Profiler::addTrianglesEntry (
            unsigned numberOfTriangles,
            unsigned time )
```

Adds an entry to the profiler. Add en entry given a polycount and the time taken to render the frame.

Parameters

in	*numberOfTriangles*	The number of triangles
in	*time*	The time taken to render a frame

6.11.4.2 clear()

```
void gpu::Profiler::clear ( )  [inline]
```

Clear profiler. Deletes the data vector.

6.11.4.3 getData()

```
std::vector<std::vector<unsigned> > gpu::Profiler::getData ( )  [inline]
```

Gets the data recorded.

Returns

The data recorded until now.

6.11.4.4 getGLExtensions()

```
std::string gpu::Profiler::getGLExtensions ( )  [inline]
```

Gets the gl extensions.

Returns

The data.

6.11.4.5 getGLShadingLanguageVersion()

```
std::string gpu::Profiler::getGLShadingLanguageVersion ( )  [inline]
```

Gets the gl shading language version.

Returns

The data.

6.11.4.6 getGLVersion()

```
std::string gpu::Profiler::getGLVersion ( )  [inline]
```

Gets the gl version.

Returns

The data.

6.11.4.7 getRenderer()

```
std::string gpu::Profiler::getRenderer ( )  [inline]
```

Gets the renderer.

Returns

The data.

6.11.4.8 getVendor()

```
std::string gpu::Profiler::getVendor ( )  [inline]
```

Gets the GPU vendor.

Returns

The data.

6.11.4.9 getVkApiVersion()

```
uint32_t gpu::Profiler::getVkApiVersion ( )  [inline]
```

Gets the vulkan api version.

Returns

The vulkan api version.

6.11.4.10 getVkDeviceID()

```
uint32_t gpu::Profiler::getVkDeviceID ( )  [inline]
```

Gets the vulkan device id.

Returns

The vulkan device id.

6.11.4.11 getVkDeviceName()

```
std::string gpu::Profiler::getVkDeviceName ( )  [inline]
```

Gets the vulkan device name.

Returns

The vulkan device name.

6.11.4.12 getVkDriverVersion()

```
uint32_t gpu::Profiler::getVkDriverVersion ( )  [inline]
```

Gets the vulkan driver version.

Returns

The vulkan driver version.

6.11.4.13 getVkExtensions()

```
std::vector<std::string> gpu::Profiler::getVkExtensions ( )  [inline]
```

Gets the vulkan extensions.

Returns

The vulkan extensions.

6.11.4.14 getVkVendorID()

```
uint32_t gpu::Profiler::getVkVendorID ( )  [inline]
```

Gets the vulkan vendor id.

Returns

The vulkan vendor id.

6.11.4.15 save()

```
bool Profiler::save (
            std::string filename )
```

Save data to csv. Saves the data gathered into a CSV file.

Parameters

in	*filename*	The filename

Returns

True if save is successful, False otherwise.

6.11.4.16 setGLExtensions()

```
void gpu::Profiler::setGLExtensions (
            std::string data )  [inline]
```

Sets the gl extensions.

Parameters

in	*data*	The data

6.11.4.17 setGLShadingLanguageVersion()

```
void gpu::Profiler::setGLShadingLanguageVersion (
            std::string data )  [inline]
```

Sets the gl shading language version.

Parameters

in	*data*	The data

6.11.4.18 setGLVersion()

```
void gpu::Profiler::setGLVersion (
            std::string data )  [inline]
```

Sets the gl version.

Parameters

in	*data*	The data

6.11.4.19 setRenderer()

```
void gpu::Profiler::setRenderer (
```

```
            std::string data )  [inline]
```

Sets the renderer.

Parameters

in	*data*	The data

6.11.4.20 setVendor()

```
void gpu::Profiler::setVendor (
            std::string data )  [inline]
```

Sets the GPU vendor.

Parameters

in	*data*	The data

6.11.4.21 setVkApiVersion()

```
void gpu::Profiler::setVkApiVersion (
            uint32_t data )  [inline]
```

Sets the vulkan api version.

Parameters

in	*data*	The data

6.11.4.22 setVkDeviceID()

```
void gpu::Profiler::setVkDeviceID (
            uint32_t data )  [inline]
```

Sets the vulkan device id.

Parameters

in	*data*	The data

6.11.4.23 setVkDeviceName()

```
void gpu::Profiler::setVkDeviceName (
            std::string data )  [inline]
```

Sets the vulkan device name.

Parameters

in	*data*	The data

6.11.4.24 setVkDriverVersion()

```
void gpu::Profiler::setVkDriverVersion (
            uint32_t data )  [inline]
```

Sets the vulkan driver version.

Parameters

in	*data*	The data

6.11.4.25 setVkExtensions()

```
void gpu::Profiler::setVkExtensions (
            std::vector< std::string > extensions )  [inline]
```

Sets the vulkan extensions.

Parameters

in	*extensions*	The extensions

6.11.4.26 setVkVendorID()

```
void gpu::Profiler::setVkVendorID (
            uint32_t data )  [inline]
```

Sets the vulkan vendor id.

Parameters

in	*data*	The data

6.11.5 Member Data Documentation

6.11.5.1 glInfo

```
GLInfo gpu::Profiler::glInfo
```

6.11.5.2 vkInfo

```
VkInfo gpu::Profiler::vkInfo
```

The documentation for this class was generated from the following files:

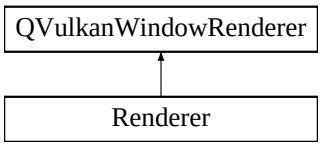
- libs/opengl-engine/libs/profiler/src/Profiler.h
- libs/opengl-engine/libs/profiler/src/Profiler.cpp

6.12 Renderer Class Reference

Class for renderer. Renderer implementation based on the QVulkanWindowRenderer widget.

```
#include <renderer.h>
```

Inheritance diagram for Renderer:



Public Member Functions

- Renderer (QVulkanWindow *w, gpu::Profiler *profiler=nullptr, int framerate=60)

  Constructs the object. Creates an instance of the Renderer.
- void initResources () override

  Initialises the resources. Initialises the shaders, models, ...
- void initSwapChainResources () override

  Initialises the swap chain resources.
- void releaseSwapChainResources () override

  Releases the swap chain resources (cleans the swap chain).
- void releaseResources () override

  Releases the resources (clean). Releases the shader resources, models data, ...
- void startNextFrame () override

  Starts the next frame. Builds the next frame.

6.12.1 Detailed Description

Class for renderer. Renderer implementation based on the QVulkanWindowRenderer widget.

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Renderer()

```
Renderer::Renderer (
            QVulkanWindow * w,
            gpu::Profiler * profiler = nullptr,
            int framerate = 60 )
```

Constructs the object. Creates an instance of the Renderer.

Parameters

w	Vulkan Window (QVulkanWindow)
profiler	The profiler used to record the GPU information
framerate	The target framerate (60 by default)

6.12.3 Member Function Documentation

6.12.3.1 initResources()

```
void Renderer::initResources ( )  [override]
```

Initialises the resources. Initialises the shaders, models, ...

6.12.3.2 initSwapChainResources()

```
void Renderer::initSwapChainResources ( )  [override]
```

Initialises the swap chain resources.

6.12.3.3 releaseResources()

```
void Renderer::releaseResources ( )  [override]
```

Releases the resources (clean). Releases the shader resources, models data, ...

6.12.3.4 releaseSwapChainResources()

```
void Renderer::releaseSwapChainResources ( )  [override]
```

Releases the swap chain resources (cleans the swap chain).

6.12.3.5 startNextFrame()

```
void Renderer::startNextFrame ( )  [override]
```

Starts the next frame. Builds the next frame.

The documentation for this class was generated from the following files:

- libs/vulkan-engine/src/renderer.h
- libs/vulkan-engine/src/renderer.cpp
- libs/vulkan-engine/src/setvkinfo.cpp

6.13 Shader Class Reference

Class for shader. Loads a shader for file so that it can be used by the Vulkan Engine.

```
#include <shader.h>
```

Public Member Functions

- Shader ()

  Constructs the object. Empty constructor.
- bool load (QVulkanInstance *instance, VkDevice device, const QString &shaderPath)

  Loads a shader given a path. Loads the shader from file and create the shader module using the QVulkanInstance and the VkDevice.
- bool isValid ()

  Determines if valid. Checks if the shader has been loaded successfully.

Public Attributes

- VkShaderModule shaderModule = nullptr

6.13.1 Detailed Description

Class for shader. Loads a shader for file so that it can be used by the Vulkan Engine.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 Shader()

```
Shader::Shader ( )  [inline]
```

Constructs the object. Empty constructor.

6.13.3 Member Function Documentation

6.13.3.1 isValid()

```
bool Shader::isValid ( )  [inline]
```

Determines if valid. Checks if the shader has been loaded successfully.

Returns

True if valid, False otherwise.

6.13.3.2 load()

```
bool Shader::load (
            QVulkanInstance * instance,
            VkDevice device,
            const QString & shaderPath )
```

Loads a shader given a path. Loads the shader from file and create the shader module using the QVulkanInstance and the VkDevice.

Parameters

	instance	The vulkan instance
in	*device*	The logical device that interfaces with the physical GPU
in	*shaderPath*	The shader path

Returns

True if load is successful, False otherwise.

6.13.4 Member Data Documentation

6.13.4.1 shaderModule

```
VkShaderModule Shader::shaderModule = nullptr
```

The documentation for this class was generated from the following files:

- libs/vulkan-engine/src/shader.h
- libs/vulkan-engine/src/shader.cpp

6.14 math::Vector2 Class Reference

2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()

```
#include <Vector2.hpp>
```

Public Member Functions

- Vector2 ()

  Constructs the object. Empty construcor sets x=0 and y=0.
- Vector2 (double x, double y)

  Constructs the object. Uses the x and y provided to build the 2D vector.
- ∼Vector2 ()

  Destroys the object.
- Vector2 operator+ (const Vector2 &other)

  Overloads + operator. Vec+Vec, sum of the x and y components of the 2D vector.
- void operator+= (const Vector2 &other)

  Overloads += operator. Vec+=Vec, sum of the x and y components.
- Vector2 operator- (const Vector2 &other)

  Overloads - operator. Vec-Vec, subtraction of x and y components.
- void operator-= (const Vector2 &other)

  Overloads -= operator. Vec-=Vec, subtraction of x and y components.
- void operator= (const Vector2 &other)

  Overloads = operator. Vec=Vec, assign x and y components to left hand side 2D vector.
- bool operator== (const Vector2 &other)

  Overloads == operator. Vec==Vec, compares if x and y components are the same.
- Vector2 operator∗ (double value)

  Overloads ∗ operator. Vec∗Vec, multiplies the x and y components of the 2D vectors.
- void operator∗= (double value)

Overloads ∗= operator. Vec∗=Vec, multiplies the x and y components of the 2D vectors.

- Vector2 operator/ (double value)

  Overloads / operator. Vec/Vec, divide left vector by right vector.
- void operator/= (double value)

  Overloads /= operator. Vec/=Vec, divide left vector by right vector.
- double dot (Vector2 &other)

  Dot product. Vec.dot(Vec), result of the dot product operation. V1 is first vector and V2 second vector, DotProduct↩ Result = V1.x ∗ V2.x + V1.y ∗ V2.y.
- Vector2 normal ()

  Computes the vector mormal. Vec.normal(), divides x and y components by magnitude.
- void normalise ()

  Normalises the vector. Vec.normal(), divides x and y components by magnitude.
- double magnitude ()

  Computes the magnitude. Magnitude = sqrt(x∗x + y∗y).
- double magnitudeSquared ()

  Computes the magnitude squared. MagnitudeSquared = x∗x + y∗y.

Public Attributes

- double x
- double y

6.14.1 Detailed Description

2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Vector2() [1/2]

```
math::Vector2::Vector2 ( )  [inline]
```

Constructs the object. Empty construcor sets x=0 and y=0.

6.14.2.2 Vector2() [2/2]

```
math::Vector2::Vector2 (
            double x,
            double y )  [inline]
```

Constructs the object. Uses the x and y provided to build the 2D vector.

Parameters

in	*x*	x component of the 2D vector
in	*y*	y component of the 2D vector

6.14.2.3 ~Vector2()

```
math::Vector2::~Vector2 ( )  [inline]
```

Destroys the object.

6.14.3 Member Function Documentation

6.14.3.1 dot()

```
double math::Vector2::dot (
            Vector2 & other )  [inline]
```

Dot product. Vec.dot(Vec), result of the dot product operation. V1 is first vector and V2 second vector, DotProduct↩ Result = V1.x ∗ V2.x + V1.y ∗ V2.y.

Parameters

other	The other vector used in the operation

Returns

Result of the dot product.

6.14.3.2 magnitude()

```
double math::Vector2::magnitude ( )  [inline]
```

Computes the magnitude. Magnitude = sqrt(x∗x + y∗y).

Returns

The vector magnitude.

6.14.3.3 magnitudeSquared()

```
double math::Vector2::magnitudeSquared ( )  [inline]
```

Computes the magnitude squared. MagnitudeSquared = x∗x + y∗y.

Returns

The magnitude squared.

6.14.3.4 normal()

```
Vector2 math::Vector2::normal ( )  [inline]
```

Computes the vector mormal. Vec.normal(), divides x and y components by magnitude.

Returns

Normal 2D vector.

6.14.3.5 normalise()

```
void math::Vector2::normalise ( )  [inline]
```

Normalises the vector. Vec.normal(), divides x and y components by magnitude.

6.14.3.6 operator∗()

```
Vector2 math::Vector2::operator* (
            double value )  [inline]
```

Overloads ∗ operator. Vec∗Vec, multiplies the x and y components of the 2D vectors.

Parameters

in	*value*	The double value used in the operation

Returns

Result of the multiplication.

6.14.3.7 operator*=()

```
void math::Vector2::operator*= (
            double value )  [inline]
```

Overloads *= operator. Vec*=Vec, multiplies the x and y components of the 2D vectors.

Parameters

in	*value*	The double value used in the operation

6.14.3.8 operator+()

```
Vector2 math::Vector2::operator+ (
            const Vector2 & other )  [inline]
```

Overloads + operator. Vec+Vec, sum of the x and y components of the 2D vector.

Parameters

other	The other vector used in the operation

Returns

Result of the addition.

6.14.3.9 operator+=()

```
void math::Vector2::operator+= (
            const Vector2 & other )  [inline]
```

Overloads += operator. Vec+=Vec, sum of the x and y components.

Parameters

other	The other vector used in the operation

6.14.3.10 operator-()

```
Vector2 math::Vector2::operator- (
            const Vector2 & other )  [inline]
```

Overloads - operator. Vec-Vec, subtraction of x and y components.

Parameters

other	The other vector used in the operation

Returns

Result of the subtraction.

6.14.3.11 operator-=()

```
void math::Vector2::operator-= (
            const Vector2 & other )  [inline]
```

Overloads -= operator. Vec-=Vec, subtraction of x and y components.

Parameters

other	The other vector used in the operation

6.14.3.12 operator/()

```
Vector2 math::Vector2::operator/ (
            double value )  [inline]
```

Overloads / operator. Vec/Vec, divide left vector by right vector.

Parameters

in	*value*	The double value used in the operation

Returns

Result of the division.

6.14.3.13 operator/=()

```
void math::Vector2::operator/= (
            double value )  [inline]
```

Overloads /= operator. Vec/=Vec, divide left vector by right vector.

Parameters

in	*value*	The double value used in the operation

6.14.3.14 operator=()

```
void math::Vector2::operator= (
            const Vector2 & other )  [inline]
```

Overloads = operator. Vec=Vec, assign x and y components to left hand side 2D vector.

Parameters

in	*other*	The other vector used in the operation

6.14.3.15 operator==()

```
bool math::Vector2::operator== (
            const Vector2 & other )  [inline]
```

Overloads == operator. Vec==Vec, compares if x and y components are the same.

Parameters

other	The other vector used in the operation

Returns

True if vectors are the same, False otherwise.

6.14.4 Member Data Documentation

6.14.4.1 x

```
double math::Vector2::x
```

6.14.4.2 y

```
double math::Vector2::y
```

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

- libs/opengl-engine/libs/math-library/src/Vector2.hpp

6.15 math::Vector3 Class Reference

Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.↩ magnitudeSquare()

```
#include <Vector3.hpp>
```

Public Member Functions

- Vector3 ()
- Vector3 (double x, double y, double z)

  Constructor. Uses the x, y and z provided to build the 3D vector.
- ∼Vector3 ()

  Destroys the object.
- Vector3 operator+ (const Vector3 &other)

  Overloads + operator. Vec+Vec, sum of the x, y and z components of the 3D vector.
- void operator+= (const Vector3 &other)

  Overloads += operator. Vec+=Vec, sum of the x, y and z components.
- Vector3 operator- (const Vector3 &other)

  Overloads - operator. Vec-Vec, subtraction of x, y and z components.
- void operator-= (const Vector3 &other)

  Overloads -= operator. Vec-=Vec, subtraction of x, y and z components.
- void operator= (const Vector3 &other)

  Overloads equal operator. Vec=Vec, assign x, y and z components to left hand side 3D vector.
- bool operator== (const Vector3 &other)

  Overloads == operator. Vec==Vec, compares if x, y and z components are the same.
- Vector3 operator∗ (double other)

  Overloads multiplication with double. Vec∗Vec, multiplies the x, y and z components of the 3D vectors.
- void operator∗= (double other)

  Overloads ∗= operator. Vec∗=Vec, multiplies the x, y and z components of the 3D vectors.
- Vector3 operator/ (double other)

  Overloads division with double. Vec/Vec, divide left vector by right vector.
- void operator/= (double other)

  Overloads /= operator. Vec/=Vec, divide left vector by right vector.
- double dot (const Vector3 &other)

  Dot product. Vec.dot(Vec), result of the dot product operation. V1 is first vector and V2 second vector, DotProduct↩ Result = V1.x ∗ V2.x + V1.y ∗ V2.y + V1.z ∗ V2.z.
- Vector3 cross (const Vector3 &other)

  Cross product. V1 is the first vector and V2 is the second vector, CrossProduct = [V1.y∗V2.z - V1.z∗V2.y, V1.z∗V2.x - V1.x∗V2.z, V1.x∗V2.y-V1.y∗V2.x].

- Vector3 normal ()

  Computes the vector mormal. Vec.normal(), divides x, y and z components by magnitude.
- void normalise ()

  Normalises the vector. Vec.normal(), divides x, y and z components by magnitude.
- double magnitude ()

  Computes the vector magnitude. Magnitude = sqrt(x∗x + y∗y + z∗z).
- double magnitudeSquared ()

  Computes the magnitude squared. MagnitudeSquared = x∗x + y∗y + z∗z.

Public Attributes

- double x
- double y
- double z

6.15.1 Detailed Description

Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.↩ magnitudeSquare()

6.15.2 Constructor & Destructor Documentation

6.15.2.1 Vector3() [1/2]

```
math::Vector3::Vector3 ( )  [inline]
```

Default constructor. Empty construcor sets x=0, y=0 and z=0.

6.15.2.2 Vector3() [2/2]

```
math::Vector3::Vector3 (
            double x,
            double y,
            double z )  [inline]
```

Constructor. Uses the x, y and z provided to build the 3D vector.

Parameters

in	*x*	x value
in	*y*	y value
in	*z*	z value

6.15.2.3 ~Vector3()

```
math::Vector3::~Vector3 ( )  [inline]
```

Destroys the object.

6.15.3 Member Function Documentation

6.15.3.1 cross()

```
Vector3 math::Vector3::cross (
            const Vector3 & other )  [inline]
```

Cross product. V1 is the first vector and V2 is the second vector, CrossProduct = [V1.y∗V2.z - V1.z∗V2.y, V1.z∗V2.x - V1.x∗V2.z, V1.x∗V2.y-V1.y∗V2.x].

Parameters

in	*other*	The other Vector3 used in the operation

Returns

Result of the cross product.

6.15.3.2 dot()

```
double math::Vector3::dot (
            const Vector3 & other )  [inline]
```

Dot product. Vec.dot(Vec), result of the dot product operation. V1 is first vector and V2 second vector, DotProduct↩ Result = V1.x ∗ V2.x + V1.y ∗ V2.y + V1.z ∗ V2.z.

Parameters

other	The other vector

Returns

Result of the dot product.

6.15.3.3 magnitude()

```
double math::Vector3::magnitude ( )  [inline]
```

Computes the vector magnitude. Magnitude = sqrt(x∗x + y∗y + z∗z).

Returns

Magnitude computed.

6.15.3.4 magnitudeSquared()

```
double math::Vector3::magnitudeSquared ( )  [inline]
```

Computes the magnitude squared. MagnitudeSquared = x∗x + y∗y + z∗z.

Returns

Magnitude squared.

6.15.3.5 normal()

```
Vector3 math::Vector3::normal ( )  [inline]
```

Computes the vector mormal. Vec.normal(), divides x, y and z components by magnitude.

Returns

Normal 3D vector.

6.15.3.6 normalise()

```
void math::Vector3::normalise ( )  [inline]
```

Normalises the vector. Vec.normal(), divides x, y and z components by magnitude.

6.15.3.7 operator∗()

```
Vector3 math::Vector3::operator* (
            double other )  [inline]
```

Overloads multiplication with double. Vec∗Vec, multiplies the x, y and z components of the 3D vectors.

Parameters

in	*other*	Double value used in the operation

Returns

Multiplication result.

6.15.3.8 operator*=()

```
void math::Vector3::operator*= (
            double other )  [inline]
```

Overloads *= operator. Vec*=Vec, multiplies the x, y and z components of the 3D vectors.

Parameters

in	*other*	Double value used in the operation

6.15.3.9 operator+()

```
Vector3 math::Vector3::operator+ (
            const Vector3 & other )  [inline]
```

Overloads + operator. Vec+Vec, sum of the x, y and z components of the 3D vector.

Parameters

in	*other*	The other Vector3 used in the operation

Returns

Sum of the two vectors.

6.15.3.10 operator+=()

```
void math::Vector3::operator+= (
            const Vector3 & other )  [inline]
```

Overloads += operator. Vec+=Vec, sum of the x, y and z components.

Parameters

in	*other*	The other Vector3 used in the operation

6.15.3.11 operator-()

```
Vector3 math::Vector3::operator- (
            const Vector3 & other )  [inline]
```

Overloads - operator. Vec-Vec, subtraction of x, y and z components.

Parameters

in	*other*	The other Vector3 used in the operation

Returns

Substraction of the two vectors.

6.15.3.12 operator-=()

```
void math::Vector3::operator-= (
            const Vector3 & other )  [inline]
```

Overloads -= operator. Vec-=Vec, subtraction of x, y and z components.

Parameters

in	*other*	The other Vector3 used in the operation

6.15.3.13 operator/()

```
Vector3 math::Vector3::operator/ (
            double other )  [inline]
```

Overloads division with double. Vec/Vec, divide left vector by right vector.

Parameters

in	*other*	Double value used in the operation

Returns

Division result.

6.15.3.14 operator/=()

```
void math::Vector3::operator/= (
            double other )  [inline]
```

Overloads /= operator. Vec/=Vec, divide left vector by right vector.

Parameters

in	*other*	Double value used in the operation

6.15.3.15 operator=()

```
void math::Vector3::operator= (
            const Vector3 & other )  [inline]
```

Overloads equal operator. Vec=Vec, assign x, y and z components to left hand side 3D vector.

Parameters

in	*other*	The other Vector3 used in the operation

6.15.3.16 operator==()

```
bool math::Vector3::operator== (
            const Vector3 & other )  [inline]
```

Overloads == operator. Vec==Vec, compares if x, y and z components are the same.

Parameters

other	The other vector

Returns

True if vectors are the same, False otherwise.

6.15.4 Member Data Documentation

6.15.4.1 x

```
double math::Vector3::x
```

6.15.4.2 y

```
double math::Vector3::y
```

6.15.4.3 z

```
double math::Vector3::z
```

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

- libs/opengl-engine/libs/math-library/src/Vector3.hpp

6.16 VkDescriptions Struct Reference

Vulkan variables descriptions. Structure containing the descriptions of the Vulkan variables.

```
#include <vkdescriptions.hpp>
```

Public Attributes

- QString maxImageDimension1D = "maxImageDimension1D is the maximum dimension (width) supported for all images created with an imageType of VK_IMAGE_TYPE_1D."
- QString maxImageDimension2D = "maxImageDimension2D is the maximum dimension (width or height) supported for all images created with an imageType of VK_IMAGE_TYPE_2D and without VK_IMAGE_CREA↩ TE_CUBE_COMPATIBLE_BIT set in flags."
- QString maxImageDimension3D = "maxImageDimension3D is the maximum dimension (width, height, or depth) supported for all images created with an imageType of VK_IMAGE_TYPE_3D."
- QString maxImageDimensionCube = "maxImageDimensionCube is the maximum dimension (width or height) supported for all images created with an imageType of VK_IMAGE_TYPE_2D and with VK_IMAGE_CRE↩ ATE_CUBE_COMPATIBLE_BIT set in flags."
- QString maxImageArrayLayers = "maxImageArrayLayers is the maximum number of layers (arrayLayers) for an image."
- QString maxTexelBufferElements = "maxTexelBufferElements is the maximum number of addressable texels for a buffer view created on a buffer which was created with the VK_BUFFER_USAGE_UNIFORM_TEXEL↩ _BUFFER_BIT or VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT set in the usage member of the VkBufferCreateInfo structure."

- QString maxUniformBufferRange = "maxUniformBufferRange is the maximum value that can be specified in the range member of any VkDescriptorBufferInfo structures passed to a call to vkUpdateDescriptorSets for descriptors of type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFO↩ RM_BUFFER_DYNAMIC."
- QString maxStorageBufferRange = "maxStorageBufferRange is the maximum value that can be specified in the range member of any VkDescriptorBufferInfo structures passed to a call to vkUpdateDescriptorSets for descriptors of type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STOR↩ AGE_BUFFER_DYNAMIC."
- QString maxPushConstantsSize = "maxPushConstantsSize is the maximum size, in bytes, of the pool of push constant memory. For each of the push constant ranges indicated by the pPushConstantRanges member of the VkPipelineLayoutCreateInfo structure, (offset + size) must be less than or equal to this limit."
- QString maxMemoryAllocationCount = "maxMemoryAllocationCount is the maximum number of device memory allocations, as created by vkAllocateMemory, which can simultaneously exist."
- QString maxSamplerAllocationCount = "maxSamplerAllocationCount is the maximum number of sampler objects, as created by vkCreateSampler, which can simultaneously exist on a device."
- QString bufferImageGranularity = "bufferImageGranularity is the granularity, in bytes, at which buffer or linear image resources, and optimal image resources can be bound to adjacent offsets in the same VkDevice↩ Memory object without aliasing. See Buffer-Image Granularity for more details."
- QString sparseAddressSpaceSize = "sparseAddressSpaceSize is the total amount of address space available, in bytes, for sparse memory resources. This is an upper bound on the sum of the size of all sparse resources, regardless of whether any memory is bound to them."
- QString maxBoundDescriptorSets = "maxBoundDescriptorSets is the maximum number of descriptor sets that can be simultaneously used by a pipeline. All DescriptorSet decorations in shader modules must have a value less than maxBoundDescriptorSets."
- QString maxPerStageDescriptorSamplers = "maxPerStageDescriptorSamplers is the maximum number of samplers that can be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_SAMPLER or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_L↩ AYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit. A descriptor is accessible to a shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set."
- QString maxPerStageDescriptorUniformBuffers = "maxPerStageDescriptorUniformBuffers is the maximum number of uniform buffers that can be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_B↩ UFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the V↩ K_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit. A descriptor is accessible to a shader stage when the stageFlags member of the VkDescriptor↩ SetLayoutBinding structure has the bit for that shader stage set."
- QString maxPerStageDescriptorStorageBuffers = "maxPerStageDescriptorStorageBuffers is the maximum number of storage buffers that can be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUF↩ FER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK↩ _DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the stageFlags member of the Vk↩ DescriptorSetLayoutBinding structure has the bit for that shader stage set."
- QString maxPerStageDescriptorSampledImages = "maxPerStageDescriptorSampledImages is the maximum number of sampled images that can be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SA↩ MPLED_IMAGE, or VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UP↩ DATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set."
- QString maxPerStageDescriptorStorageImages = "maxPerStageDescriptorStorageImages is the maximum number of storage images that can be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_STORAGE_T↩ EXEL_BUFFER count against this limit. Only descriptors in descriptor set layouts created without the VK↩ _DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against

this limit. A descriptor is accessible to a pipeline shader stage when the stageFlags member of the Vk↩ DescriptorSetLayoutBinding structure has the bit for that shader stage set."

- QString maxPerStageDescriptorInputAttachments = "maxPerStageDescriptorInputAttachments is the maximum number of input attachments that can be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UP↩ DATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit. A descriptor is accessible to a pipeline shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set. These are only supported for the fragment stage."
- QString maxPerStageResources = "maxPerStageResources is the maximum number of resources that can be accessible to a single shader stage in a pipeline layout. Descriptors with a type of VK_DESCRIPTO↩ R_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, VK_DESCRI↩ PTOR_TYPE_STORAGE_IMAGE, VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, VK_DESCR↩ IPTOR_TYPE_STORAGE_TEXEL_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DES↩ CRIPTOR_TYPE_STORAGE_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, VK↩ _DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, or VK_DESCRIPTOR_TYPE_INPUT_ATTACH↩ MENT count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCR↩ IPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit. For the fragment shader stage the framebuffer color attachments also count against this limit."
- QString maxDescriptorSetSamplers = "maxDescriptorSetSamplers is the maximum number of samplers that can be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_SAMPLER or VK_DESCRIPTOR_TYP↩ E_COMBINED_IMAGE_SAMPLER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
- QString maxDescriptorSetUniformBuffers = "maxDescriptorSetUniformBuffers is the maximum number of uniform buffers that can be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_↩ BIND_POOL_BIT_EXT bit set count against this limit."
- QString maxDescriptorSetUniformBuffersDynamic = "maxDescriptorSetUniformBuffersDynamic is the maximum number of dynamic uniform buffers that can be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYP↩ E_UNIFORM_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
- QString maxDescriptorSetStorageBuffers = "maxDescriptorSetStorageBuffers is the maximum number of storage buffers that can be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUF↩ FER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFT↩ ER_BIND_POOL_BIT_EXT bit set count against this limit."
- QString maxDescriptorSetStorageBuffersDynamic = "maxDescriptorSetStorageBuffersDynamic is the maximum number of dynamic storage buffers that can be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYP↩ E_STORAGE_BUFFER_DYNAMIC count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
- QString maxDescriptorSetSampledImages = "maxDescriptorSetSampledImages is the maximum number of sampled images that can be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_COMBINED_IM↩ AGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, or VK_DESCRIPTOR_TYPE_UNIFOR↩ M_TEXEL_BUFFER count against this limit. Only descriptors in descriptor set layouts created without the V↩ K_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
- QString maxDescriptorSetStorageImages = "maxDescriptorSetStorageImages is the maximum number of storage images that can be included in descriptor bindings in a pipeline layout across all pipeline shader

stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_IMA↩ GE, or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER count against this limit. Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER↩ _BIND_POOL_BIT_EXT bit set count against this limit."

- QString maxDescriptorSetInputAttachments = "maxDescriptorSetInputAttachments is the maximum number of input attachments that can be included in descriptor bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers. Descriptors with a type of VK_DESCRIPTOR_TYPE_INPUT_ATTA↩ CHMENT count against this limit. Only descriptors in descriptor set layouts created without the VK_DE↩ SCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
- QString maxVertexInputAttributes = "maxVertexInputAttributes is the maximum number of vertex input attributes that can be specified for a graphics pipeline. These are described in the array of VkVertexInput↩ AttributeDescription structures that are provided at graphics pipeline creation time via the pVertexAttribute↩ Descriptions member of the VkPipelineVertexInputStateCreateInfo structure."
- QString maxVertexInputBindings = "maxVertexInputBindings is the maximum number of vertex buffers that can be specified for providing vertex attributes to a graphics pipeline. These are described in the array of Vk↩ VertexInputBindingDescription structures that are provided at graphics pipeline creation time via the pVertex↩ BindingDescriptions member of the VkPipelineVertexInputStateCreateInfo structure. The binding member of VkVertexInputBindingDescription must be less than this limit."
- QString maxVertexInputAttributeOffset = "maxVertexInputAttributeOffset is the maximum vertex input attribute offset that can be added to the vertex input binding stride. The offset member of the VkVertex↩ InputAttributeDescription structure must be less than or equal to this limit."
- QString maxVertexInputBindingStride = "maxVertexInputBindingStride is the maximum vertex input binding stride that can be specified in a vertex input binding. The stride member of the VkVertexInputBinding↩ Description structure must be less than or equal to this limit."
- QString maxVertexOutputComponents = "maxVertexOutputComponents is the maximum number of components of output variables which can be output by a vertex shader."
- QString maxTessellationGenerationLevel = "maxTessellationGenerationLevel is the maximum tessellation generation level supported by the fixed-function tessellation primitive generator."
- QString maxTessellationPatchSize = "maxTessellationPatchSize is the maximum patch size, in vertices, of patches that can be processed by the tessellation control shader and tessellation primitive generator. The patchControlPoints member of the VkPipelineTessellationStateCreateInfo structure specified at pipeline creation time and the value provided in the OutputVertices execution mode of shader modules must be less than or equal to this limit."
- QString maxTessellationControlPerVertexInputComponents = "maxTessellationControlPerVertexInput↩ Components is the maximum number of components of input variables which can be provided as per-vertex inputs to the tessellation control shader stage."
- QString maxTessellationControlPerVertexOutputComponents = "maxTessellationControlPerVertexOutput↩ Components is the maximum number of components of per-vertex output variables which can be output from the tessellation control shader stage."
- QString maxTessellationControlPerPatchOutputComponents = "maxTessellationControlPerPatchOutput↩ Components is the maximum number of components of per-patch output variables which can be output from the tessellation control shader stage."
- QString maxTessellationControlTotalOutputComponents = "maxTessellationControlTotalOutputComponents is the maximum total number of components of per-vertex and per-patch output variables which can be output from the tessellation control shader stage."
- QString maxTessellationEvaluationInputComponents = "maxTessellationEvaluationInputComponents is the maximum number of components of input variables which can be provided as per-vertex inputs to the tessellation evaluation shader stage."
- QString maxTessellationEvaluationOutputComponents = "maxTessellationEvaluationOutputComponents is the maximum number of components of per-vertex output variables which can be output from the tessellation evaluation shader stage."
- QString maxGeometryShaderInvocations = "maxGeometryShaderInvocations is the maximum invocation count supported for instanced geometry shaders. The value provided in the Invocations execution mode of shader modules must be less than or equal to this limit."
- QString maxGeometryInputComponents = "maxGeometryInputComponents is the maximum number of components of input variables which can be provided as inputs to the geometry shader stage."

- QString maxGeometryOutputComponents = "maxGeometryOutputComponents is the maximum number of components of output variables which can be output from the geometry shader stage."
- QString maxGeometryOutputVertices = "maxGeometryOutputVertices is the maximum number of vertices which can be emitted by any geometry shader."
- QString maxGeometryTotalOutputComponents = "maxGeometryTotalOutputComponents is the maximum total number of components of output, across all emitted vertices, which can be output from the geometry shader stage."
- QString maxFragmentInputComponents = "maxFragmentInputComponents is the maximum number of components of input variables which can be provided as inputs to the fragment shader stage."
- QString maxFragmentOutputAttachments = "maxFragmentOutputAttachments is the maximum number of output attachments which can be written to by the fragment shader stage."
- QString maxFragmentDualSrcAttachments = "maxFragmentDualSrcAttachments is the maximum number of output attachments which can be written to by the fragment shader stage when blending is enabled and one of the dual source blend modes is in use."
- QString maxFragmentCombinedOutputResources = "maxFragmentCombinedOutputResources is the total number of storage buffers, storage images, and output buffers which can be used in the fragment shader stage."
- QString maxComputeSharedMemorySize = "maxComputeSharedMemorySize is the maximum total storage size, in bytes, available for variables declared with the Workgroup storage class in shader modules (or with the shared storage qualifier in GLSL) in the compute shader stage. The amount of storage consumed by the variables declared with the Workgroup storage class is implementation-dependent. However, the amount of storage consumed may not exceed the largest block size that would be obtained if all active variables declared with Workgroup storage class were assigned offsets in an arbitrary order by successively taking the smallest valid offset according to the Standard Storage Buffer Layout rules. (This is equivalent to using the GLSL std430 layout rules.)"
- QString maxComputeWorkGroupCountX = "maxComputeWorkGroupCount is the maximum number of local workgroups that can be dispatched by a single dispatch command. These three values represent the maximum number of local workgroups for the X, Y, and Z dimensions, respectively. The workgroup count parameters to the dispatch commands must be less than or equal to the corresponding limit."
- QString maxComputeWorkGroupCountY = "maxComputeWorkGroupCount is the maximum number of local workgroups that can be dispatched by a single dispatch command. These three values represent the maximum number of local workgroups for the X, Y, and Z dimensions, respectively. The workgroup count parameters to the dispatch commands must be less than or equal to the corresponding limit."
- QString maxComputeWorkGroupCountZ = "maxComputeWorkGroupCount is the maximum number of local workgroups that can be dispatched by a single dispatch command. These three values represent the maximum number of local workgroups for the X, Y, and Z dimensions, respectively. The workgroup count parameters to the dispatch commands must be less than or equal to the corresponding limit."
- QString maxComputeWorkGroupInvocations = "maxComputeWorkGroupInvocations is the maximum total number of compute shader invocations in a single local workgroup. The product of the X, Y, and Z sizes, as specified by the LocalSize execution mode in shader modules or by the object decorated by the WorkgroupSize decoration, must be less than or equal to this limit."
- QString maxComputeWorkGroupSizeX = "maxComputeWorkGroupSize is the maximum size of a local compute workgroup, per dimension. These three values represent the maximum local workgroup size in the X, Y, and Z dimensions, respectively. The x, y, and z sizes, as specified by the LocalSize execution mode or by the object decorated by the WorkgroupSize decoration in shader modules, must be less than or equal to the corresponding limit."
- QString maxComputeWorkGroupSizeY = "maxComputeWorkGroupSize is the maximum size of a local compute workgroup, per dimension. These three values represent the maximum local workgroup size in the X, Y, and Z dimensions, respectively. The x, y, and z sizes, as specified by the LocalSize execution mode or by the object decorated by the WorkgroupSize decoration in shader modules, must be less than or equal to the corresponding limit."
- QString maxComputeWorkGroupSizeZ = "maxComputeWorkGroupSize is the maximum size of a local compute workgroup, per dimension. These three values represent the maximum local workgroup size in the X, Y, and Z dimensions, respectively. The x, y, and z sizes, as specified by the LocalSize execution mode or by the object decorated by the WorkgroupSize decoration in shader modules, must be less than or equal to the corresponding limit."
- QString subPixelPrecisionBits = "subPixelPrecisionBits is the number of bits of subpixel precision in framebuffer coordinates xf and yf."

- QString subTexelPrecisionBits = "subTexelPrecisionBits is the number of bits of precision in the division along an axis of an image used for minification and magnification filters."
- QString mipmapPrecisionBits = "mipmapPrecisionBits is the number of bits of division that the LOD calculation for mipmap fetching get snapped to when determining the contribution from each mip level to the mip filtered results."
- QString maxDrawIndexedIndexValue = "maxDrawIndexedIndexValue is the maximum index value that can be used for indexed draw calls when using 32-bit indices. This excludes the primitive restart index value of 0xFFFFFFFF. See fullDrawIndexUint32."
- QString maxDrawIndirectCount = "maxDrawIndirectCount is the maximum draw count that is supported for indirect draw calls. See multiDrawIndirect."
- QString maxSamplerLodBias = "maxSamplerLodBias is the maximum absolute sampler LOD bias. The sum of the mipLodBias member of the VkSamplerCreateInfo structure and the Bias operand of image sampling operations in shader modules (or 0 if no Bias operand is provided to an image sampling operation) are clamped to the range [-maxSamplerLodBias,+maxSamplerLodBias]."
- QString maxSamplerAnisotropy = "maxSamplerAnisotropy is the maximum degree of sampler anisotropy. The maximum degree of anisotropic filtering used for an image sampling operation is the minimum of the maxAnisotropy member of the VkSamplerCreateInfo structure and this limit."
- QString maxViewports = "maxViewports is the maximum number of active viewports. The viewportCount member of the VkPipelineViewportStateCreateInfo structure that is provided at pipeline creation must be less than or equal to this limit."
- QString maxViewportDimensionsX = "maxViewportDimensions are the maximum viewport dimensions in the X (width) and Y (height) dimensions, respectively. The maximum viewport dimensions must be greater than or equal to the largest image which can be created and used as a framebuffer attachment. See Controlling the Viewport."
- QString maxViewportDimensionsY = "maxViewportDimensions are the maximum viewport dimensions in the X (width) and Y (height) dimensions, respectively. The maximum viewport dimensions must be greater than or equal to the largest image which can be created and used as a framebuffer attachment. See Controlling the Viewport."
- QString viewportBoundsRangeX = "viewportBoundsRange is the [minimum, maximum] range that the corners of a viewport must be contained in. This range must be at least [-2 × size, 2 × size - 1], where size = max(maxViewportDimensions[0], maxViewportDimensions[1]). See Controlling the Viewport."
- QString viewportBoundsRangeY = "viewportBoundsRange is the [minimum, maximum] range that the corners of a viewport must be contained in. This range must be at least [-2 × size, 2 × size - 1], where size = max(maxViewportDimensions[0], maxViewportDimensions[1]). See Controlling the Viewport."
- QString viewportSubPixelBits = "viewportSubPixelBits is the number of bits of subpixel precision for viewport bounds. The subpixel precision that floating - point viewport bounds are interpreted at is given by this limit."
- QString minMemoryMapAlignment = "minMemoryMapAlignment is the minimum required alignment, in bytes, of host visible memory allocations within the host address space. When mapping a memory allocation with vkMapMemory, subtracting offset bytes from the returned pointer will always produce an integer multiple of this limit."
- QString minTexelBufferOffsetAlignment = "minTexelBufferOffsetAlignment is the minimum required alignment, in bytes, for the offset member of the VkBufferViewCreateInfo structure for texel buffers. If texelBuffer↩ Alignment is enabled, this limit is equivalent to the maximum of the uniformTexelBufferOffsetAlignment↩ Bytes and storageTexelBufferOffsetAlignmentBytes members of VkPhysicalDeviceTexelBufferAlignment↩ PropertiesEXT, but smaller alignment is optionally: allowed by storageTexelBufferOffsetSingleTexelAlignment and uniformTexelBufferOffsetSingleTexelAlignment. If texelBufferAlignment is not enabled, VkBufferView↩ CreateInfo::offset must be a multiple of this value."
- QString minUniformBufferOffsetAlignment = "minUniformBufferOffsetAlignment is the minimum required alignment, in bytes, for the offset member of the VkDescriptorBufferInfo structure for uniform buffers. When a desc =riptor of type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIF↩ ORM_BUFFER_DYNAMIC is updated, the offset must be an integer multiple of this limit. Similarly, dynamic offsets for uniform buffers must be multiples of this limit."
- QString minStorageBufferOffsetAlignment = "minStorageBufferOffsetAlignment is the minimum required alignment, in bytes, for the offset member of the VkDescriptorBufferInfo structure for storage buffers. When a desc =riptor of type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STOR↩ AGE_BUFFER_DYNAMIC is updated, the offset must be an integer multiple of this limit. Similarly, dynamic offsets for storage buffers must be multiples of this limit."

- QString minTexelOffset = "minTexelOffset is the minimum offset value for the ConstOffset image operand of any of the OpImageSample∗ or OpImageFetch∗ image instructions."
- QString maxTexelOffset = "maxTexelOffset is the maximum offset value for the ConstOffset image operand of any of the OpImageSample∗ or OpImageFetch∗ image instructions."
- QString minTexelGatherOffset = "minTexelGatherOffset is the minimum offset value for the Offset or ConstOffsets image operands of any of the OpImage∗Gather image instructions."
- QString maxTexelGatherOffset = "maxTexelGatherOffset is the maximum offset value for the Offset or ConstOffsets image operands of any of the OpImage∗Gather image instructions."
- QString minInterpolationOffset = "minInterpolationOffset is the minimum negative offset value for the offset operand of the InterpolateAtOffset extended instruction."
- QString maxInterpolationOffset = "maxInterpolationOffset is the maximum positive offset value for the offset operand of the InterpolateAtOffset extended instruction."
- QString subPixelInterpolationOffsetBits = "subPixelInterpolationOffsetBits is the number of subpixel fractional bits that the x and y offsets to the InterpolateAtOffset extended instruction may be rounded to as fixed-point values."
- QString maxFramebufferWidth = "maxFramebufferWidth is the maximum width for a framebuffer. The width member of the VkFramebufferCreateInfo structure must be less than or equal to this limit."
- QString maxFramebufferHeight = "maxFramebufferHeight is the maximum height for a framebuffer. The height member of the VkFramebufferCreateInfo structure must be less than or equal to this limit."
- QString maxFramebufferLayers = "maxFramebufferLayers is the maximum layer count for a layered framebuffer. The layers member of the VkFramebufferCreateInfo structure must be less than or equal to this limit."
- QString framebufferColorSampleCounts = "framebufferColorSampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the color sample counts that are supported for all framebuffer color attachments with floating - or fixed - point formats. There is no limit that specifies the color sample counts that are supported for all color attachments with integer formats."
- QString framebufferDepthSampleCounts = "framebufferDepthSampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the supported depth sample counts for all framebuffer depth/stencil attachments, when the format includes a depth component."
- QString framebufferStencilSampleCounts = "framebufferStencilSampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the supported stencil sample counts for all framebuffer depth / stencil attachments, when the format includes a stencil component."
- QString framebufferNoAttachmentsSampleCounts = "framebufferNoAttachmentsSampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the supported sample counts for a framebuffer with no attachments."
- QString maxColorAttachments = "maxColorAttachments is the maximum number of color attachments that can be used by a subpass in a render pass. The colorAttachmentCount member of the VkSubpassDescription structure must be less than or equal to this limit."
- QString sampledImageColorSampleCounts = "sampledImageColorSampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with VK_IMAGE_TILING_OPTIMAL, usage containing VK_IMAGE_USAGE_SAMPLED_BIT, and a non - integer color format."
- QString sampledImageIntegerSampleCounts = "sampledImageIntegerSampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with VK_IMAGE_TILING_OPTIMAL, usage containing VK_IMAGE_USAGE_SAMPLED_BIT, and an integer color format."
- QString sampledImageDepthSampleCounts = "sampledImageDepthSampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with VK_IMAGE_TILING_OPTIMAL, usage containing VK_IMAGE_USAGE_SAMPLED_BIT, and a depth format."
- QString sampledImageStencilSampleCounts = "sampledImageStencilSampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the sample supported for all 2D images created with VK_IMAGE_TILING_OPTIMAL, usage containing VK_IMAGE_USAGE_SAMPLED_BIT, and a stencil format."
- QString storageImageSampleCounts = "storageImageSampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with VK_IMAGE_TILING_OPTIMAL, and usage containing VK_IMAGE_USAGE_STORAGE_BIT."
- QString maxSampleMaskWords = "maxSampleMaskWords is the maximum number of array elements of a variable decorated with the SampleMask built - in decoration."
- QString timestampComputeAndGraphics = "timestampComputeAndGraphics specifies support for timestamps on all graphics and compute queues. If this limit is set to VK_TRUE, all queues that advertise the VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT in the VkQueueFamilyProperties::queueFlags support VkQueueFamilyProperties::timestampValidBits of at least 36. See Timestamp Queries."

- QString timestampPeriod = "timestampPeriod is the number of nanoseconds required for a timestamp query to be incremented by 1. See Timestamp Queries."
- QString maxClipDistances = "maxClipDistances is the maximum number of clip distances that can be used in a single shader stage. The size of any array declared with the ClipDistance built-in decoration in a shader module must be less than or equal to this limit."
- QString maxCullDistances = "maxCullDistances is the maximum number of cull distances that can be used in a single shader stage. The size of any array declared with the CullDistance built-in decoration in a shader module must be less than or equal to this limit."
- QString maxCombinedClipAndCullDistances = "maxCombinedClipAndCullDistances is the maximum combined number of clip and cull distances that can be used in a single shader stage. The sum of the sizes of any pair of arrays declared with the ClipDistance and CullDistance built-in decoration used by a single shader stage in a shader module must be less than or equal to this limit."
- QString discreteQueuePriorities = "discreteQueuePriorities is the number of discrete priorities that can be assigned to a queue based on the value of each member of VkDeviceQueueCreateInfo::pQueuePriorities. This must be at least 2, and levels must be spread evenly over the range, with at least one level at 1.0, and another at 0.0."
- QString pointSizeRangeX = "pointSizeRange is the range [minimum,maximum] of supported sizes for points. Values written to variables decorated with the PointSize built-in decoration are clamped to this range."
- QString pointSizeRangeY = "pointSizeRange is the range [minimum,maximum] of supported sizes for points. Values written to variables decorated with the PointSize built-in decoration are clamped to this range."
- QString lineWidthRangeX = "lineWidthRange is the range [minimum,maximum] of supported widths for lines. Values specified by the lineWidth member of the VkPipelineRasterizationStateCreateInfo or the lineWidth parameter to vkCmdSetLineWidth are clamped to this range."
- QString lineWidthRangeY = "lineWidthRange is the range [minimum,maximum] of supported widths for lines. Values specified by the lineWidth member of the VkPipelineRasterizationStateCreateInfo or the lineWidth parameter to vkCmdSetLineWidth are clamped to this range."
- QString pointSizeGranularity = "pointSizeGranularity is the granularity of supported point sizes. Not all point sizes in the range defined by pointSizeRange are supported. This limit specifies the granularity (or increment) between successive supported point sizes."
- QString lineWidthGranularity = "lineWidthGranularity is the granularity of supported line widths. Not all line widths in the range defined by lineWidthRange are supported. This limit specifies the granularity (or increment) between successive supported line widths."
- QString strictLines = "strictLines specifies whether lines are rasterized according to the preferred method of rasterization. If set to VK_FALSE, lines may be rasterized under a relaxed set of rules. If set to VK_TRUE, lines are rasterized as per the strict definition. See Basic Line Segment Rasterization."
- QString standardSampleLocations = "standardSampleLocations specifies whether rasterization uses the standard sample locations as documented in Multisampling. If set to VK_TRUE, the implementation uses the documented sample locations. If set to VK_FALSE, the implementation may use different sample locations."
- QString optimalBufferCopyOffsetAlignment = "optimalBufferCopyOffsetAlignment is the optimal buffer offset alignment in bytes for vkCmdCopyBufferToImage and vkCmdCopyImageToBuffer. The per texel alignment requirements are enforced, but applications should use the optimal alignment for optimal performance and power use."
- QString optimalBufferCopyRowPitchAlignment = "optimalBufferCopyRowPitchAlignment is the optimal buffer row pitch alignment in bytes for vkCmdCopyBufferToImage and vkCmdCopyImageToBuffer. Row pitch is the number of bytes between texels with the same X coordinate in adjacent rows (Y coordinates differ by one). The per texel alignment requirements are enforced, but applications should use the optimal alignment for optimal performance and power use."
- QString nonCoherentAtomSize = "nonCoherentAtomSize is the size and alignment in bytes that bounds concurrent access to host - mapped device memory."

6.16.1 Detailed Description

Vulkan variables descriptions. Structure containing the descriptions of the Vulkan variables.

6.16.2 Member Data Documentation

6.16.2.1 bufferImageGranularity

```
QString VkDescriptions::bufferImageGranularity = "bufferImageGranularity is the granularity,
in bytes, at which buffer or linear image resources, and optimal image resources can be bound
to adjacent offsets in the same VkDeviceMemory object without aliasing.  See Buffer-Image
Granularity for more details."
```

6.16.2.2 discreteQueuePriorities

```
QString VkDescriptions::discreteQueuePriorities = "discreteQueuePriorities is the number of
discrete priorities that can be assigned to a queue based on the value of each member of Vk↩
DeviceQueueCreateInfo::pQueuePriorities.  This must be at least 2, and levels must be spread
evenly over the range, with at least one level at 1.0, and another at 0.0."
```

6.16.2.3 framebufferColorSampleCounts

```
QString VkDescriptions::framebufferColorSampleCounts = "framebufferColorSampleCounts is a
bitmask1 of VkSampleCountFlagBits indicating the color sample counts that are supported for
all framebuffer color attachments with floating - or fixed - point formats.  There is no limit
that specifies the color sample counts that are supported for all color attachments with integer
formats."
```

6.16.2.4 framebufferDepthSampleCounts

```
QString VkDescriptions::framebufferDepthSampleCounts = "framebufferDepthSampleCounts is a
bitmask1 of VkSampleCountFlagBits indicating the supported depth sample counts for all framebuffer
depth/stencil attachments, when the format includes a depth component."
```

6.16.2.5 framebufferNoAttachmentsSampleCounts

```
QString VkDescriptions::framebufferNoAttachmentsSampleCounts = "framebufferNoAttachments↩
SampleCounts is a bitmask1 of VkSampleCountFlagBits indicating the supported sample counts for
a framebuffer with no attachments."
```

6.16.2.6 framebufferStencilSampleCounts

```
QString VkDescriptions::framebufferStencilSampleCounts = "framebufferStencilSampleCounts is
a bitmask1 of VkSampleCountFlagBits indicating the supported stencil sample counts for all
framebuffer depth / stencil attachments, when the format includes a stencil component."
```

6.16.2.7 lineWidthGranularity

```
QString VkDescriptions::lineWidthGranularity = "lineWidthGranularity is the granularity of
supported line widths.  Not all line widths in the range defined by lineWidthRange are supported.
This limit specifies the granularity (or increment) between successive supported line widths."
```

6.16.2.8 lineWidthRangeX

```
QString VkDescriptions::lineWidthRangeX = "lineWidthRange is the range [minimum,maximum] of
supported widths for lines.  Values specified by the lineWidth member of the VkPipeline↩
RasterizationStateCreateInfo or the lineWidth parameter to vkCmdSetLineWidth are clamped to
this range."
```

6.16.2.9 lineWidthRangeY

```
QString VkDescriptions::lineWidthRangeY = "lineWidthRange is the range [minimum,maximum] of
supported widths for lines.  Values specified by the lineWidth member of the VkPipeline↩
RasterizationStateCreateInfo or the lineWidth parameter to vkCmdSetLineWidth are clamped to
this range."
```

6.16.2.10 maxBoundDescriptorSets

```
QString VkDescriptions::maxBoundDescriptorSets = "maxBoundDescriptorSets is the maximum number
of descriptor sets that can be simultaneously used by a pipeline.  All DescriptorSet decorations
in shader modules must have a value less than maxBoundDescriptorSets."
```

6.16.2.11 maxClipDistances

```
QString VkDescriptions::maxClipDistances = "maxClipDistances is the maximum number of clip
distances that can be used in a single shader stage.  The size of any array declared with the
ClipDistance built-in decoration in a shader module must be less than or equal to this limit."
```

6.16.2.12 maxColorAttachments

```
QString VkDescriptions::maxColorAttachments = "maxColorAttachments is the maximum number of
color attachments that can be used by a subpass in a render pass.  The colorAttachmentCount
member of the VkSubpassDescription structure must be less than or equal to this limit."
```

6.16.2.13 maxCombinedClipAndCullDistances

```
QString VkDescriptions::maxCombinedClipAndCullDistances = "maxCombinedClipAndCullDistances is
the maximum combined number of clip and cull distances that can be used in a single shader
stage.  The sum of the sizes of any pair of arrays declared with the ClipDistance and Cull↩
Distance built-in decoration used by a single shader stage in a shader module must be less
than or equal to this limit."
```

6.16.2.14 maxComputeSharedMemorySize

```
QString VkDescriptions::maxComputeSharedMemorySize = "maxComputeSharedMemorySize is the maximum
total storage size, in bytes, available for variables declared with the Workgroup storage
class in shader modules (or with the shared storage qualifier in GLSL) in the compute shader
stage.  The amount of storage consumed by the variables declared with the Workgroup storage
class is implementation-dependent.  However, the amount of storage consumed may not exceed
the largest block size that would be obtained if all active variables declared with Workgroup
storage class were assigned offsets in an arbitrary order by successively taking the smallest
valid offset according to the Standard Storage Buffer Layout rules.  (This is equivalent to
using the GLSL std430 layout rules.)"
```

6.16.2.15 maxComputeWorkGroupCountX

```
QString VkDescriptions::maxComputeWorkGroupCountX = "maxComputeWorkGroupCount is the maximum
number of local workgroups that can be dispatched by a single dispatch command.  These three
values represent the maximum number of local workgroups for the X, Y, and Z dimensions, respectively.
The workgroup count parameters to the dispatch commands must be less than or equal to the
corresponding limit."
```

6.16.2.16 maxComputeWorkGroupCountY

```
QString VkDescriptions::maxComputeWorkGroupCountY = "maxComputeWorkGroupCount is the maximum
number of local workgroups that can be dispatched by a single dispatch command.  These three
values represent the maximum number of local workgroups for the X, Y, and Z dimensions, respectively.
The workgroup count parameters to the dispatch commands must be less than or equal to the
corresponding limit."
```

6.16.2.17 maxComputeWorkGroupCountZ

```
QString VkDescriptions::maxComputeWorkGroupCountZ = "maxComputeWorkGroupCount is the maximum
number of local workgroups that can be dispatched by a single dispatch command.  These three
values represent the maximum number of local workgroups for the X, Y, and Z dimensions, respectively.
The workgroup count parameters to the dispatch commands must be less than or equal to the
corresponding limit."
```

6.16.2.18 maxComputeWorkGroupInvocations

```
QString VkDescriptions::maxComputeWorkGroupInvocations = "maxComputeWorkGroupInvocations is
the maximum total number of compute shader invocations in a single local workgroup.  The product
of the X, Y, and Z sizes, as specified by the LocalSize execution mode in shader modules or
by the object decorated by the WorkgroupSize decoration, must be less than or equal to this
limit."
```

6.16.2.19 maxComputeWorkGroupSizeX

```
QString VkDescriptions::maxComputeWorkGroupSizeX = "maxComputeWorkGroupSize is the maximum
size of a local compute workgroup, per dimension.  These three values represent the maximum
local workgroup size in the X, Y, and Z dimensions, respectively.  The x, y, and z sizes,
as specified by the LocalSize execution mode or by the object decorated by the WorkgroupSize
decoration in shader modules, must be less than or equal to the corresponding limit."
```

6.16.2.20 maxComputeWorkGroupSizeY

```
QString VkDescriptions::maxComputeWorkGroupSizeY = "maxComputeWorkGroupSize is the maximum
size of a local compute workgroup, per dimension.  These three values represent the maximum
local workgroup size in the X, Y, and Z dimensions, respectively.  The x, y, and z sizes,
as specified by the LocalSize execution mode or by the object decorated by the WorkgroupSize
decoration in shader modules, must be less than or equal to the corresponding limit."
```

6.16.2.21 maxComputeWorkGroupSizeZ

```
QString VkDescriptions::maxComputeWorkGroupSizeZ = "maxComputeWorkGroupSize is the maximum
size of a local compute workgroup, per dimension.  These three values represent the maximum
local workgroup size in the X, Y, and Z dimensions, respectively.  The x, y, and z sizes,
as specified by the LocalSize execution mode or by the object decorated by the WorkgroupSize
decoration in shader modules, must be less than or equal to the corresponding limit."
```

6.16.2.22 maxCullDistances

```
QString VkDescriptions::maxCullDistances = "maxCullDistances is the maximum number of cull
distances that can be used in a single shader stage.  The size of any array declared with the
CullDistance built-in decoration in a shader module must be less than or equal to this limit."
```

6.16.2.23 maxDescriptorSetInputAttachments

```
QString VkDescriptions::maxDescriptorSetInputAttachments = "maxDescriptorSetInputAttachments
is the maximum number of input attachments that can be included in descriptor bindings in a
pipeline layout across all pipeline shader stages and descriptor set numbers.  Descriptors
with a type of VK_DESCRIPTOR_TYPE_INPUT_ATTACHMENT count against this limit.  Only descriptors
in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_B↩
IND_POOL_BIT_EXT bit set count against this limit."
```

6.16.2.24 maxDescriptorSetSampledImages

```
QString VkDescriptions::maxDescriptorSetSampledImages = "maxDescriptorSetSampledImages is the
maximum number of sampled images that can be included in descriptor bindings in a pipeline
layout across all pipeline shader stages and descriptor set numbers.  Descriptors with a type
of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, or VK_DESCR↩
IPTOR_TYPE_UNIFORM_TEXEL_BUFFER count against this limit.  Only descriptors in descriptor set
layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit
set count against this limit."
```

6.16.2.25 maxDescriptorSetSamplers

```
QString VkDescriptions::maxDescriptorSetSamplers = "maxDescriptorSetSamplers is the maximum
number of samplers that can be included in descriptor bindings in a pipeline layout across all
pipeline shader stages and descriptor set numbers.  Descriptors with a type of VK_DESCRIPT↩
OR_TYPE_SAMPLER or VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER count against this limit.  Only
descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UP↩
DATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
```

6.16.2.26 maxDescriptorSetStorageBuffers

```
QString VkDescriptions::maxDescriptorSetStorageBuffers = "maxDescriptorSetStorageBuffers is
the maximum number of storage buffers that can be included in descriptor bindings in a pipeline
layout across all pipeline shader stages and descriptor set numbers.  Descriptors with a type
of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against
this limit.  Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SE↩
T_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
```

6.16.2.27 maxDescriptorSetStorageBuffersDynamic

```
QString VkDescriptions::maxDescriptorSetStorageBuffersDynamic = "maxDescriptorSetStorage↩
BuffersDynamic is the maximum number of dynamic storage buffers that can be included in descriptor
bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers.
Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit.
Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREA↩
TE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
```

6.16.2.28 maxDescriptorSetStorageImages

```
QString VkDescriptions::maxDescriptorSetStorageImages = "maxDescriptorSetStorageImages is the
maximum number of storage images that can be included in descriptor bindings in a pipeline
layout across all pipeline shader stages and descriptor set numbers.  Descriptors with a type
of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_DESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER count against
this limit.  Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SE↩
T_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
```

6.16.2.29 maxDescriptorSetUniformBuffers

```
QString VkDescriptions::maxDescriptorSetUniformBuffers = "maxDescriptorSetUniformBuffers is
the maximum number of uniform buffers that can be included in descriptor bindings in a pipeline
layout across all pipeline shader stages and descriptor set numbers.  Descriptors with a type
of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against
this limit.  Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SE↩
T_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
```

6.16.2.30 maxDescriptorSetUniformBuffersDynamic

```
QString VkDescriptions::maxDescriptorSetUniformBuffersDynamic = "maxDescriptorSetUniform↩
BuffersDynamic is the maximum number of dynamic uniform buffers that can be included in descriptor
bindings in a pipeline layout across all pipeline shader stages and descriptor set numbers.
Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against this limit.
Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREA↩
TE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit."
```

6.16.2.31 maxDrawIndexedIndexValue

```
QString VkDescriptions::maxDrawIndexedIndexValue = "maxDrawIndexedIndexValue is the maximum
index value that can be used for indexed draw calls when using 32-bit indices.  This excludes
the primitive restart index value of 0xFFFFFFFF. See fullDrawIndexUint32."
```

6.16.2.32 maxDrawIndirectCount

```
QString VkDescriptions::maxDrawIndirectCount = "maxDrawIndirectCount is the maximum draw count
that is supported for indirect draw calls.  See multiDrawIndirect."
```

6.16.2.33 maxFragmentCombinedOutputResources

```
QString VkDescriptions::maxFragmentCombinedOutputResources = "maxFragmentCombinedOutput↩
Resources is the total number of storage buffers, storage images, and output buffers which can
be used in the fragment shader stage."
```

6.16.2.34 maxFragmentDualSrcAttachments

```
QString VkDescriptions::maxFragmentDualSrcAttachments = "maxFragmentDualSrcAttachments is the
maximum number of output attachments which can be written to by the fragment shader stage when
blending is enabled and one of the dual source blend modes is in use."
```

6.16.2.35 maxFragmentInputComponents

```
QString VkDescriptions::maxFragmentInputComponents = "maxFragmentInputComponents is the maximum
number of components of input variables which can be provided as inputs to the fragment shader
stage."
```

6.16.2.36 maxFragmentOutputAttachments

```
QString VkDescriptions::maxFragmentOutputAttachments = "maxFragmentOutputAttachments is the
maximum number of output attachments which can be written to by the fragment shader stage."
```

6.16.2.37 maxFramebufferHeight

```
QString VkDescriptions::maxFramebufferHeight = "maxFramebufferHeight is the maximum height for
a framebuffer.  The height member of the VkFramebufferCreateInfo structure must be less than
or equal to this limit."
```

6.16.2.38 maxFramebufferLayers

```
QString VkDescriptions::maxFramebufferLayers = "maxFramebufferLayers is the maximum layer
count for a layered framebuffer.  The layers member of the VkFramebufferCreateInfo structure
must be less than or equal to this limit."
```

6.16.2.39 maxFramebufferWidth

```
QString VkDescriptions::maxFramebufferWidth = "maxFramebufferWidth is the maximum width for a
framebuffer.  The width member of the VkFramebufferCreateInfo structure must be less than or
equal to this limit."
```

6.16.2.40 maxGeometryInputComponents

```
QString VkDescriptions::maxGeometryInputComponents = "maxGeometryInputComponents is the maximum
number of components of input variables which can be provided as inputs to the geometry shader
stage."
```

6.16.2.41 maxGeometryOutputComponents

```
QString VkDescriptions::maxGeometryOutputComponents = "maxGeometryOutputComponents is the
maximum number of components of output variables which can be output from the geometry shader
stage."
```

6.16.2.42 maxGeometryOutputVertices

```
QString VkDescriptions::maxGeometryOutputVertices = "maxGeometryOutputVertices is the maximum
number of vertices which can be emitted by any geometry shader."
```

6.16.2.43 maxGeometryShaderInvocations

```
QString VkDescriptions::maxGeometryShaderInvocations = "maxGeometryShaderInvocations is the
maximum invocation count supported for instanced geometry shaders.  The value provided in the
Invocations execution mode of shader modules must be less than or equal to this limit."
```

6.16.2.44 maxGeometryTotalOutputComponents

```
QString VkDescriptions::maxGeometryTotalOutputComponents = "maxGeometryTotalOutputComponents
is the maximum total number of components of output, across all emitted vertices, which can be
output from the geometry shader stage."
```

6.16.2.45 maxImageArrayLayers

```
QString VkDescriptions::maxImageArrayLayers = "maxImageArrayLayers is the maximum number of
layers (arrayLayers) for an image."
```

6.16.2.46 maxImageDimension1D

```
QString VkDescriptions::maxImageDimension1D = "maxImageDimension1D is the maximum dimension
(width) supported for all images created with an imageType of VK_IMAGE_TYPE_1D."
```

6.16.2.47 maxImageDimension2D

```
QString VkDescriptions::maxImageDimension2D = "maxImageDimension2D is the maximum dimension
(width or height) supported for all images created with an imageType of VK_IMAGE_TYPE_2D and
without VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT set in flags."
```

6.16.2.48 maxImageDimension3D

```
QString VkDescriptions::maxImageDimension3D = "maxImageDimension3D is the maximum dimension
(width, height, or depth) supported for all images created with an imageType of VK_IMAGE_TY↩
PE_3D."
```

6.16.2.49 maxImageDimensionCube

```
QString VkDescriptions::maxImageDimensionCube = "maxImageDimensionCube is the maximum dimension
(width or height) supported for all images created with an imageType of VK_IMAGE_TYPE_2D and
with VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT set in flags."
```

6.16.2.50 maxInterpolationOffset

```
QString VkDescriptions::maxInterpolationOffset = "maxInterpolationOffset is the maximum positive
offset value for the offset operand of the InterpolateAtOffset extended instruction."
```

6.16.2.51 maxMemoryAllocationCount

```
QString VkDescriptions::maxMemoryAllocationCount = "maxMemoryAllocationCount is the maximum
number of device memory allocations, as created by vkAllocateMemory, which can simultaneously
exist."
```

6.16.2.52 maxPerStageDescriptorInputAttachments

```
QString VkDescriptions::maxPerStageDescriptorInputAttachments = "maxPerStageDescriptorInput↩
Attachments is the maximum number of input attachments that can be accessible to a single
shader stage in a pipeline layout.  Descriptors with a type of VK_DESCRIPTOR_TYPE_INPUT_ATT↩
ACHMENT count against this limit.  Only descriptors in descriptor set layouts created without
the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this
limit.  A descriptor is accessible to a pipeline shader stage when the stageFlags member of
the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set.  These are
only supported for the fragment stage."
```

6.16.2.53 maxPerStageDescriptorSampledImages

```
QString VkDescriptions::maxPerStageDescriptorSampledImages = "maxPerStageDescriptorSampled↩
Images is the maximum number of sampled images that can be accessible to a single shader stage
in a pipeline layout.  Descriptors with a type of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPL↩
ER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE, or VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER count against
this limit.  Only descriptors in descriptor set layouts created without the VK_DESCRIPTOR_SE↩
T_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set count against this limit.  A descriptor
is accessible to a pipeline shader stage when the stageFlags member of the VkDescriptorSet↩
LayoutBinding structure has the bit for that shader stage set."
```

6.16.2.54 maxPerStageDescriptorSamplers

```
QString VkDescriptions::maxPerStageDescriptorSamplers = "maxPerStageDescriptorSamplers is
the maximum number of samplers that can be accessible to a single shader stage in a pipeline
layout.  Descriptors with a type of VK_DESCRIPTOR_TYPE_SAMPLER or VK_DESCRIPTOR_TYPE_COM↩
BINED_IMAGE_SAMPLER count against this limit.  Only descriptors in descriptor set layouts
created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit set
count against this limit.  A descriptor is accessible to a shader stage when the stageFlags
member of the VkDescriptorSetLayoutBinding structure has the bit for that shader stage set."
```

6.16.2.55 maxPerStageDescriptorStorageBuffers

```
QString VkDescriptions::maxPerStageDescriptorStorageBuffers = "maxPerStageDescriptorStorage↩
Buffers is the maximum number of storage buffers that can be accessible to a single shader
stage in a pipeline layout.  Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_BUFFER
or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC count against this limit.  Only descriptors in
descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BI↩
ND_POOL_BIT_EXT bit set count against this limit.  A descriptor is accessible to a pipeline
shader stage when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the
bit for that shader stage set."
```

6.16.2.56 maxPerStageDescriptorStorageImages

```
QString VkDescriptions::maxPerStageDescriptorStorageImages = "maxPerStageDescriptorStorage↩
Images is the maximum number of storage images that can be accessible to a single shader stage
in a pipeline layout.  Descriptors with a type of VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, or VK_D↩
ESCRIPTOR_TYPE_STORAGE_TEXEL_BUFFER count against this limit.  Only descriptors in descriptor
set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_↩
EXT bit set count against this limit.  A descriptor is accessible to a pipeline shader stage
when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that
shader stage set."
```

6.16.2.57 maxPerStageDescriptorUniformBuffers

```
QString VkDescriptions::maxPerStageDescriptorUniformBuffers = "maxPerStageDescriptorUniform↩
Buffers is the maximum number of uniform buffers that can be accessible to a single shader
stage in a pipeline layout.  Descriptors with a type of VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER
or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC count against this limit.  Only descriptors in
descriptor set layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIN↩
D_POOL_BIT_EXT bit set count against this limit.  A descriptor is accessible to a shader stage
when the stageFlags member of the VkDescriptorSetLayoutBinding structure has the bit for that
shader stage set."
```

6.16.2.58 maxPerStageResources

```
QString VkDescriptions::maxPerStageResources = "maxPerStageResources is the maximum number of
resources that can be accessible to a single shader stage in a pipeline layout.  Descriptors
with a type of VK_DESCRIPTOR_TYPE_COMBINED_IMAGE_SAMPLER, VK_DESCRIPTOR_TYPE_SAMPLED_IMAGE,
VK_DESCRIPTOR_TYPE_STORAGE_IMAGE, VK_DESCRIPTOR_TYPE_UNIFORM_TEXEL_BUFFER, VK_DESCRIPTOR_TY↩
PE_STORAGE_TEXEL_BUFFER, VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER,
VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC, VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC, or VK_↩
DESCRIPTOR_TYPE_INPUT_ATTACHMENT count against this limit.  Only descriptors in descriptor set
layouts created without the VK_DESCRIPTOR_SET_LAYOUT_CREATE_UPDATE_AFTER_BIND_POOL_BIT_EXT bit
set count against this limit.  For the fragment shader stage the framebuffer color attachments
also count against this limit."
```

6.16.2.59 maxPushConstantsSize

```
QString VkDescriptions::maxPushConstantsSize = "maxPushConstantsSize is the maximum size, in
bytes, of the pool of push constant memory.  For each of the push constant ranges indicated
by the pPushConstantRanges member of the VkPipelineLayoutCreateInfo structure, (offset + size)
must be less than or equal to this limit."
```

6.16.2.60 maxSampleMaskWords

```
QString VkDescriptions::maxSampleMaskWords = "maxSampleMaskWords is the maximum number of
array elements of a variable decorated with the SampleMask built - in decoration."
```

6.16.2.61 maxSamplerAllocationCount

```
QString VkDescriptions::maxSamplerAllocationCount = "maxSamplerAllocationCount is the maximum
number of sampler objects, as created by vkCreateSampler, which can simultaneously exist on a
device."
```

6.16.2.62 maxSamplerAnisotropy

```
QString VkDescriptions::maxSamplerAnisotropy = "maxSamplerAnisotropy is the maximum degree of
sampler anisotropy.  The maximum degree of anisotropic filtering used for an image sampling
operation is the minimum of the maxAnisotropy member of the VkSamplerCreateInfo structure and
this limit."
```

6.16.2.63 maxSamplerLodBias

```
QString VkDescriptions::maxSamplerLodBias = "maxSamplerLodBias is the maximum absolute sampler
LOD bias.  The sum of the mipLodBias member of the VkSamplerCreateInfo structure and the Bias
operand of image sampling operations in shader modules (or 0 if no Bias operand is provided
to an image sampling operation) are clamped to the range [-maxSamplerLodBias,+maxSamplerLod↩
Bias]."
```

6.16.2.64 maxStorageBufferRange

```
QString VkDescriptions::maxStorageBufferRange = "maxStorageBufferRange is the maximum value
that can be specified in the range member of any VkDescriptorBufferInfo structures passed to
a call to vkUpdateDescriptorSets for descriptors of type VK_DESCRIPTOR_TYPE_STORAGE_BUFFER or
VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC."
```

6.16.2.65 maxTessellationControlPerPatchOutputComponents

```
QString VkDescriptions::maxTessellationControlPerPatchOutputComponents = "maxTessellation↩
ControlPerPatchOutputComponents is the maximum number of components of per-patch output variables
which can be output from the tessellation control shader stage."
```

6.16.2.66 maxTessellationControlPerVertexInputComponents

```
QString VkDescriptions::maxTessellationControlPerVertexInputComponents = "maxTessellation↩
ControlPerVertexInputComponents is the maximum number of components of input variables which
can be provided as per-vertex inputs to the tessellation control shader stage."
```

6.16.2.67 maxTessellationControlPerVertexOutputComponents

```
QString VkDescriptions::maxTessellationControlPerVertexOutputComponents = "maxTessellation↩
ControlPerVertexOutputComponents is the maximum number of components of per-vertex output
variables which can be output from the tessellation control shader stage."
```

6.16.2.68 maxTessellationControlTotalOutputComponents

```
QString VkDescriptions::maxTessellationControlTotalOutputComponents = "maxTessellationControl↩
TotalOutputComponents is the maximum total number of components of per-vertex and per-patch
output variables which can be output from the tessellation control shader stage."
```

6.16.2.69 maxTessellationEvaluationInputComponents

```
QString VkDescriptions::maxTessellationEvaluationInputComponents = "maxTessellationEvaluation↩
InputComponents is the maximum number of components of input variables which can be provided
as per-vertex inputs to the tessellation evaluation shader stage."
```

6.16.2.70 maxTessellationEvaluationOutputComponents

```
QString VkDescriptions::maxTessellationEvaluationOutputComponents = "maxTessellationEvaluation↩
OutputComponents is the maximum number of components of per-vertex output variables which can
be output from the tessellation evaluation shader stage."
```

6.16.2.71 maxTessellationGenerationLevel

```
QString VkDescriptions::maxTessellationGenerationLevel = "maxTessellationGenerationLevel is
the maximum tessellation generation level supported by the fixed-function tessellation primitive
generator."
```

6.16.2.72 maxTessellationPatchSize

```
QString VkDescriptions::maxTessellationPatchSize = "maxTessellationPatchSize is the maximum
patch size, in vertices, of patches that can be processed by the tessellation control shader
and tessellation primitive generator.  The patchControlPoints member of the VkPipelineTessellation↩
StateCreateInfo structure specified at pipeline creation time and the value provided in the
OutputVertices execution mode of shader modules must be less than or equal to this limit."
```

6.16.2.73 maxTexelBufferElements

```
QString VkDescriptions::maxTexelBufferElements = "maxTexelBufferElements is the maximum number
of addressable texels for a buffer view created on a buffer which was created with the VK_↩
BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT or VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT set in the
usage member of the VkBufferCreateInfo structure."
```

6.16.2.74 maxTexelGatherOffset

```
QString VkDescriptions::maxTexelGatherOffset = "maxTexelGatherOffset is the maximum offset
value for the Offset or ConstOffsets image operands of any of the OpImage*Gather image instructions."
```

6.16.2.75 maxTexelOffset

```
QString VkDescriptions::maxTexelOffset = "maxTexelOffset is the maximum offset value for the
ConstOffset image operand of any of the OpImageSample* or OpImageFetch* image instructions."
```

6.16.2.76 maxUniformBufferRange

```
QString VkDescriptions::maxUniformBufferRange = "maxUniformBufferRange is the maximum value
that can be specified in the range member of any VkDescriptorBufferInfo structures passed to
a call to vkUpdateDescriptorSets for descriptors of type VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER or
VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC."
```

6.16.2.77 maxVertexInputAttributeOffset

```
QString VkDescriptions::maxVertexInputAttributeOffset = "maxVertexInputAttributeOffset is the
maximum vertex input attribute offset that can be added to the vertex input binding stride.
The offset member of the VkVertexInputAttributeDescription structure must be less than or
equal to this limit."
```

6.16.2.78 maxVertexInputAttributes

```
QString VkDescriptions::maxVertexInputAttributes = "maxVertexInputAttributes is the maximum
number of vertex input attributes that can be specified for a graphics pipeline.  These are
described in the array of VkVertexInputAttributeDescription structures that are provided at
graphics pipeline creation time via the pVertexAttributeDescriptions member of the VkPipeline↩
VertexInputStateCreateInfo structure."
```

6.16.2.79 maxVertexInputBindings

```
QString VkDescriptions::maxVertexInputBindings = "maxVertexInputBindings is the maximum number
of vertex buffers that can be specified for providing vertex attributes to a graphics pipeline.
These are described in the array of VkVertexInputBindingDescription structures that are provided
at graphics pipeline creation time via the pVertexBindingDescriptions member of the VkPipeline↩
VertexInputStateCreateInfo structure.  The binding member of VkVertexInputBindingDescription
must be less than this limit."
```

6.16.2.80 maxVertexInputBindingStride

```
QString VkDescriptions::maxVertexInputBindingStride = "maxVertexInputBindingStride is the
maximum vertex input binding stride that can be specified in a vertex input binding.  The
stride member of the VkVertexInputBindingDescription structure must be less than or equal to
this limit."
```

6.16.2.81 maxVertexOutputComponents

```
QString VkDescriptions::maxVertexOutputComponents = "maxVertexOutputComponents is the maximum
number of components of output variables which can be output by a vertex shader."
```

6.16.2.82 maxViewportDimensionsX

```
QString VkDescriptions::maxViewportDimensionsX = "maxViewportDimensions are the maximum viewport
dimensions in the X (width) and Y (height) dimensions, respectively.  The maximum viewport
dimensions must be greater than or equal to the largest image which can be created and used as
a framebuffer attachment.  See Controlling the Viewport."
```

6.16.2.83 maxViewportDimensionsY

```
QString VkDescriptions::maxViewportDimensionsY = "maxViewportDimensions are the maximum viewport
dimensions in the X (width) and Y (height) dimensions, respectively.  The maximum viewport
dimensions must be greater than or equal to the largest image which can be created and used as
a framebuffer attachment.  See Controlling the Viewport."
```

6.16.2.84 maxViewports

```
QString VkDescriptions::maxViewports = "maxViewports is the maximum number of active viewports.
The viewportCount member of the VkPipelineViewportStateCreateInfo structure that is provided
at pipeline creation must be less than or equal to this limit."
```

6.16.2.85 minInterpolationOffset

```
QString VkDescriptions::minInterpolationOffset = "minInterpolationOffset is the minimum negative
offset value for the offset operand of the InterpolateAtOffset extended instruction."
```

6.16.2.86 minMemoryMapAlignment

```
QString VkDescriptions::minMemoryMapAlignment = "minMemoryMapAlignment is the minimum required
alignment, in bytes, of host visible memory allocations within the host address space.  When
mapping a memory allocation with vkMapMemory, subtracting offset bytes from the returned pointer
will always produce an integer multiple of this limit."
```

6.16.2.87 minStorageBufferOffsetAlignment

```
QString VkDescriptions::minStorageBufferOffsetAlignment = "minStorageBufferOffsetAlignment is
the minimum required alignment, in bytes, for the offset member of the VkDescriptorBuffer↩
Info structure for storage buffers.  When a desc =riptor of type VK_DESCRIPTOR_TYPE_STORAGE↩
_BUFFER or VK_DESCRIPTOR_TYPE_STORAGE_BUFFER_DYNAMIC is updated, the offset must be an integer
multiple of this limit.  Similarly, dynamic offsets for storage buffers must be multiples of
this limit."
```

6.16.2.88 minTexelBufferOffsetAlignment

```
QString VkDescriptions::minTexelBufferOffsetAlignment = "minTexelBufferOffsetAlignment is the
minimum required alignment, in bytes, for the offset member of the VkBufferViewCreateInfo
structure for texel buffers.  If texelBufferAlignment is enabled, this limit is equivalent
to the maximum of the uniformTexelBufferOffsetAlignmentBytes and storageTexelBufferOffset↩
AlignmentBytes members of VkPhysicalDeviceTexelBufferAlignmentPropertiesEXT, but smaller alignment
is optionally:  allowed by storageTexelBufferOffsetSingleTexelAlignment and uniformTexel↩
BufferOffsetSingleTexelAlignment.  If texelBufferAlignment is not enabled, VkBufferView↩
CreateInfo::offset must be a multiple of this value."
```

6.16.2.89 minTexelGatherOffset

```
QString VkDescriptions::minTexelGatherOffset = "minTexelGatherOffset is the minimum offset
value for the Offset or ConstOffsets image operands of any of the OpImage*Gather image instructions."
```

6.16.2.90 minTexelOffset

```
QString VkDescriptions::minTexelOffset = "minTexelOffset is the minimum offset value for the
ConstOffset image operand of any of the OpImageSample* or OpImageFetch* image instructions."
```

6.16.2.91 minUniformBufferOffsetAlignment

```
QString VkDescriptions::minUniformBufferOffsetAlignment = "minUniformBufferOffsetAlignment is
the minimum required alignment, in bytes, for the offset member of the VkDescriptorBuffer↩
Info structure for uniform buffers.  When a desc =riptor of type VK_DESCRIPTOR_TYPE_UNIFORM↩
_BUFFER or VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC is updated, the offset must be an integer
multiple of this limit.  Similarly, dynamic offsets for uniform buffers must be multiples of
this limit."
```

6.16.2.92 mipmapPrecisionBits

```
QString VkDescriptions::mipmapPrecisionBits = "mipmapPrecisionBits is the number of bits of
division that the LOD calculation for mipmap fetching get snapped to when determining the
contribution from each mip level to the mip filtered results."
```

6.16.2.93 nonCoherentAtomSize

```
QString VkDescriptions::nonCoherentAtomSize = "nonCoherentAtomSize is the size and alignment
in bytes that bounds concurrent access to host - mapped device memory."
```

6.16.2.94 optimalBufferCopyOffsetAlignment

```
QString VkDescriptions::optimalBufferCopyOffsetAlignment = "optimalBufferCopyOffsetAlignment
is the optimal buffer offset alignment in bytes for vkCmdCopyBufferToImage and vkCmdCopy↩
ImageToBuffer.  The per texel alignment requirements are enforced, but applications should use
the optimal alignment for optimal performance and power use."
```

6.16.2.95 optimalBufferCopyRowPitchAlignment

```
QString VkDescriptions::optimalBufferCopyRowPitchAlignment = "optimalBufferCopyRowPitch↩
Alignment is the optimal buffer row pitch alignment in bytes for vkCmdCopyBufferToImage and
vkCmdCopyImageToBuffer.  Row pitch is the number of bytes between texels with the same X coordinate
in adjacent rows (Y coordinates differ by one).  The per texel alignment requirements are
enforced, but applications should use the optimal alignment for optimal performance and power
use."
```

6.16.2.96 pointSizeGranularity

```
QString VkDescriptions::pointSizeGranularity = "pointSizeGranularity is the granularity of
supported point sizes.  Not all point sizes in the range defined by pointSizeRange are supported.
This limit specifies the granularity (or increment) between successive supported point sizes."
```

6.16.2.97 pointSizeRangeX

```
QString VkDescriptions::pointSizeRangeX = "pointSizeRange is the range [minimum,maximum] of
supported sizes for points.  Values written to variables decorated with the PointSize built-in
decoration are clamped to this range."
```

6.16.2.98 pointSizeRangeY

```
QString VkDescriptions::pointSizeRangeY = "pointSizeRange is the range [minimum,maximum] of
supported sizes for points.  Values written to variables decorated with the PointSize built-in
decoration are clamped to this range."
```

6.16.2.99 sampledImageColorSampleCounts

```
QString VkDescriptions::sampledImageColorSampleCounts = "sampledImageColorSampleCounts is a
bitmask1 of VkSampleCountFlagBits indicating the sample counts supported for all 2D images
created with VK_IMAGE_TILING_OPTIMAL, usage containing VK_IMAGE_USAGE_SAMPLED_BIT, and a non
- integer color format."
```

6.16.2.100 sampledImageDepthSampleCounts

```
QString VkDescriptions::sampledImageDepthSampleCounts = "sampledImageDepthSampleCounts is a
bitmask1 of VkSampleCountFlagBits indicating the sample counts supported for all 2D images
created with VK_IMAGE_TILING_OPTIMAL, usage containing VK_IMAGE_USAGE_SAMPLED_BIT, and a depth
format."
```

6.16.2.101 sampledImageIntegerSampleCounts

```
QString VkDescriptions::sampledImageIntegerSampleCounts = "sampledImageIntegerSampleCounts is
a bitmask1 of VkSampleCountFlagBits indicating the sample counts supported for all 2D images
created with VK_IMAGE_TILING_OPTIMAL, usage containing VK_IMAGE_USAGE_SAMPLED_BIT, and an
integer color format."
```

6.16.2.102 sampledImageStencilSampleCounts

```
QString VkDescriptions::sampledImageStencilSampleCounts = "sampledImageStencilSampleCounts is
a bitmask1 of VkSampleCountFlagBits indicating the sample supported for all 2D images created
with VK_IMAGE_TILING_OPTIMAL, usage containing VK_IMAGE_USAGE_SAMPLED_BIT, and a stencil format."
```

6.16.2.103 sparseAddressSpaceSize

```
QString VkDescriptions::sparseAddressSpaceSize = "sparseAddressSpaceSize is the total amount
of address space available, in bytes, for sparse memory resources.  This is an upper bound
on the sum of the size of all sparse resources, regardless of whether any memory is bound to
them."
```

6.16.2.104 standardSampleLocations

```
QString VkDescriptions::standardSampleLocations = "standardSampleLocations specifies whether
rasterization uses the standard sample locations as documented in Multisampling.  If set to
VK_TRUE, the implementation uses the documented sample locations.  If set to VK_FALSE, the
implementation may use different sample locations."
```

6.16.2.105 storageImageSampleCounts

```
QString VkDescriptions::storageImageSampleCounts = "storageImageSampleCounts is a bitmask1 of
VkSampleCountFlagBits indicating the sample counts supported for all 2D images created with
VK_IMAGE_TILING_OPTIMAL, and usage containing VK_IMAGE_USAGE_STORAGE_BIT."
```

6.16.2.106 strictLines

```
QString VkDescriptions::strictLines = "strictLines specifies whether lines are rasterized
according to the preferred method of rasterization.  If set to VK_FALSE, lines may be rasterized
under a relaxed set of rules.  If set to VK_TRUE, lines are rasterized as per the strict definition.
See Basic Line Segment Rasterization."
```

6.16.2.107 subPixelInterpolationOffsetBits

```
QString VkDescriptions::subPixelInterpolationOffsetBits = "subPixelInterpolationOffsetBits
is the number of subpixel fractional bits that the x and y offsets to the InterpolateAtOffset
extended instruction may be rounded to as fixed-point values."
```

6.16.2.108 subPixelPrecisionBits

```
QString VkDescriptions::subPixelPrecisionBits = "subPixelPrecisionBits is the number of bits
of subpixel precision in framebuffer coordinates xf and yf."
```

6.16.2.109 subTexelPrecisionBits

```
QString VkDescriptions::subTexelPrecisionBits = "subTexelPrecisionBits is the number of bits
of precision in the division along an axis of an image used for minification and magnification
filters."
```

6.16.2.110 timestampComputeAndGraphics

```
QString VkDescriptions::timestampComputeAndGraphics = "timestampComputeAndGraphics specifies
support for timestamps on all graphics and compute queues.  If this limit is set to VK_TRUE,
all queues that advertise the VK_QUEUE_GRAPHICS_BIT or VK_QUEUE_COMPUTE_BIT in the VkQueue↩
FamilyProperties::queueFlags support VkQueueFamilyProperties::timestampValidBits of at least
36.  See Timestamp Queries."
```

6.16.2.111 timestampPeriod

```
QString VkDescriptions::timestampPeriod = "timestampPeriod is the number of nanoseconds required
for a timestamp query to be incremented by 1.  See Timestamp Queries."
```

6.16.2.112 viewportBoundsRangeX

```
QString VkDescriptions::viewportBoundsRangeX = "viewportBoundsRange is the [minimum, maximum]
range that the corners of a viewport must be contained in.  This range must be at least [-2
× size, 2 × size - 1], where size = max(maxViewportDimensions[0], maxViewportDimensions[1]).
See Controlling the Viewport."
```

6.16.2.113 viewportBoundsRangeY

```
QString VkDescriptions::viewportBoundsRangeY = "viewportBoundsRange is the [minimum, maximum]
range that the corners of a viewport must be contained in.  This range must be at least [-2
× size, 2 × size - 1], where size = max(maxViewportDimensions[0], maxViewportDimensions[1]).
See Controlling the Viewport."
```

6.16.2.114 viewportSubPixelBits

```
QString VkDescriptions::viewportSubPixelBits = "viewportSubPixelBits is the number of bits of
subpixel precision for viewport bounds.  The subpixel precision that floating - point viewport
bounds are interpreted at is given by this limit."
```

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

- src/vkdescriptions.hpp

6.17 gpu::VkInfo Struct Reference

VkInfo struct used to store the GPU and OpenGL data. Structure containing the Vulkan information.

```
#include <VkInfo.hpp>
```

Public Attributes

- uint32_t maxImageDimension1D = -1
- uint32_t maxImageDimension2D = -1
- uint32_t maxImageDimension3D = -1
- uint32_t maxImageDimensionCube = -1
- uint32_t maxImageArrayLayers = -1
- uint32_t maxTexelBufferElements = -1
- uint32_t maxUniformBufferRange = -1
- uint32_t maxStorageBufferRange = -1
- uint32_t maxPushConstantsSize = -1
- uint32_t maxMemoryAllocationCount = -1
- uint32_t maxSamplerAllocationCount = -1
- uint64_t bufferImageGranularity = -1
- uint64_t sparseAddressSpaceSize = -1

- uint32_t maxBoundDescriptorSets = -1
- uint32_t maxPerStageDescriptorSamplers = -1
- uint32_t maxPerStageDescriptorUniformBuffers = -1
- uint32_t maxPerStageDescriptorStorageBuffers = -1
- uint32_t maxPerStageDescriptorSampledImages = -1
- uint32_t maxPerStageDescriptorStorageImages = -1
- uint32_t maxPerStageDescriptorInputAttachments = -1
- uint32_t maxPerStageResources = -1
- uint32_t maxDescriptorSetSamplers = -1
- uint32_t maxDescriptorSetUniformBuffers = -1
- uint32_t maxDescriptorSetUniformBuffersDynamic = -1
- uint32_t maxDescriptorSetStorageBuffers = -1
- uint32_t maxDescriptorSetStorageBuffersDynamic = -1
- uint32_t maxDescriptorSetSampledImages = -1
- uint32_t maxDescriptorSetStorageImages = -1
- uint32_t maxDescriptorSetInputAttachments = -1
- uint32_t maxVertexInputAttributes = -1
- uint32_t maxVertexInputBindings = -1
- uint32_t maxVertexInputAttributeOffset = -1
- uint32_t maxVertexInputBindingStride = -1
- uint32_t maxVertexOutputComponents = -1
- uint32_t maxTessellationGenerationLevel = -1
- uint32_t maxTessellationPatchSize = -1
- uint32_t maxTessellationControlPerVertexInputComponents = -1
- uint32_t maxTessellationControlPerVertexOutputComponents = -1
- uint32_t maxTessellationControlPerPatchOutputComponents = -1
- uint32_t maxTessellationControlTotalOutputComponents = -1
- uint32_t maxTessellationEvaluationInputComponents = -1
- uint32_t maxTessellationEvaluationOutputComponents = -1
- uint32_t maxGeometryShaderInvocations = -1
- uint32_t maxGeometryInputComponents = -1
- uint32_t maxGeometryOutputComponents = -1
- uint32_t maxGeometryOutputVertices = -1
- uint32_t maxGeometryTotalOutputComponents = -1
- uint32_t maxFragmentInputComponents = -1
- uint32_t maxFragmentOutputAttachments = -1
- uint32_t maxFragmentDualSrcAttachments = -1
- uint32_t maxFragmentCombinedOutputResources = -1
- uint32_t maxComputeSharedMemorySize = -1
- uint32_t maxComputeWorkGroupCountX = -1
- uint32_t maxComputeWorkGroupCountY = -1
- uint32_t maxComputeWorkGroupCountZ = -1
- uint32_t maxComputeWorkGroupInvocations = -1
- uint32_t maxComputeWorkGroupSizeX = -1
- uint32_t maxComputeWorkGroupSizeY = -1
- uint32_t maxComputeWorkGroupSizeZ = -1
- uint32_t subPixelPrecisionBits = -1
- uint32_t subTexelPrecisionBits = -1
- uint32_t mipmapPrecisionBits = -1
- uint32_t maxDrawIndexedIndexValue = -1
- uint32_t maxDrawIndirectCount = -1
- float maxSamplerLodBias = -1
- float maxSamplerAnisotropy = -1
- uint32_t maxViewports = -1
- uint32_t maxViewportDimensionsX = -1

- uint32_t maxViewportDimensionsY = -1
- float viewportBoundsRangeX = -1
- float viewportBoundsRangeY = -1
- uint32_t viewportSubPixelBits = -1
- size_t minMemoryMapAlignment = -1
- uint64_t minTexelBufferOffsetAlignment = -1
- uint64_t minUniformBufferOffsetAlignment = -1
- uint64_t minStorageBufferOffsetAlignment = -1
- int32_t minTexelOffset = -1
- uint32_t maxTexelOffset = -1
- int32_t minTexelGatherOffset = -1
- uint32_t maxTexelGatherOffset = -1
- float minInterpolationOffset = -1
- float maxInterpolationOffset = -1
- uint32_t subPixelInterpolationOffsetBits = -1
- uint32_t maxFramebufferWidth = -1
- uint32_t maxFramebufferHeight = -1
- uint32_t maxFramebufferLayers = -1
- Count framebufferColorSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
- Count framebufferDepthSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
- Count framebufferStencilSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
- Count framebufferNoAttachmentsSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
- uint32_t maxColorAttachments = -1
- Count sampledImageColorSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
- Count sampledImageIntegerSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
- Count sampledImageDepthSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
- Count sampledImageStencilSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
- Count storageImageSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
- uint32_t maxSampleMaskWords = -1
- bool timestampComputeAndGraphics = -1
- float timestampPeriod = -1
- uint32_t maxClipDistances = -1
- uint32_t maxCullDistances = -1
- uint32_t maxCombinedClipAndCullDistances = -1
- uint32_t discreteQueuePriorities = -1
- float pointSizeRangeX = -1
- float pointSizeRangeY = -1
- float lineWidthRangeX = -1
- float lineWidthRangeY = -1
- float pointSizeGranularity = -1
- float lineWidthGranularity = -1
- bool strictLines = -1
- bool standardSampleLocations = -1
- uint64_t optimalBufferCopyOffsetAlignment = -1
- uint64_t optimalBufferCopyRowPitchAlignment = -1
- uint64_t nonCoherentAtomSize = -1

6.17.1 Detailed Description

VkInfo struct used to store the GPU and OpenGL data. Structure containing the Vulkan information.

6.17.2 Member Data Documentation

6.17.2.1 bufferImageGranularity

```
uint64_t gpu::VkInfo::bufferImageGranularity = -1
```

6.17.2.2 discreteQueuePriorities

```
uint32_t gpu::VkInfo::discreteQueuePriorities = -1
```

6.17.2.3 framebufferColorSampleCounts

```
Count gpu::VkInfo::framebufferColorSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
```

6.17.2.4 framebufferDepthSampleCounts

```
Count gpu::VkInfo::framebufferDepthSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
```

6.17.2.5 framebufferNoAttachmentsSampleCounts

```
Count gpu::VkInfo::framebufferNoAttachmentsSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
```

6.17.2.6 framebufferStencilSampleCounts

```
Count gpu::VkInfo::framebufferStencilSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
```

6.17.2.7 lineWidthGranularity

```
float gpu::VkInfo::lineWidthGranularity = -1
```

6.17.2.8 lineWidthRangeX

```
float gpu::VkInfo::lineWidthRangeX = -1
```

6.17.2.9 lineWidthRangeY

```
float gpu::VkInfo::lineWidthRangeY = -1
```

6.17.2.10 maxBoundDescriptorSets

```
uint32_t gpu::VkInfo::maxBoundDescriptorSets = -1
```

6.17.2.11 maxClipDistances

```
uint32_t gpu::VkInfo::maxClipDistances = -1
```

6.17.2.12 maxColorAttachments

```
uint32_t gpu::VkInfo::maxColorAttachments = -1
```

6.17.2.13 maxCombinedClipAndCullDistances

```
uint32_t gpu::VkInfo::maxCombinedClipAndCullDistances = -1
```

6.17.2.14 maxComputeSharedMemorySize

```
uint32_t gpu::VkInfo::maxComputeSharedMemorySize = -1
```

6.17.2.15 maxComputeWorkGroupCountX

```
uint32_t gpu::VkInfo::maxComputeWorkGroupCountX = -1
```

6.17.2.16 maxComputeWorkGroupCountY

```
uint32_t gpu::VkInfo::maxComputeWorkGroupCountY = -1
```

6.17.2.17 maxComputeWorkGroupCountZ

```
uint32_t gpu::VkInfo::maxComputeWorkGroupCountZ = -1
```

6.17.2.18 maxComputeWorkGroupInvocations

```
uint32_t gpu::VkInfo::maxComputeWorkGroupInvocations = -1
```

6.17.2.19 maxComputeWorkGroupSizeX

```
uint32_t gpu::VkInfo::maxComputeWorkGroupSizeX = -1
```

6.17.2.20 maxComputeWorkGroupSizeY

```
uint32_t gpu::VkInfo::maxComputeWorkGroupSizeY = -1
```

6.17.2.21 maxComputeWorkGroupSizeZ

```
uint32_t gpu::VkInfo::maxComputeWorkGroupSizeZ = -1
```

6.17.2.22 maxCullDistances

```
uint32_t gpu::VkInfo::maxCullDistances = -1
```

6.17.2.23 maxDescriptorSetInputAttachments

```
uint32_t gpu::VkInfo::maxDescriptorSetInputAttachments = -1
```

6.17.2.24 maxDescriptorSetSampledImages

```
uint32_t gpu::VkInfo::maxDescriptorSetSampledImages = -1
```

6.17.2.25 maxDescriptorSetSamplers

```
uint32_t gpu::VkInfo::maxDescriptorSetSamplers = -1
```

6.17.2.26 maxDescriptorSetStorageBuffers

```
uint32_t gpu::VkInfo::maxDescriptorSetStorageBuffers = -1
```

6.17.2.27 maxDescriptorSetStorageBuffersDynamic

```
uint32_t gpu::VkInfo::maxDescriptorSetStorageBuffersDynamic = -1
```

6.17.2.28 maxDescriptorSetStorageImages

```
uint32_t gpu::VkInfo::maxDescriptorSetStorageImages = -1
```

6.17.2.29 maxDescriptorSetUniformBuffers

```
uint32_t gpu::VkInfo::maxDescriptorSetUniformBuffers = -1
```

6.17.2.30 maxDescriptorSetUniformBuffersDynamic

```
uint32_t gpu::VkInfo::maxDescriptorSetUniformBuffersDynamic = -1
```

6.17.2.31 maxDrawIndexedIndexValue

```
uint32_t gpu::VkInfo::maxDrawIndexedIndexValue = -1
```

6.17.2.32 maxDrawIndirectCount

```
uint32_t gpu::VkInfo::maxDrawIndirectCount = -1
```

6.17.2.33 maxFragmentCombinedOutputResources

```
uint32_t gpu::VkInfo::maxFragmentCombinedOutputResources = -1
```

6.17.2.34 maxFragmentDualSrcAttachments

```
uint32_t gpu::VkInfo::maxFragmentDualSrcAttachments = -1
```

6.17.2.35 maxFragmentInputComponents

```
uint32_t gpu::VkInfo::maxFragmentInputComponents = -1
```

6.17.2.36 maxFragmentOutputAttachments

```
uint32_t gpu::VkInfo::maxFragmentOutputAttachments = -1
```

6.17.2.37 maxFramebufferHeight

```
uint32_t gpu::VkInfo::maxFramebufferHeight = -1
```

6.17.2.38 maxFramebufferLayers

```
uint32_t gpu::VkInfo::maxFramebufferLayers = -1
```

6.17.2.39 maxFramebufferWidth

```
uint32_t gpu::VkInfo::maxFramebufferWidth = -1
```

6.17.2.40 maxGeometryInputComponents

```
uint32_t gpu::VkInfo::maxGeometryInputComponents = -1
```

6.17.2.41 maxGeometryOutputComponents

```
uint32_t gpu::VkInfo::maxGeometryOutputComponents = -1
```

6.17.2.42 maxGeometryOutputVertices

```
uint32_t gpu::VkInfo::maxGeometryOutputVertices = -1
```

6.17.2.43 maxGeometryShaderInvocations

```
uint32_t gpu::VkInfo::maxGeometryShaderInvocations = -1
```

6.17.2.44 maxGeometryTotalOutputComponents

```
uint32_t gpu::VkInfo::maxGeometryTotalOutputComponents = -1
```

6.17.2.45 maxImageArrayLayers

```
uint32_t gpu::VkInfo::maxImageArrayLayers = -1
```

6.17.2.46 maxImageDimension1D

```
uint32_t gpu::VkInfo::maxImageDimension1D = -1
```

6.17.2.47 maxImageDimension2D

```
uint32_t gpu::VkInfo::maxImageDimension2D = -1
```

6.17.2.48 maxImageDimension3D

```
uint32_t gpu::VkInfo::maxImageDimension3D = -1
```

6.17.2.49 maxImageDimensionCube

```
uint32_t gpu::VkInfo::maxImageDimensionCube = -1
```

6.17.2.50 maxInterpolationOffset

```
float gpu::VkInfo::maxInterpolationOffset = -1
```

6.17.2.51 maxMemoryAllocationCount

```
uint32_t gpu::VkInfo::maxMemoryAllocationCount = -1
```

6.17.2.52 maxPerStageDescriptorInputAttachments

```
uint32_t gpu::VkInfo::maxPerStageDescriptorInputAttachments = -1
```

6.17.2.53 maxPerStageDescriptorSampledImages

```
uint32_t gpu::VkInfo::maxPerStageDescriptorSampledImages = -1
```

6.17.2.54 maxPerStageDescriptorSamplers

```
uint32_t gpu::VkInfo::maxPerStageDescriptorSamplers = -1
```

6.17.2.55 maxPerStageDescriptorStorageBuffers

```
uint32_t gpu::VkInfo::maxPerStageDescriptorStorageBuffers = -1
```

6.17.2.56 maxPerStageDescriptorStorageImages

```
uint32_t gpu::VkInfo::maxPerStageDescriptorStorageImages = -1
```

6.17.2.57 maxPerStageDescriptorUniformBuffers

```
uint32_t gpu::VkInfo::maxPerStageDescriptorUniformBuffers = -1
```

6.17.2.58 maxPerStageResources

```
uint32_t gpu::VkInfo::maxPerStageResources = -1
```

6.17.2.59 maxPushConstantsSize

```
uint32_t gpu::VkInfo::maxPushConstantsSize = -1
```

6.17.2.60 maxSampleMaskWords

```
uint32_t gpu::VkInfo::maxSampleMaskWords = -1
```

6.17.2.61 maxSamplerAllocationCount

```
uint32_t gpu::VkInfo::maxSamplerAllocationCount = -1
```

6.17.2.62 maxSamplerAnisotropy

```
float gpu::VkInfo::maxSamplerAnisotropy = -1
```

6.17.2.63 maxSamplerLodBias

```
float gpu::VkInfo::maxSamplerLodBias = -1
```

6.17.2.64 maxStorageBufferRange

```
uint32_t gpu::VkInfo::maxStorageBufferRange = -1
```

6.17.2.65 maxTessellationControlPerPatchOutputComponents

```
uint32_t gpu::VkInfo::maxTessellationControlPerPatchOutputComponents = -1
```

6.17.2.66 maxTessellationControlPerVertexInputComponents

```
uint32_t gpu::VkInfo::maxTessellationControlPerVertexInputComponents = -1
```

6.17.2.67 maxTessellationControlPerVertexOutputComponents

```
uint32_t gpu::VkInfo::maxTessellationControlPerVertexOutputComponents = -1
```

6.17.2.68 maxTessellationControlTotalOutputComponents

```
uint32_t gpu::VkInfo::maxTessellationControlTotalOutputComponents = -1
```

6.17.2.69 maxTessellationEvaluationInputComponents

```
uint32_t gpu::VkInfo::maxTessellationEvaluationInputComponents = -1
```

6.17.2.70 maxTessellationEvaluationOutputComponents

```
uint32_t gpu::VkInfo::maxTessellationEvaluationOutputComponents = -1
```

6.17.2.71 maxTessellationGenerationLevel

```
uint32_t gpu::VkInfo::maxTessellationGenerationLevel = -1
```

6.17.2.72 maxTessellationPatchSize

```
uint32_t gpu::VkInfo::maxTessellationPatchSize = -1
```

6.17.2.73 maxTexelBufferElements

```
uint32_t gpu::VkInfo::maxTexelBufferElements = -1
```

6.17.2.74 maxTexelGatherOffset

```
uint32_t gpu::VkInfo::maxTexelGatherOffset = -1
```

6.17.2.75 maxTexelOffset

```
uint32_t gpu::VkInfo::maxTexelOffset = -1
```

6.17.2.76 maxUniformBufferRange

```
uint32_t gpu::VkInfo::maxUniformBufferRange = -1
```

6.17.2.77 maxVertexInputAttributeOffset

```
uint32_t gpu::VkInfo::maxVertexInputAttributeOffset = -1
```

6.17.2.78 maxVertexInputAttributes

```
uint32_t gpu::VkInfo::maxVertexInputAttributes = -1
```

6.17.2.79 maxVertexInputBindings

```
uint32_t gpu::VkInfo::maxVertexInputBindings = -1
```

6.17.2.80 maxVertexInputBindingStride

```
uint32_t gpu::VkInfo::maxVertexInputBindingStride = -1
```

6.17.2.81 maxVertexOutputComponents

```
uint32_t gpu::VkInfo::maxVertexOutputComponents = -1
```

6.17.2.82 maxViewportDimensionsX

```
uint32_t gpu::VkInfo::maxViewportDimensionsX = -1
```

6.17.2.83 maxViewportDimensionsY

```
uint32_t gpu::VkInfo::maxViewportDimensionsY = -1
```

6.17.2.84 maxViewports

```
uint32_t gpu::VkInfo::maxViewports = -1
```

6.17.2.85 minInterpolationOffset

```
float gpu::VkInfo::minInterpolationOffset = -1
```

6.17.2.86 minMemoryMapAlignment

```
size_t gpu::VkInfo::minMemoryMapAlignment = -1
```

6.17.2.87 minStorageBufferOffsetAlignment

```
uint64_t gpu::VkInfo::minStorageBufferOffsetAlignment = -1
```

6.17.2.88 minTexelBufferOffsetAlignment

```
uint64_t gpu::VkInfo::minTexelBufferOffsetAlignment = -1
```

6.17.2.89 minTexelGatherOffset

```
int32_t gpu::VkInfo::minTexelGatherOffset = -1
```

6.17.2.90 minTexelOffset

```
int32_t gpu::VkInfo::minTexelOffset = -1
```

6.17.2.91 minUniformBufferOffsetAlignment

```
uint64_t gpu::VkInfo::minUniformBufferOffsetAlignment = -1
```

6.17.2.92 mipmapPrecisionBits

```
uint32_t gpu::VkInfo::mipmapPrecisionBits = -1
```

6.17.2.93 nonCoherentAtomSize

```
uint64_t gpu::VkInfo::nonCoherentAtomSize = -1
```

6.17.2.94 optimalBufferCopyOffsetAlignment

```
uint64_t gpu::VkInfo::optimalBufferCopyOffsetAlignment = -1
```

6.17.2.95 optimalBufferCopyRowPitchAlignment

```
uint64_t gpu::VkInfo::optimalBufferCopyRowPitchAlignment = -1
```

6.17.2.96 pointSizeGranularity

```
float gpu::VkInfo::pointSizeGranularity = -1
```

6.17.2.97 pointSizeRangeX

```
float gpu::VkInfo::pointSizeRangeX = -1
```

6.17.2.98 pointSizeRangeY

```
float gpu::VkInfo::pointSizeRangeY = -1
```

6.17.2.99 sampledImageColorSampleCounts

```
Count gpu::VkInfo::sampledImageColorSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
```

6.17.2.100 sampledImageDepthSampleCounts

```
Count gpu::VkInfo::sampledImageDepthSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
```

6.17.2.101 sampledImageIntegerSampleCounts

```
Count gpu::VkInfo::sampledImageIntegerSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
```

6.17.2.102 sampledImageStencilSampleCounts

```
Count gpu::VkInfo::sampledImageStencilSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
```

6.17.2.103 sparseAddressSpaceSize

```
uint64_t gpu::VkInfo::sparseAddressSpaceSize = -1
```

6.17.2.104 standardSampleLocations

```
bool gpu::VkInfo::standardSampleLocations = -1
```

6.17.2.105 storageImageSampleCounts

```
Count gpu::VkInfo::storageImageSampleCounts = VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
```

6.17.2.106 strictLines

```
bool gpu::VkInfo::strictLines = -1
```

6.17.2.107 subPixelInterpolationOffsetBits

```
uint32_t gpu::VkInfo::subPixelInterpolationOffsetBits = -1
```

6.17.2.108 subPixelPrecisionBits

```
uint32_t gpu::VkInfo::subPixelPrecisionBits = -1
```

6.17.2.109 subTexelPrecisionBits

```
uint32_t gpu::VkInfo::subTexelPrecisionBits = -1
```

6.17.2.110 timestampComputeAndGraphics

```
bool gpu::VkInfo::timestampComputeAndGraphics = -1
```

6.17.2.111 timestampPeriod

```
float gpu::VkInfo::timestampPeriod = -1
```

6.17.2.112 viewportBoundsRangeX

```
float gpu::VkInfo::viewportBoundsRangeX = -1
```

6.17.2.113 viewportBoundsRangeY

```
float gpu::VkInfo::viewportBoundsRangeY = -1
```

6.17.2.114 viewportSubPixelBits

```
uint32_t gpu::VkInfo::viewportSubPixelBits = -1
```

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

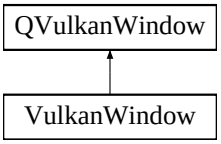
- libs/opengl-engine/libs/profiler/src/VkInfo.hpp

6.18 VulkanWindow Class Reference

Class for vulkan window. Vulkan Window class based on QVulkanWindow widget.

`#include <vulkanwindow.h>`

Inheritance diagram for VulkanWindow:



Public Member Functions

- VulkanWindow (gpu::Profiler *profiler=nullptr, int framerate=60)

  Constructs the object. The profiler is passed a a pointer so that the data can be retrieved by the GUI tool.
- QVulkanWindowRenderer * createRenderer () override

  Creates the renderer. QVulkanWindowRenderer contains all the buffers, data, resources, ... used to render the scene.

6.18.1 Detailed Description

Class for vulkan window. Vulkan Window class based on QVulkanWindow widget.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 VulkanWindow()

```
VulkanWindow::VulkanWindow (
            gpu::Profiler * profiler = nullptr,
            int framerate = 60 )
```

Constructs the object. The profiler is passed a a pointer so that the data can be retrieved by the GUI tool.

Parameters

profiler	The profiler used to save the GPU information
framerate	The target framerate

6.18.3 Member Function Documentation

6.18.3.1 createRenderer()

```
QVulkanWindowRenderer * VulkanWindow::createRenderer ( )  [override]
```

Creates the renderer. QVulkanWindowRenderer contains all the buffers, data, resources, ... used to render the scene.

Returns

Vulkan renderer.

The documentation for this class was generated from the following files:

- libs/vulkan-engine/src/vulkanwindow.h
- libs/vulkan-engine/src/vulkanwindow.cpp

Chapter 7

File Documentation

7.1 libs/opengl-engine/libs/math-library/src/MathLibrary.hpp File Reference

```
#include "Vector2.hpp"
#include "Vector3.hpp"
```

7.2 libs/opengl-engine/libs/math-library/src/Vector2.hpp File Reference

```
#include <cmath>
```

Classes

- class math::Vector2

  2D Vector class containing 2D vector operations. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()

Namespaces

- math

7.3 libs/opengl-engine/libs/math-library/src/Vector3.hpp File Reference

```
#include <cmath>
```

Classes

- class math::Vector3

  Vector3 structure. Operations available: Vec+Vec, Vec+=Vec, Vec-Vec, Vec-=Vec, Vec=Vec, Vec==Vec, Vec∗d, Vec∗=d, Vec/d, Vec/=d, Vec.doc(Vec), Vec.cross(Vec), Vec.normal(), Vec.normalise(), Vec.magnitude(), Vec.magnitudeSquare()

Namespaces

- math

7.4 libs/opengl-engine/libs/obj-loader/src/Face.hpp File Reference

Classes

- class obj::Face

  Class used to represent a face. A face is represented by 3 points. Each point is composed of a position, a texture coordinate and a normal. The data is not saved here but only the index.

Namespaces

- obj

7.5 libs/opengl-engine/libs/obj-loader/src/OBJ.cpp File Reference

```
#include "OBJ.h"
```

7.6 libs/opengl-engine/libs/obj-loader/src/OBJ.h File Reference

```
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <algorithm>
#include "MathLibrary.hpp"
#include "Face.hpp"
```

Classes

- class obj::OBJ

  Class for OBJ. Loads and OBJ loader form file, saves a model to path, adds a vertex, updates the faces and gets the data. The data is saved as vectors in an std::vector container id is accessed by index.

Namespaces

- obj

7.7 libs/opengl-engine/libs/profiler/src/GLInfo.hpp File Reference

Classes

- struct gpu::GLInfo

  GLInfo struct used to store the GPU and OpenGL data. Structure containing the OpenGL information.

Namespaces

- gpu

7.8 libs/opengl-engine/libs/profiler/src/Profiler.cpp File Reference

```
#include "Profiler.h"
```

7.9 libs/opengl-engine/libs/profiler/src/Profiler.h File Reference

```
#include <vector>
#include <string>
#include <fstream>
#include "GLInfo.hpp"
#include "VkInfo.hpp"
```

Classes

- class gpu::Profiler

  Class for profiler. Saves the GPU information: vendor, renderer. OpenGL version, shading version, extensions. Vulkan device name, API version, driver version, vendor ID, device ID, extensions. Std::vector stores Profiling data (polycount per time).

Namespaces

- gpu

7.10 libs/opengl-engine/libs/profiler/src/VkInfo.hpp File Reference

Classes

- struct gpu::VkInfo

  VkInfo struct used to store the GPU and OpenGL data. Structure containing the Vulkan information.

Namespaces

- gpu

Enumerations

- enum gpu::Count {
  gpu::VK_SAMPLE_COUNT_1_BIT = 0x00000001, gpu::VK_SAMPLE_COUNT_2_BIT = 0x00000002,
  gpu::VK_SAMPLE_COUNT_4_BIT = 0x00000004, gpu::VK_SAMPLE_COUNT_8_BIT = 0x00000008,
  gpu::VK_SAMPLE_COUNT_16_BIT = 0x00000010, gpu::VK_SAMPLE_COUNT_32_BIT = 0x00000020,
  gpu::VK_SAMPLE_COUNT_64_BIT = 0x00000040, gpu::VK_SAMPLE_COUNT_FLAG_BITS_MAX_ENUM
  = 0x7FFFFFFF }

7.11 libs/opengl-engine/libs/triangulate/src/Triangulate.hpp File Reference

```
#include "lib/obj-loader/src/OBJ.h"
#include <vector>
```

Namespaces

- obj

Functions

- void obj::triangulate (OBJ &model)

  Triangulate and subdivide an OBJ file. Uses a 3D model loaded and loops through the faces ABC. For each face, computes 3 new points (D, E and F). D = (A + B)/2, E = (B + C)/2 and F = (C + A)/2. Adds D, E and F to the model and retrieved their indices. Delete the current face and create 4 faces: ADF, DBE, DEF and ECF.

7.12 libs/opengl-engine/src/getgpuinfo.cpp File Reference

```
#include "glwidget.h"
```

7.13 libs/opengl-engine/src/glwidget.cpp File Reference

```
#include "glwidget.h"
#include <QFile>
#include <QDir>
#include <omp.h>
#include "libs/Triangulate.hpp"
```

Macros

- #define QT_APPLICATION

7.13.1 Macro Definition Documentation

7.13.1.1 QT_APPLICATION

```
#define QT_APPLICATION
```

7.14 libs/opengl-engine/src/glwidget.h File Reference

```
#include <QtGlobal>
#include <QOpenGLWidget>
#include <QOpenGLFunctions>
#include <QOpenGLBuffer>
#include <QOpenGLVertexArrayObject>
#include <QOpenGLShaderProgram>
#include <QOpenGLTexture>
#include <QKeyEvent>
#include <QDebug>
#include <QElapsedTimer>
#include <QTimer>
#include <map>
#include "model.hpp"
#include "libs/Profiler.h"
```

Classes

- class GLWidget

  Class for gl widget. OpenGL Widget class based on QGLWidget and QOpenGLFunctions to implement the OpenGL API.

7.15 libs/opengl-engine/src/glwindow.cpp File Reference

```
#include "glwindow.h"
#include "ui_glwindow.h"
```

7.16 libs/opengl-engine/src/glwindow.h File Reference

```
#include <QMainWindow>
#include "glwidget.h"
```

Classes

- class GLWindow

  Class for gl window. OpenGL window based on QMainWindow widget.

Namespaces

- Ui

7.17 libs/opengl-engine/src/model.hpp File Reference

```
#include "libs/OBJ.h"
#include <vector>
#include <QOpenGLBuffer>
#include <QOpenGLVertexArrayObject>
#include <QOpenGLShaderProgram>
#include <QOpenGLTexture>
```

Classes

- class Model

  Class for model. 3D model loaded as an OBJ. Has a reference to the OBJ instance, the buffers (vertex and UVs) buffers. Contains model array object, shader program and texture.

7.18 libs/vulkan-engine/src/mesh.cpp File Reference

```
#include "mesh.h"
```

7.19 libs/vulkan-engine/src/mesh.h File Reference

```
#include <QString>
#include <QFile>
```

Classes

- class Mesh

  Class for mesh. Loads the 3D model (mesh). Keeps track of vertex count and saves data in a QByteArray.

7.20 libs/vulkan-engine/src/renderer.cpp File Reference

```
#include "renderer.h"
```

7.21 libs/vulkan-engine/src/renderer.h File Reference

```
#include <QVulkanWindow>
#include <QVulkanFunctions>
#include <QVector>
#include <QSize>
#include <QTime>
#include <QElapsedTimer>
#include "shader.h"
#include "mesh.h"
#include "libs/Profiler.h"
```

Classes

- class Renderer

  Class for renderer. Renderer implementation based on the QVulkanWindowRenderer widget.

7.22 libs/vulkan-engine/src/setvkinfo.cpp File Reference

```
#include "renderer.h"
```

7.23 libs/vulkan-engine/src/shader.cpp File Reference

```
#include "shader.h"
#include <QFile>
#include <QVulkanDeviceFunctions>
```

7.24 libs/vulkan-engine/src/shader.h File Reference

```
#include <QVulkanInstance>
```

Classes

- class Shader

  Class for shader. Loads a shader for file so that it can be used by the Vulkan Engine.

7.25 libs/vulkan-engine/src/vulkanwindow.cpp File Reference

```
#include "vulkanwindow.h"
```

7.26 libs/vulkan-engine/src/vulkanwindow.h File Reference

```
#include "renderer.h"
#include <QLoggingCategory>
#include <QVulkanInstance>
#include "libs/Profiler.h"
```

Classes

- class VulkanWindow

  Class for vulkan window. Vulkan Window class based on QVulkanWindow widget.

7.27 src/gldescriptions.hpp File Reference

```
#include <QString>
```

Classes

- struct GLDescriptions

  OpenGL variables descriptions. Structure containing the descriptions of the OpenGL variables.

7.28 src/main.cpp File Reference

```
#include <QApplication>
#include "mainwindow.h"
```

Functions

- int main (int argc, char ∗argv[ ])

7.28.1 Function Documentation

7.28.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

7.29 libs/opengl-engine/src/main.cpp File Reference

```
#include "glwindow.h"
#include <QApplication>
```

Functions

- int main (int argc, char ∗argv[ ])

7.29.1 Function Documentation

7.29.1.1 main()

```
int main (
            int argc,
            char * argv[ ] )
```

7.30 libs/vulkan-engine/src/main.cpp File Reference

```
#include <QGuiApplication>
#include "vulkanwindow.h"
```

Functions

- int main (int argc, char ∗argv[ ])

7.30.1 Function Documentation

7.30.1.1 main()

```
int main (
            int argc,
            char * argv[ ] )
```

7.31 libs/opengl-engine/libs/triangulate/src/Main.cpp File Reference

```
#include "Triangulate.hpp"
```

Functions

- int main (int argc, char const *argv[])

7.31.1 Function Documentation

7.31.1.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

7.32 src/mainwindow.cpp File Reference

```
#include "mainwindow.h"
#include "ui_toolwindow.h"
#include <QSysInfo>
```

7.33 src/mainwindow.h File Reference

```
#include <QMainWindow>
#include <QTableWidgetItem>
#include <QThread>
#include <QSpinBox>
#include <QtCharts>
#include "libs/glwindow.h"
#include "libs/vulkanwindow.h"
```

Classes

- struct Chart

  Struct used to represent a chart. Contains QChartView, QChart, QValueAxis and QLineSeries.
- class MainWindow

  Class for main window. Main Window based on QMainWindow Widget. Contains the profilers and the charts.

Namespaces

- Ui

7.34 src/runners.cpp File Reference

```
#include "mainwindow.h"
#include "ui_toolwindow.h"
```

7.35 src/updateGLresults.cpp File Reference

```
#include "mainwindow.h"
#include "ui_toolwindow.h"
#include "gldescriptions.hpp"
```

Macros

- #define DISPLAY(r) r != -1 ? QString::number(r) : "No data"

7.35.1 Macro Definition Documentation

7.35.1.1 DISPLAY

```
#define DISPLAY(
             r ) r != -1 ?  QString::number(r) :  "No data"
```

7.36 src/updateVKresults.cpp File Reference

```
#include "mainwindow.h"
#include "ui_toolwindow.h"
#include "vkdescriptions.hpp"
```

Macros

- #define DISPLAY(r) static_cast<int>(r) != -1 ? QString::number(r) : "No data"

7.36.1 Macro Definition Documentation

7.36.1.1 DISPLAY

```
#define DISPLAY(
              r ) static_cast<int>(r) != -1 ?  QString::number(r) :  "No data"
```

7.37 src/vkdescriptions.hpp File Reference

```
#include <QString>
```

Classes

- struct VkDescriptions

  Vulkan variables descriptions. Structure containing the descriptions of the Vulkan variables.