TouchGFX v4.5.0 User's Manual

(C)opyright Draupner Graphics A/S 2014-2016

www.draupnergraphics.com
www.touchgfx.com

Contents

1 TouchGFX Manual **1**

2 The TouchGFX Distribution **5**

2.1 Updating your TouchGFX installation . . . 5

2.2 Distribution types . . . 5

2.3 Main Component Model . . . 5

2.4 Supported Hardware . . . 6

2.5 TouchGFX File Structure . . . 7

2.5.1 Applications (app/) . . . 7

2.5.2 Documentation (doc/) . . . 8

2.5.3 TouchGFX (touchgfx/) . . . 8

2.5.3.1 Board Packages (touchgfx/board/) . . . 8

2.5.3.2 Framework (touchgfx/framework/) . . . 9

2.5.3.3 Precompiled libraries (touchgfx/lib/) . . . 9

2.5.3.4 RT Operating System (touchgfx/os/) . . . 10

3 Development Environment **11**

4 Build and Execute **13**

4.1 Quickstart . . . 13

4.2 Make . . . 13

4.3 PC Simulator . . . 15

4.3.1 MinGW / GCC . . . 16

4.3.2 Visual Studio . . . 16

4.4 Keil Compiler Cross-Version Compatibility . . . 16

4.5 Target Hardware . . . 17

4.5.1 Building and downloading applications with IAR . . . 18

4.5.2 Building and downloading applications with Keil uVision . . . 19

4.5.2.1 Downloading Applications for ST ST3249xI eval board (4.3" and 5.7" display) with Keil uVision4 . . . 21

4.5.2.2 Downloading Applications for ST STM32756G-EVAL and STM32F746G-DISCO boards with Keil uVision5 . . . 21

4.5.3 GCC Building applications with GNU GCC compiler for ARM . . . 22

4.5.4 Programming TouchGFX demo board 4.3" LPC4350 when Compiling with GCC 23
4.5.5 Programming TouchGFX demo board 4.3"/7.0" LPC4353 when Compiling with GCC 28
4.5.6 Programming ST Boards when Compiling with GCC 30

5 Application Development 35

5.1 Application File Structure 35
5.1.1 Application assets 36
5.1.1.1 Font assets 36
5.1.1.2 Image assets 36
5.1.1.3 Text assets 36
5.1.2 Application configuration 37
5.1.3 Graphical User Interface 37
5.1.4 Platform Configuration 37
5.1.5 PC Simulator 37
5.1.6 Target 37
5.2 Creating a New Application 38
5.2.1 Visual Studio Project Creation 38
5.2.1.1 Starting from scratch 39
5.3 Project Configuration 39
5.3.1 GCC 40
5.3.2 Microsoft Visual Studio 2010/2012 40
5.3.3 Screen Orientation 41
5.3.4 Image Output Format 41
5.3.5 Alpha Dithering 41
5.3.6 Text Format 42
5.4 Assets and Tools 42
5.4.1 Font Converter 42
5.4.1.1 Kerning support 42
5.4.2 Image Converter 43
5.4.3 Text Converter 43
5.4.3.1 Text Database 43
5.5 Concepts 45
5.5.1 Model-View-Presenter 45
5.5.1.1 Model 45
5.5.1.2 View 46
5.5.1.3 Presenter 46
5.5.1.4 FrontendApplication 46
5.5.1.5 Switching Screens 46
5.5.1.6 Class Diagram, TouchGFX MVP 46
5.5.2 Drawables 47

5.5.2.1 Adding Drawables to a View . . . 48
5.5.2.2 Coordinate Systems . . . 49
5.5.2.3 Size and Position . . . 50
5.5.2.4 Limitations on the number of drawables . . . 50
5.5.3 Events . . . 50
5.5.3.1 Input Events . . . 50
5.5.3.2 Drag Events . . . 51
5.5.3.3 Gesture Events . . . 51
5.5.3.4 Other Input Events . . . 52
5.5.3.5 Timers . . . 52
5.5.3.6 External Events . . . 52
5.5.4 Memory Allocation . . . 52
5.5.5 Custom Widgets and Containers . . . 53
5.5.5.1 Tweaking Existing Widgets . . . 54
5.5.5.2 Custom Container . . . 55
5.5.5.3 Custom Widget . . . 55
5.5.6 Canvas Widgets . . . 57
5.5.6.1 Using Canvas Widgets . . . 57
5.5.6.2 Memory allocation and usage . . . 58
5.5.6.3 The Coordinate System . . . 58
5.5.6.4 Custom Canvas Widgets . . . 59
5.5.6.5 Painters . . . 60
5.5.7 Mixins . . . 60
5.5.8 Using Text and Fonts . . . 61
5.5.8.1 Using Texts in Code . . . 61
5.5.8.2 Wildcards . . . 61
5.5.8.3 Switching Language . . . 62
5.5.8.4 Conversion Process . . . 62
5.5.9 Using Images . . . 62
5.5.9.1 Using Images in Code . . . 62
5.5.9.2 The Conversion Process . . . 63
5.5.10 Using Callbacks . . . 63
5.6 Widgets, Containers, Mixins, Canvas Widgets . . . 65
5.6.1 Widgets . . . 65
5.6.2 Containers . . . 65
5.6.3 Mixins . . . 66
5.6.4 Canvas Widgets . . . 66
5.6.5 Painters . . . 66
5.7 TouchGFX Open Repository . . . 66
5.8 Miscellaneous . . . 66

5.8.1 Screenshots . . . 66

6 Examples **67**

6.1 Animation Example . . . 67
6.1.1 Framework Elements Used . . . 67
6.1.2 Example Location . . . 68
6.2 Arabic Text Example . . . 68
6.2.1 Framework Elements Used . . . 68
6.2.2 Example Location . . . 68
6.3 Button Example . . . 68
6.3.1 Framework Elements Used . . . 69
6.3.2 Example Location . . . 69
6.4 CanvasWidget Example . . . 69
6.4.1 Framework Elements Used . . . 70
6.4.2 Example Location . . . 70
6.5 Custom Transition Example . . . 70
6.5.1 Framework Elements Used . . . 71
6.5.2 Example Location . . . 71
6.6 Custom Widget Example . . . 71
6.6.1 Framework Elements Used . . . 72
6.6.2 Example Location . . . 72
6.7 Display Orientation Example . . . 72
6.7.1 Framework Elements Used . . . 74
6.7.2 Example Location . . . 74
6.8 Drag Example . . . 74
6.8.1 Framework Elements Used . . . 74
6.8.2 Example Location . . . 75
6.9 Dynamic Bitmap Example . . . 75
6.9.1 Framework Elements Used . . . 75
6.9.2 Example Location . . . 75
6.10 List Layout Example . . . 75
6.10.1 Framework Elements Used . . . 76
6.10.2 Example Location . . . 76
6.11 Monochrome Example . . . 76
6.11.1 Framework Elements Used . . . 77
6.11.2 Example Location . . . 77
6.12 Move and Fade Example . . . 77
6.12.1 Framework Elements Used . . . 78
6.12.2 Example Location . . . 78
6.13 Keyboard Example . . . 78

6.13.1 Framework Elements Used . . . 79
6.13.2 Example Location . . . 79
6.14 Pixel Data Example . . . 79
6.14.1 Framework Elements Used . . . 80
6.14.2 Example Location . . . 80
6.15 Radio Button Example . . . 80
6.15.1 Framework Elements Used . . . 81
6.15.2 Example Location . . . 81
6.16 Scale and Zoom Example . . . 81
6.16.1 Framework Elements Used . . . 82
6.16.2 Example Location . . . 82
6.17 Scrollable Container Example . . . 82
6.17.1 Framework Elements Used . . . 83
6.17.2 Example Location . . . 83
6.18 Slider Example . . . 83
6.18.1 Framework Elements Used . . . 84
6.18.2 Example Location . . . 84
6.19 Text Example . . . 84
6.19.1 Framework Elements Used . . . 85
6.19.2 Example Location . . . 85
6.20 Text Rotate Example . . . 85
6.20.1 Framework Elements Used . . . 86
6.20.2 Example Location . . . 86
6.21 Texture Mapper Example . . . 86
6.21.1 Framework Elements Used . . . 87
6.21.2 Example Location . . . 87
6.22 Transition Example . . . 87
6.22.1 Framework Elements Used . . . 88
6.22.2 Example Location . . . 88

7 Demos **89**
7.1 Compiling and running . . . 89
7.2 Demo 2015 (5.7" - 640x480) . . . 89
7.3 Demo 2015 (4.3" - 480x272) . . . 91
7.4 Demo 2014 (4.3" - 480x272) . . . 93
7.5 Demo 2014 small (2.4" - 240x320) . . . 95
7.6 Home Control Demo (7.0" - 800x480) . . . 97

8 Porting Guide **101**
8.1 Recommended Approach . . . 102
8.2 Initial system . . . 102

8.2.1 Board Bring-up . . . 102
8.2.2 LCD Pin Configuration . . . 103
8.2.3 TFT controller initialization . . . 103
8.2.4 Simple Display Test . . . 103
8.3 Minimalistic TouchGFX System . . . 104
8.3.1 MCU Component . . . 104
8.3.2 OS Abstraction Layer . . . 105
8.3.2.1 VSYNC signaling . . . 106
8.3.2.2 Frame Buffer Access . . . 106
8.3.3 Installing LCD Interrupt Handler . . . 106
8.3.4 BoardConfiguration (Wiring Things Together) . . . 107
8.4 Placing Graphics Data in External Flash . . . 108
8.4.1 Linker Script . . . 108
8.4.2 Flash Loader . . . 108
8.4.3 Separate Projects for Internal and External Flash . . . 108
8.5 Touch Driver . . . 108
8.5.1 Coordinate Transformations . . . 109
8.6 DMA Component . . . 109
8.6.1 Two-dimensional Transfers . . . 111
8.6.2 Optimizing DMA Transfers . . . 111
8.6.3 Example Transfer . . . 112
8.6.4 Interrupt Configuration . . . 112
8.7 Adding a GPIO implementation . . . 113
8.8 Non-memory mapped external flash / Supporting bitmap caching . . . 114
8.8.1 Configuring the Cache . . . 114
8.9 Optimization . . . 115
8.9.1 Optimizing DMA During TFT Controller Access . . . 115
8.9.2 Optimizing VSYNC Timing . . . 115
8.9.3 Touch Sample Method and Frequency . . . 115
8.10 MCUs without TFT controller . . . 116

9 Licensing **117**
9.1 Precompiled Libraries . . . 117
9.2 Examples, templates and demos . . . 118
9.2.1 Exceptions . . . 119
9.2.1.1 FreeRTOS . . . 119
9.2.1.2 libpng . . . 119
9.2.1.3 Hal Related Source Code . . . 119
9.2.1.4 Fonts . . . 119
9.2.1.5 Penner Easing Equations . . . 119

9.3 Font Converter . . . 120
9.3.1 Exceptions . . . 120
9.4 Image Converter . . . 120
9.4.1 Exceptions . . . 121
9.5 Text Converter . . . 121
9.6 Manual . . . 122

10 Advanced **123**
10.1 Bitmap Caching . . . 123
10.1.1 Configuring the Bitmap Cache . . . 123
10.1.2 Using the Bitmap Cache . . . 123
10.2 Changing the Location of Bitmap Data . . . 124
10.3 Timing and TFT Controller Integration . . . 124
10.3.1 Rendering Pass . . . 125
10.3.2 Frame Rate . . . 126
10.3.3 Interleaving MCU- and DMA-based Rendering . . . 128
10.3.4 Number of Frame Buffers . . . 129
10.4 Animation Storage . . . 130
10.5 Blitting Operations and Hardware Blit Capabilities . . . 130
10.6 Languages and Chars . . . 131
10.6.1 Glyph Limitations . . . 131
10.6.2 Text Limitations . . . 131
10.6.3 List of Supported Languages . . . 132
10.6.4 Unsupported Languages . . . 133
10.7 Performance Measurement . . . 133
10.7.1 GPIO . . . 133
10.7.2 MCU Instrumentation . . . 133
10.7.3 HAL . . . 133
10.7.4 OS . . . 134
10.8 Platform Abstraction . . . 134
10.8.1 Button Controllers . . . 135
10.8.2 i2c . . . 136
10.8.3 Touch Controllers . . . 136
10.8.4 HAL . . . 137
10.9 Target Configurations . . . 138
10.10 Configuring TouchGFX for 1BPP Monochrome . . . 139
10.10.1 1BPP Asset Generation . . . 139
10.10.1.1 Image Data . . . 139
10.10.1.2 Compressed Bitmaps . . . 140
10.10.1.3 Font Data . . . 140

10.10.2 Board Configuration . . . 140
10.11 Switching Display Orientation . . . 141
10.11.1 Graphics Data in a Non-Native User Interface Orientation . . . 142
10.11.2 Dynamic Switching at Runtime (supporting both orientations) . . . 142
10.11.2.1 Performing the Switch . . . 142

11 Changelog **145**

12 Known Issues **155**

13 Globals **157**

14 Namespace Index **159**
14.1 Namespace List . . . 159

15 Hierarchical Index **161**
15.1 Class Hierarchy . . . 161

16 Class Index **165**
16.1 Class List . . . 165

17 Namespace Documentation **171**
17.1 touchgfx Namespace Reference . . . 171
17.1.1 Typedef Documentation . . . 179
17.1.1.1 Alignment . . . 179
17.1.1.2 EasingEquation . . . 179
17.1.1.3 TextDirection . . . 179
17.1.2 Enumeration Type Documentation . . . 179
17.1.2.1 BlitOperations . . . 179
17.1.2.2 Direction . . . 179
17.1.2.3 DisplayOrientation . . . 180
17.1.2.4 DisplayRotation . . . 180
17.1.2.5 FrameBuffer . . . 180
17.1.2.6 GlyphFlags . . . 180
17.1.2.7 Gradient . . . 180
17.1.2.8 RenderingVariant . . . 181
17.1.2.9 TextRotation . . . 181
17.1.3 Function Documentation . . . 181
17.1.3.1 abs . . . 181
17.1.3.2 ceil28_4 . . . 181
17.1.3.3 finalizeTransition . . . 181
17.1.3.4 fixed28_4Mul . . . 182
17.1.3.5 fixed28_4ToFloat . . . 182

17.1.3.6 floatToFixed16_16 . . . 182
17.1.3.7 floatToFixed28_4 . . . 182
17.1.3.8 floorDivMod . . . 182
17.1.3.9 hw_init . . . 183
17.1.3.10 lookupBilinearRenderVariant . . . 183
17.1.3.11 lookupNearestNeighborRenderVariant . . . 183
17.1.3.12 makeTransition . . . 183
17.1.3.13 memset . . . 184
17.1.3.14 operator∗ . . . 184
17.1.3.15 operator∗ . . . 184
17.1.3.16 prepareTransition . . . 184
17.1.3.17 touchgfx_generic_init . . . 185
17.1.3.18 touchgfx_init . . . 185
17.1.4 Variable Documentation . . . 185
17.1.4.1 TYPED_TEXT_INVALID . . . 185

18 Class Documentation **187**
18.1 AbstractButton Class Reference . . . 187
18.1.1 Detailed Description . . . 187
18.1.2 Constructor & Destructor Documentation . . . 188
18.1.2.1 AbstractButton . . . 188
18.1.3 Member Function Documentation . . . 188
18.1.3.1 getPressedState . . . 188
18.1.3.2 getType . . . 188
18.1.3.3 handleClickEvent . . . 188
18.1.3.4 setAction . . . 188
18.2 AbstractPainter Class Reference . . . 189
18.2.1 Detailed Description . . . 189
18.2.2 Constructor & Destructor Documentation . . . 189
18.2.2.1 AbstractPainter . . . 189
18.2.2.2 ∼AbstractPainter . . . 189
18.2.3 Member Function Documentation . . . 190
18.2.3.1 render . . . 190
18.2.3.2 setOffset . . . 191
18.2.3.3 setWidgetAlpha . . . 191
18.3 AbstractPainterBW Class Reference . . . 191
18.3.1 Detailed Description . . . 192
18.3.2 Member Function Documentation . . . 192
18.3.2.1 render . . . 192
18.3.2.2 renderInit . . . 192

18.3.2.3 renderNext . . . 192
18.4 AbstractPainterRGB565 Class Reference . . . 192
18.4.1 Detailed Description . . . 193
18.4.2 Member Function Documentation . . . 193
18.4.2.1 render . . . 193
18.4.2.2 renderInit . . . 193
18.4.2.3 renderNext . . . 193
18.4.2.4 renderPixel . . . 194
18.5 AbstractPainterRGB888 Class Reference . . . 194
18.5.1 Detailed Description . . . 195
18.5.2 Member Function Documentation . . . 195
18.5.2.1 render . . . 195
18.5.2.2 renderInit . . . 195
18.5.2.3 renderNext . . . 195
18.5.2.4 renderPixel . . . 195
18.6 AbstractPartition Class Reference . . . 196
18.6.1 Detailed Description . . . 197
18.6.2 Constructor & Destructor Documentation . . . 197
18.6.2.1 ∼AbstractPartition . . . 197
18.6.2.2 AbstractPartition . . . 197
18.6.3 Member Function Documentation . . . 197
18.6.3.1 allocate . . . 197
18.6.3.2 allocate . . . 197
18.6.3.3 allocateAt . . . 198
18.6.3.4 allocateAt . . . 198
18.6.3.5 at . . . 198
18.6.3.6 at . . . 199
18.6.3.7 capacity . . . 199
18.6.3.8 clear . . . 199
18.6.3.9 dec . . . 199
18.6.3.10 element . . . 199
18.6.3.11 element . . . 200
18.6.3.12 element_size . . . 201
18.6.3.13 find . . . 201
18.6.3.14 getAllocationCount . . . 201
18.6.3.15 indexOf . . . 201
18.7 AbstractShape Class Reference . . . 202
18.7.1 Detailed Description . . . 203
18.7.2 Constructor & Destructor Documentation . . . 203
18.7.2.1 AbstractShape . . . 203

18.7.2.2 ~AbstractShape . . . 203
18.7.3 Member Function Documentation . . . 203
18.7.3.1 drawCanvasWidget . . . 203
18.7.3.2 getAngle . . . 204
18.7.3.3 getCacheX . . . 204
18.7.3.4 getCacheY . . . 204
18.7.3.5 getCornerX . . . 204
18.7.3.6 getCornerY . . . 205
18.7.3.7 getMinimalRect . . . 206
18.7.3.8 getNumPoints . . . 206
18.7.3.9 getOrigin . . . 206
18.7.3.10 moveOrigin . . . 206
18.7.3.11 setAngle . . . 207
18.7.3.12 setCache . . . 207
18.7.3.13 setCorner . . . 207
18.7.3.14 setOrigin . . . 208
18.7.3.15 setScale . . . 208
18.7.3.16 setScale . . . 208
18.7.3.17 setShape . . . 209
18.7.3.18 updateAbstractShapeCache . . . 209
18.7.3.19 updateAngle . . . 209
18.8 AnimatedImage Class Reference . . . 209
18.8.1 Detailed Description . . . 211
18.8.2 Constructor & Destructor Documentation . . . 211
18.8.2.1 AnimatedImage . . . 211
18.8.2.2 AnimatedImage . . . 211
18.8.3 Member Function Documentation . . . 211
18.8.3.1 getType . . . 211
18.8.3.2 handleTickEvent . . . 211
18.8.3.3 isReverse . . . 212
18.8.3.4 isRunning . . . 212
18.8.3.5 pauseAnimation . . . 212
18.8.3.6 setBitmaps . . . 212
18.8.3.7 setDoneAction . . . 212
18.8.3.8 setUpdateTicksInterval . . . 212
18.8.3.9 startAnimation . . . 213
18.8.3.10 stopAnimation . . . 213
18.9 Application Class Reference . . . 213
18.9.1 Detailed Description . . . 215
18.9.2 Constructor & Destructor Documentation . . . 215

18.9.2.1 Application . . . 215
18.9.3 Member Function Documentation . . . 215
18.9.3.1 appSwitchScreen . . . 215
18.9.3.2 cacheDrawOperations . . . 215
18.9.3.3 clearAllTimerWidgets . . . 215
18.9.3.4 draw . . . 215
18.9.3.5 draw . . . 215
18.9.3.6 getCurrentScreen . . . 216
18.9.3.7 getInstance . . . 216
18.9.3.8 handleClickEvent . . . 216
18.9.3.9 handleDragEvent . . . 216
18.9.3.10 handleGestureEvent . . . 216
18.9.3.11 handleKeyEvent . . . 217
18.9.3.12 handlePendingScreenTransition . . . 217
18.9.3.13 handleTickEvent . . . 217
18.9.3.14 invalidateArea . . . 217
18.9.3.15 registerTimerWidget . . . 217
18.9.3.16 switchScreen . . . 218
18.9.3.17 unregisterTimerWidget . . . 219
18.9.4 Member Data Documentation . . . 219
18.9.4.1 instance . . . 219
18.9.4.2 MAX_TIMER_WIDGETS . . . 219
18.10 Bitmap Class Reference . . . 219
18.10.1 Detailed Description . . . 221
18.10.2 Member Enumeration Documentation . . . 221
18.10.2.1 BitmapFormat . . . 221
18.10.3 Constructor & Destructor Documentation . . . 221
18.10.3.1 Bitmap . . . 221
18.10.4 Member Function Documentation . . . 221
18.10.4.1 cache . . . 221
18.10.4.2 cacheAll . . . 222
18.10.4.3 cacheGetAddress . . . 222
18.10.4.4 cacheIsCached . . . 222
18.10.4.5 cacheRemoveBitmap . . . 223
18.10.4.6 cacheReplaceBitmap . . . 223
18.10.4.7 clearCache . . . 223
18.10.4.8 dynamicBitmapCreate . . . 223
18.10.4.9 dynamicBitmapDelete . . . 223
18.10.4.10 dynamicBitmapGetAddress . . . 224
18.10.4.11 dynamicBitmapSetSolidRect . . . 224

18.10.4.12 getAlphaData . . . 224
18.10.4.13 getData . . . 224
18.10.4.14 getFormat . . . 225
18.10.4.15 getHeight . . . 225
18.10.4.16 getId . . . 225
18.10.4.17 getRect . . . 225
18.10.4.18 getSolidRect . . . 225
18.10.4.19 getWidth . . . 225
18.10.4.20 hasTransparentPixels . . . 225
18.10.4.21 isAlphaPerPixel . . . 226
18.10.4.22 operator"!= . . . 226
18.10.4.23 operator== . . . 226
18.10.4.24 registerBitmapDatabase . . . 226
18.10.4.25 setCache . . . 226
18.11 Bitmap::BitmapData Struct Reference . . . 227
18.11.1 Detailed Description . . . 227
18.12 BlitOp Struct Reference . . . 227
18.12.1 Detailed Description . . . 228
18.12.2 Member Data Documentation . . . 228
18.12.2.1 operation . . . 228
18.13 Box Class Reference . . . 228
18.13.1 Detailed Description . . . 229
18.13.2 Constructor & Destructor Documentation . . . 229
18.13.2.1 Box . . . 229
18.13.2.2 Box . . . 229
18.13.3 Member Function Documentation . . . 229
18.13.3.1 draw . . . 229
18.13.3.2 forceReportAsSolid . . . 229
18.13.3.3 getAlpha . . . 230
18.13.3.4 getColor . . . 230
18.13.3.5 getSolidRect . . . 230
18.13.3.6 getType . . . 230
18.13.3.7 setAlpha . . . 230
18.13.3.8 setColor . . . 231
18.13.4 Member Data Documentation . . . 231
18.13.4.1 reportAsSolid . . . 231
18.14 Button Class Reference . . . 231
18.14.1 Detailed Description . . . 232
18.14.2 Constructor & Destructor Documentation . . . 232
18.14.2.1 Button . . . 232

18.14.2.2 ~Button . . . 232
18.14.3 Member Function Documentation . . . 232
18.14.3.1 draw . . . 232
18.14.3.2 getAlpha . . . 232
18.14.3.3 getCurrentlyDisplayedBitmap . . . 232
18.14.3.4 getSolidRect . . . 233
18.14.3.5 getType . . . 233
18.14.3.6 setAlpha . . . 233
18.14.3.7 setBitmaps . . . 233
18.15 ButtonController Class Reference . . . 233
18.15.1 Detailed Description . . . 234
18.15.2 Constructor & Destructor Documentation . . . 234
18.15.2.1 ~ButtonController . . . 234
18.15.3 Member Function Documentation . . . 234
18.15.3.1 init . . . 234
18.15.3.2 reset . . . 234
18.15.3.3 sample . . . 234
18.16 Buttons Class Reference . . . 234
18.16.1 Member Function Documentation . . . 235
18.16.1.1 init . . . 235
18.16.1.2 sample . . . 235
18.17 ButtonWithIcon Class Reference . . . 235
18.17.1 Detailed Description . . . 236
18.17.2 Member Function Documentation . . . 236
18.17.2.1 draw . . . 236
18.17.2.2 getCurrentlyDisplayedIcon . . . 236
18.17.2.3 getIconX . . . 236
18.17.2.4 getIconY . . . 236
18.17.2.5 getType . . . 236
18.17.2.6 setBitmaps . . . 237
18.17.2.7 setIconX . . . 238
18.17.2.8 setIconXY . . . 238
18.17.2.9 setIconY . . . 238
18.18 ButtonWithLabel Class Reference . . . 238
18.18.1 Detailed Description . . . 239
18.18.2 Constructor & Destructor Documentation . . . 239
18.18.2.1 ButtonWithLabel . . . 239
18.18.3 Member Function Documentation . . . 240
18.18.3.1 draw . . . 240
18.18.3.2 getLabelRotation . . . 240

18.18.3.3 getLabelText . . . 240
18.18.3.4 getSolidRect . . . 240
18.18.3.5 getType . . . 240
18.18.3.6 setLabelColor . . . 240
18.18.3.7 setLabelColorPressed . . . 241
18.18.3.8 setLabelRotation . . . 241
18.18.3.9 setLabelText . . . 241
18.18.3.10updateTextPosition . . . 241
18.19Bitmap::CacheTableEntry Struct Reference . . . 241
18.19.1 Detailed Description . . . 241
18.20Callback< dest_type, T1, T2 > Struct Template Reference . . . 242
18.20.1 Detailed Description . . . 242
18.20.2 Constructor & Destructor Documentation . . . 242
18.20.2.1 Callback . . . 242
18.20.2.2 Callback . . . 242
18.20.3 Member Function Documentation . . . 243
18.20.3.1 execute . . . 243
18.20.3.2 isValid . . . 243
18.21Callback< dest_type, T1, void > Struct Template Reference . . . 243
18.21.1 Detailed Description . . . 243
18.21.2 Constructor & Destructor Documentation . . . 244
18.21.2.1 Callback . . . 244
18.21.2.2 Callback . . . 244
18.21.3 Member Function Documentation . . . 244
18.21.3.1 execute . . . 244
18.21.3.2 isValid . . . 244
18.22Callback< dest_type, void, void > Struct Template Reference . . . 245
18.22.1 Detailed Description . . . 245
18.22.2 Constructor & Destructor Documentation . . . 245
18.22.2.1 Callback . . . 245
18.22.2.2 Callback . . . 245
18.22.3 Member Function Documentation . . . 246
18.22.3.1 execute . . . 246
18.22.3.2 isValid . . . 246
18.23Keyboard::CallbackArea Struct Reference . . . 246
18.24Canvas Class Reference . . . 246
18.24.1 Detailed Description . . . 247
18.24.2 Constructor & Destructor Documentation . . . 247
18.24.2.1 Canvas . . . 247
18.24.2.2 ~Canvas . . . 247

18.24.3 Member Function Documentation . . . 247
18.24.3.1 lineTo . . . 247
18.24.3.2 lineTo . . . 248
18.24.3.3 moveTo . . . 249
18.24.3.4 moveTo . . . 249
18.24.3.5 render . . . 249
18.25 CanvasWidget Class Reference . . . 249
18.25.1 Detailed Description . . . 250
18.25.2 Constructor & Destructor Documentation . . . 250
18.25.2.1 CanvasWidget . . . 250
18.25.2.2 ∼CanvasWidget . . . 250
18.25.3 Member Function Documentation . . . 250
18.25.3.1 draw . . . 250
18.25.3.2 drawCanvasWidget . . . 251
18.25.3.3 getAlpha . . . 251
18.25.3.4 getMinimalRect . . . 251
18.25.3.5 getPainter . . . 252
18.25.3.6 getSolidRect . . . 252
18.25.3.7 invalidate . . . 252
18.25.3.8 setAlpha . . . 252
18.25.3.9 setPainter . . . 252
18.26 CanvasWidgetRenderer Class Reference . . . 253
18.26.1 Detailed Description . . . 253
18.26.2 Member Function Documentation . . . 254
18.26.2.1 getMissingBufferSize . . . 254
18.26.2.2 getOutlineBuffer . . . 254
18.26.2.3 getOutlineBufferSize . . . 254
18.26.2.4 getScanlineCounts . . . 254
18.26.2.5 getScanlineCovers . . . 254
18.26.2.6 getScanlineStartIndices . . . 254
18.26.2.7 getScanlineWidth . . . 255
18.26.2.8 getUsedBufferSize . . . 255
18.26.2.9 hasBuffer . . . 255
18.26.2.10 numCellsMissing . . . 255
18.26.2.11 numCellsUsed . . . 255
18.26.2.12 setScanlineWidth . . . 255
18.26.2.13 setupBuffer . . . 256
18.27 Cell Struct Reference . . . 256
18.27.1 Detailed Description . . . 256
18.27.2 Member Function Documentation . . . 257

18.27.2.1 addCover . . . 257
18.27.2.2 packedCoord . . . 257
18.27.2.3 set . . . 257
18.27.2.4 setCoord . . . 257
18.27.2.5 setCover . . . 257
18.28 Circle Class Reference . . . 257
18.28.1 Detailed Description . . . 258
18.28.2 Constructor & Destructor Documentation . . . 259
18.28.2.1 Circle . . . 259
18.28.3 Member Function Documentation . . . 259
18.28.3.1 drawCanvasWidget . . . 259
18.28.3.2 getArc . . . 259
18.28.3.3 getArcEnd . . . 259
18.28.3.4 getArcStart . . . 259
18.28.3.5 getCenter . . . 260
18.28.3.6 getMinimalRect . . . 260
18.28.3.7 getMinimalRect . . . 260
18.28.3.8 getRadius . . . 260
18.28.3.9 setArc . . . 261
18.28.3.10 setCapPrecision . . . 261
18.28.3.11 setCenter . . . 261
18.28.3.12 setCircle . . . 262
18.28.3.13 setLineWidth . . . 262
18.28.3.14 setPrecision . . . 262
18.28.3.15 setRadius . . . 262
18.28.3.16 updateArcEnd . . . 263
18.28.3.17 updateArcStart . . . 263
18.29 ClickEvent Class Reference . . . 263
18.29.1 Detailed Description . . . 264
18.29.2 Member Enumeration Documentation . . . 264
18.29.2.1 ClickEventType . . . 264
18.29.3 Constructor & Destructor Documentation . . . 264
18.29.3.1 ClickEvent . . . 264
18.29.3.2 ~ClickEvent . . . 265
18.29.4 Member Function Documentation . . . 265
18.29.4.1 getEventType . . . 265
18.29.4.2 getForce . . . 265
18.29.4.3 getType . . . 265
18.29.4.4 getX . . . 265
18.29.4.5 getY . . . 265

18.29.4.6 setType . . . 266
18.29.4.7 setX . . . 267
18.29.4.8 setY . . . 267
18.30 ClickListener< T > Class Template Reference . . . 267
18.30.1 Detailed Description . . . 267
18.30.2 Constructor & Destructor Documentation . . . 268
18.30.2.1 ClickListener . . . 268
18.30.3 Member Function Documentation . . . 268
18.30.3.1 handleClickEvent . . . 268
18.30.3.2 setClickAction . . . 268
18.31 Color Class Reference . . . 268
18.31.1 Detailed Description . . . 269
18.31.2 Member Function Documentation . . . 269
18.31.2.1 getBlueColor . . . 269
18.31.2.2 getColorFrom24BitRGB . . . 269
18.31.2.3 getGreenColor . . . 269
18.31.2.4 getRedColor . . . 269
18.32 colortype Struct Reference . . . 270
18.32.1 Detailed Description . . . 270
18.32.2 Constructor & Destructor Documentation . . . 270
18.32.2.1 colortype . . . 270
18.32.2.2 colortype . . . 270
18.32.3 Member Function Documentation . . . 271
18.32.3.1 getColor32 . . . 271
18.32.3.2 operator uint16_t . . . 271
18.33 ConstFont Class Reference . . . 271
18.33.1 Detailed Description . . . 272
18.33.2 Constructor & Destructor Documentation . . . 272
18.33.2.1 ConstFont . . . 272
18.33.3 Member Function Documentation . . . 272
18.33.3.1 find . . . 272
18.33.3.2 getGlyph . . . 272
18.33.3.3 getKerning . . . 273
18.33.3.4 getPixelData . . . 273
18.34 Container Class Reference . . . 273
18.34.1 Detailed Description . . . 274
18.34.2 Constructor & Destructor Documentation . . . 275
18.34.2.1 Container . . . 275
18.34.2.2 ∼Container . . . 275
18.34.3 Member Function Documentation . . . 275

18.34.3.1 add . . . 275
18.34.3.2 contains . . . 275
18.34.3.3 draw . . . 275
18.34.3.4 forEachChild . . . 275
18.34.3.5 getContainedArea . . . 276
18.34.3.6 getLastChild . . . 276
18.34.3.7 getSolidRect . . . 276
18.34.3.8 getType . . . 276
18.34.3.9 insert . . . 277
18.34.3.10 moveChildrenRelative . . . 277
18.34.3.11 remove . . . 277
18.34.3.12 removeAll . . . 277
18.34.3.13 setupDrawChain . . . 277
18.34.3.14 unlink . . . 278
18.35 CortexMMCUInstrumentation Class Reference . . . 278
18.35.1 Detailed Description . . . 278
18.35.2 Member Function Documentation . . . 278
18.35.2.1 getCPUCycles . . . 278
18.35.2.2 getElapsedUS . . . 279
18.35.2.3 init . . . 280
18.35.2.4 setMCUActive . . . 280
18.36 CWRUtil Struct Reference . . . 280
18.36.1 Detailed Description . . . 281
18.36.2 Member Function Documentation . . . 281
18.36.2.1 angle . . . 281
18.36.2.2 angle . . . 281
18.36.2.3 angle . . . 282
18.36.2.4 angle . . . 282
18.36.2.5 arcsine . . . 282
18.36.2.6 cosine . . . 282
18.36.2.7 sine . . . 283
18.36.2.8 sqrtQ10 . . . 283
18.36.2.9 toQ10 . . . 283
18.36.2.10 toQ5 . . . 284
18.37 DisplayTransformation Class Reference . . . 285
18.37.1 Detailed Description . . . 285
18.37.2 Member Function Documentation . . . 285
18.37.2.1 transformDisplayToFrameBuffer . . . 285
18.37.2.2 transformDisplayToFrameBuffer . . . 286
18.37.2.3 transformDisplayToFrameBuffer . . . 286

18.37.2.4 transformDisplayToFrameBuffer . . . 286
18.37.2.5 transformDisplayToFrameBuffer . . . 286
18.37.2.6 transformDisplayToFrameBuffer . . . 286
18.37.2.7 transformFrameBufferToDisplay . . . 286
18.38 DMA_Interface Class Reference . . . 287
18.38.1 Detailed Description . . . 288
18.38.2 Constructor & Destructor Documentation . . . 288
18.38.2.1 ~DMA_Interface . . . 288
18.38.2.2 DMA_Interface . . . 288
18.38.3 Member Function Documentation . . . 288
18.38.3.1 addToQueue . . . 288
18.38.3.2 disableAlpha . . . 288
18.38.3.3 enableAlpha . . . 289
18.38.3.4 enableCopyWithTransparentPixels . . . 290
18.38.3.5 execute . . . 290
18.38.3.6 executeCompleted . . . 290
18.38.3.7 flush . . . 290
18.38.3.8 getAllowed . . . 290
18.38.3.9 getBlitCaps . . . 290
18.38.3.10 initialize . . . 290
18.38.3.11 isDmaQueueEmpty . . . 291
18.38.3.12 isDmaQueueFull . . . 291
18.38.3.13 isDMARunning . . . 291
18.38.3.14 seedExecution . . . 291
18.38.3.15 setAllowed . . . 291
18.38.3.16 setupDataCopy . . . 291
18.38.3.17 setupDataFill . . . 291
18.38.3.18 signalDMAInterrupt . . . 292
18.38.3.19 start . . . 292
18.38.3.20 waitForFrameBufferSemaphore . . . 292
18.39 DMA_Queue Class Reference . . . 292
18.39.1 Detailed Description . . . 293
18.39.2 Constructor & Destructor Documentation . . . 293
18.39.2.1 ~DMA_Queue . . . 293
18.39.2.2 DMA_Queue . . . 293
18.39.3 Member Function Documentation . . . 293
18.39.3.1 first . . . 293
18.39.3.2 isEmpty . . . 293
18.39.3.3 isFull . . . 293
18.39.3.4 pop . . . 293

18.39.3.5 pushCopyOf . . . 293
18.40 DragEvent Class Reference . . . 294
18.40.1 Detailed Description . . . 294
18.40.2 Constructor & Destructor Documentation . . . 295
18.40.2.1 DragEvent . . . 295
18.40.2.2 ∼DragEvent . . . 296
18.40.3 Member Function Documentation . . . 296
18.40.3.1 getDeltaX . . . 296
18.40.3.2 getDeltaY . . . 296
18.40.3.3 getEventType . . . 296
18.40.3.4 getNewX . . . 296
18.40.3.5 getNewY . . . 296
18.40.3.6 getOldX . . . 297
18.40.3.7 getOldY . . . 297
18.40.3.8 getType . . . 297
18.41 Draggable< T > Class Template Reference . . . 297
18.41.1 Detailed Description . . . 297
18.41.2 Constructor & Destructor Documentation . . . 297
18.41.2.1 Draggable . . . 297
18.41.2.2 ∼Draggable . . . 298
18.41.3 Member Function Documentation . . . 298
18.41.3.1 handleDragEvent . . . 298
18.42 Drawable Class Reference . . . 298
18.42.1 Detailed Description . . . 301
18.42.2 Member Enumeration Documentation . . . 301
18.42.2.1 DrawableType . . . 301
18.42.3 Constructor & Destructor Documentation . . . 301
18.42.3.1 Drawable . . . 301
18.42.3.2 ∼Drawable . . . 301
18.42.4 Member Function Documentation . . . 301
18.42.4.1 childGeometryChanged . . . 301
18.42.4.2 draw . . . 301
18.42.4.3 getAbsoluteRect . . . 302
18.42.4.4 getCachedAbsX . . . 302
18.42.4.5 getCachedAbsY . . . 302
18.42.4.6 getCachedVisibleRect . . . 302
18.42.4.7 getHeight . . . 302
18.42.4.8 getLastChild . . . 302
18.42.4.9 getNextSibling . . . 303
18.42.4.10 getParent . . . 303

18.42.4.11 getRect . . . 303
18.42.4.12 getSolidRect . . . 303
18.42.4.13 getSolidRectAbsolute . . . 303
18.42.4.14 getType . . . 304
18.42.4.15 getVisibleRect . . . 304
18.42.4.16 getWidth . . . 304
18.42.4.17 getX . . . 304
18.42.4.18 getY . . . 304
18.42.4.19 handleClickEvent . . . 304
18.42.4.20 handleDragEvent . . . 305
18.42.4.21 handleGestureEvent . . . 305
18.42.4.22 handleTickEvent . . . 305
18.42.4.23 invalidate . . . 305
18.42.4.24 invalidateRect . . . 305
18.42.4.25 isTouchable . . . 306
18.42.4.26 isVisible . . . 306
18.42.4.27 moveRelative . . . 306
18.42.4.28 moveTo . . . 306
18.42.4.29 resetDrawChainCache . . . 307
18.42.4.30 setHeight . . . 307
18.42.4.31 setPosition . . . 307
18.42.4.32 setTouchable . . . 307
18.42.4.33 setupDrawChain . . . 307
18.42.4.34 setVisible . . . 308
18.42.4.35 setWidth . . . 308
18.42.4.36 setX . . . 308
18.42.4.37 setXY . . . 308
18.42.4.38 setY . . . 309
18.42.4.39 translateRectToAbsolute . . . 309
18.43 DrawingSurface Struct Reference . . . 309
18.44 Bitmap::DynamicBitmapData Struct Reference . . . 309
18.44.1 Detailed Description . . . 310
18.45 EasingEquations Class Reference . . . 310
18.45.1 Detailed Description . . . 311
18.45.2 Member Function Documentation . . . 312
18.45.2.1 backEaseIn . . . 312
18.45.2.2 backEaseInOut . . . 313
18.45.2.3 backEaseOut . . . 313
18.45.2.4 bounceEaseIn . . . 313
18.45.2.5 bounceEaseInOut . . . 314

18.45.2.6 bounceEaseOut . . . 314
18.45.2.7 circEaseIn . . . 314
18.45.2.8 circEaseInOut . . . 314
18.45.2.9 circEaseOut . . . 315
18.45.2.10 cubicEaseIn . . . 315
18.45.2.11 cubicEaseInOut . . . 315
18.45.2.12 cubicEaseOut . . . 316
18.45.2.13 elasticEaseIn . . . 317
18.45.2.14 elasticEaseInOut . . . 317
18.45.2.15 elasticEaseOut . . . 317
18.45.2.16 expoEaseIn . . . 318
18.45.2.17 expoEaseInOut . . . 319
18.45.2.18 expoEaseOut . . . 319
18.45.2.19 linearEaseIn . . . 319
18.45.2.20 linearEaseInOut . . . 320
18.45.2.21 linearEaseNone . . . 321
18.45.2.22 linearEaseOut . . . 321
18.45.2.23 quadEaseIn . . . 321
18.45.2.24 quadEaseInOut . . . 321
18.45.2.25 quadEaseOut . . . 322
18.45.2.26 quartEaseIn . . . 322
18.45.2.27 quartEaseInOut . . . 322
18.45.2.28 quartEaseOut . . . 323
18.45.2.29 quintEaseIn . . . 324
18.45.2.30 quintEaseInOut . . . 324
18.45.2.31 quintEaseOut . . . 324
18.45.2.32 sineEaseIn . . . 325
18.45.2.33 sineEaseInOut . . . 326
18.45.2.34 sineEaseOut . . . 326
18.46 Edge Struct Reference . . . 326
18.46.1 Constructor & Destructor Documentation . . . 327
18.46.1.1 Edge . . . 327
18.46.2 Member Function Documentation . . . 328
18.46.2.1 step . . . 328
18.46.2.2 step . . . 328
18.47 Event Class Reference . . . 328
18.47.1 Detailed Description . . . 328
18.47.2 Member Enumeration Documentation . . . 328
18.47.2.1 EventType . . . 328
18.47.3 Constructor & Destructor Documentation . . . 329

18.47.3.1 ∼Event . . . 329
18.47.4 Member Function Documentation . . . 329
18.47.4.1 getEventType . . . 329
18.48 FadeAnimator< T > Class Template Reference . . . 329
18.48.1 Detailed Description . . . 330
18.48.2 Constructor & Destructor Documentation . . . 330
18.48.2.1 FadeAnimator . . . 330
18.48.2.2 ∼FadeAnimator . . . 330
18.48.3 Member Function Documentation . . . 330
18.48.3.1 getFadeAnimationDelay . . . 330
18.48.3.2 handleTickEvent . . . 331
18.48.3.3 isRunning . . . 331
18.48.3.4 setFadeAnimationDelay . . . 331
18.48.3.5 setFadeAnimationEndedAction . . . 331
18.48.3.6 startFadeAnimation . . . 331
18.49 Font Class Reference . . . 331
18.49.1 Detailed Description . . . 333
18.49.2 Constructor & Destructor Documentation . . . 333
18.49.2.1 Font . . . 333
18.49.3 Member Function Documentation . . . 333
18.49.3.1 getBitsPerPixel . . . 333
18.49.3.2 getCharWidth . . . 333
18.49.3.3 getFontHeight . . . 334
18.49.3.4 getGlyph . . . 334
18.49.3.5 getGlyph . . . 334
18.49.3.6 getKerning . . . 334
18.49.3.7 getMaxPixelsLeft . . . 335
18.49.3.8 getMaxPixelsRight . . . 335
18.49.3.9 getMaxTextHeight . . . 335
18.49.3.10 getMinimumTextHeight . . . 335
18.49.3.11 getNumberOfLines . . . 335
18.49.3.12 getSpacingAbove . . . 336
18.49.3.13 getStringWidth . . . 336
18.49.3.14 getStringWidth . . . 336
18.49.3.15 getStringWidthLTR . . . 337
18.49.3.16 getStringWidthRTL . . . 337
18.50 FontManager Class Reference . . . 337
18.50.1 Detailed Description . . . 338
18.50.2 Member Function Documentation . . . 338
18.50.2.1 getFont . . . 338

18.50.2.2 setFontProvider . . . 338
18.51 FontProvider Class Reference . . . 338
18.51.1 Detailed Description . . . 338
18.51.2 Constructor & Destructor Documentation . . . 338
18.51.2.1 ∼FontProvider . . . 338
18.51.3 Member Function Documentation . . . 339
18.51.3.1 getFont . . . 339
18.52 GenericCallback< T1, T2 > Class Template Reference . . . 339
18.52.1 Detailed Description . . . 339
18.52.2 Constructor & Destructor Documentation . . . 340
18.52.2.1 ∼GenericCallback . . . 340
18.52.3 Member Function Documentation . . . 340
18.52.3.1 execute . . . 340
18.52.3.2 isValid . . . 340
18.53 GenericCallback< T1, void > Class Template Reference . . . 340
18.53.1 Detailed Description . . . 341
18.53.2 Constructor & Destructor Documentation . . . 341
18.53.2.1 ∼GenericCallback . . . 341
18.53.3 Member Function Documentation . . . 341
18.53.3.1 execute . . . 341
18.53.3.2 isValid . . . 341
18.54 GenericCallback< void > Class Template Reference . . . 341
18.54.1 Detailed Description . . . 342
18.54.2 Constructor & Destructor Documentation . . . 342
18.54.2.1 ∼GenericCallback . . . 342
18.54.3 Member Function Documentation . . . 342
18.54.3.1 execute . . . 342
18.54.3.2 isValid . . . 342
18.55 GestureEvent Class Reference . . . 342
18.55.1 Detailed Description . . . 343
18.55.2 Member Enumeration Documentation . . . 343
18.55.2.1 GestureType . . . 343
18.55.3 Constructor & Destructor Documentation . . . 343
18.55.3.1 GestureEvent . . . 343
18.55.4 Member Function Documentation . . . 344
18.55.4.1 getEventType . . . 344
18.55.4.2 getType . . . 344
18.55.4.3 getVelocity . . . 344
18.55.4.4 getX . . . 344
18.55.4.5 getY . . . 344

18.56 Gestures Class Reference . . . 344
18.56.1 Detailed Description . . . 345
18.56.2 Constructor & Destructor Documentation . . . 345
18.56.2.1 Gestures . . . 345
18.56.3 Member Function Documentation . . . 345
18.56.3.1 registerClickEvent . . . 345
18.56.3.2 registerDragEvent . . . 345
18.56.3.3 registerEventListener . . . 346
18.56.3.4 setDragThreshold . . . 347
18.56.3.5 tick . . . 347
18.57 GlyphNode Struct Reference . . . 347
18.57.1 Detailed Description . . . 348
18.57.2 Member Function Documentation . . . 348
18.57.2.1 advance . . . 348
18.57.2.2 height . . . 348
18.57.2.3 top . . . 348
18.57.2.4 width . . . 348
18.58 GPIO Class Reference . . . 348
18.58.1 Detailed Description . . . 349
18.58.2 Member Enumeration Documentation . . . 349
18.58.2.1 GPIO_ID . . . 349
18.58.3 Member Function Documentation . . . 349
18.58.3.1 clear . . . 349
18.58.3.2 get . . . 349
18.58.3.3 init . . . 350
18.58.3.4 set . . . 350
18.58.3.5 toggle . . . 350
18.59 Gradients Struct Reference . . . 350
18.59.1 Constructor & Destructor Documentation . . . 351
18.59.1.1 Gradients . . . 351
18.60 HAL Class Reference . . . 351
18.60.1 Detailed Description . . . 355
18.60.2 Constructor & Destructor Documentation . . . 355
18.60.2.1 HAL . . . 355
18.60.2.2 ~HAL . . . 355
18.60.3 Member Function Documentation . . . 356
18.60.3.1 allowDMATransfers . . . 356
18.60.3.2 backPorchExited . . . 356
18.60.3.3 beginFrame . . . 356
18.60.3.4 blitCopy . . . 356

18.60.3.5 blitCopyARGB8888 . . . 356
18.60.3.6 blitCopyGlyph . . . 357
18.60.3.7 blitFill . . . 357
18.60.3.8 blitSetTransparencyKey . . . 357
18.60.3.9 blockCopy . . . 357
18.60.3.10 cacheTextString . . . 358
18.60.3.11 configureInterrupts . . . 358
18.60.3.12 copyFBRegionToMemory . . . 358
18.60.3.13 disableInterrupts . . . 358
18.60.3.14 enableInterrupts . . . 359
18.60.3.15 enableLCDControllerInterrupt . . . 359
18.60.3.16 enableMCULoadCalculation . . . 359
18.60.3.17 endFrame . . . 359
18.60.3.18 flushDMA . . . 359
18.60.3.19 flushFrameBuffer . . . 359
18.60.3.20 flushFrameBuffer . . . 359
18.60.3.21 frontPorchEntered . . . 359
18.60.3.22 getAnimationStorage . . . 360
18.60.3.23 getBlitCaps . . . 360
18.60.3.24 getButtonController . . . 360
18.60.3.25 getClientFrameBuffer . . . 360
18.60.3.26 getCPUCycles . . . 360
18.60.3.27 getDisplayHeight . . . 360
18.60.3.28 getDisplayOrientation . . . 361
18.60.3.29 getDisplayWidth . . . 361
18.60.3.30 getFingerSize . . . 361
18.60.3.31 getInstance . . . 361
18.60.3.32 getLCDRefreshCount . . . 361
18.60.3.33 getMCULoadPct . . . 361
18.60.3.34 getTFTFrameBuffer . . . 362
18.60.3.35 getTouchSampleRate . . . 362
18.60.3.36 initialize . . . 362
18.60.3.37 lcd . . . 362
18.60.3.38 lockDMAToFrontPorch . . . 362
18.60.3.39 lockFrameBuffer . . . 362
18.60.3.40 noTouch . . . 363
18.60.3.41 performDisplayOrientationChange . . . 363
18.60.3.42 registerEventListener . . . 363
18.60.3.43 registerTextCache . . . 363
18.60.3.44 sampleKey . . . 363

18.60.3.45 setButtonController . . . 364
18.60.3.46 setDisplayOrientation . . . 365
18.60.3.47 setDragThreshold . . . 365
18.60.3.48 setFingerSize . . . 365
18.60.3.49 setFrameBufferStartAddress . . . 365
18.60.3.50 setFrameRateCompensation . . . 366
18.60.3.51 setMCUActive . . . 366
18.60.3.52 setMCUInstrumentation . . . 366
18.60.3.53 setTFTFrameBuffer . . . 366
18.60.3.54 setTouchSampleRate . . . 366
18.60.3.55 signalDMAInterrupt . . . 366
18.60.3.56 swapFrameBuffers . . . 367
18.60.3.57 taskEntry . . . 367
18.60.3.58 tick . . . 367
18.60.3.59 touch . . . 367
18.60.3.60 unlockFrameBuffer . . . 367
18.60.3.61 vSync . . . 367
18.61 HALSDL Class Reference . . . 367
18.61.1 Detailed Description . . . 368
18.61.2 Constructor & Destructor Documentation . . . 369
18.61.2.1 HALSDL . . . 369
18.61.3 Member Function Documentation . . . 370
18.61.3.1 blitSetTransparencyKey . . . 370
18.61.3.2 blockCopy . . . 370
18.61.3.3 configureInterrupts . . . 370
18.61.3.4 configureLCDInterrupt . . . 370
18.61.3.5 disableInterrupts . . . 370
18.61.3.6 doSampleTouch . . . 370
18.61.3.7 enableInterrupts . . . 371
18.61.3.8 enableLCDControllerInterrupt . . . 371
18.61.3.9 flushFrameBuffer . . . 371
18.61.3.10 flushFrameBuffer . . . 371
18.61.3.11 getTFTFrameBuffer . . . 371
18.61.3.12 performDisplayOrientationChange . . . 371
18.61.3.13 renderLCD_FrameBufferToMemory . . . 372
18.61.3.14 sampleKey . . . 373
18.61.3.15 saveScreenshot . . . 373
18.61.3.16 sdl_init . . . 373
18.61.3.17 setTFTFrameBuffer . . . 373
18.61.3.18 taskEntry . . . 373

18.62 I2C Class Reference . . . 374
18.62.1 Detailed Description . . . 374
18.62.2 Constructor & Destructor Documentation . . . 374
18.62.2.1 I2C . . . 374
18.62.2.2 ∼I2C . . . 374
18.62.3 Member Function Documentation . . . 375
18.62.3.1 init . . . 375
18.62.3.2 readRegister . . . 375
18.62.3.3 writeRegister . . . 375
18.63 I2C_LPC1788 Class Reference . . . 375
18.63.1 Detailed Description . . . 376
18.63.2 Constructor & Destructor Documentation . . . 376
18.63.2.1 I2C_LPC1788 . . . 376
18.63.3 Member Function Documentation . . . 376
18.63.3.1 init . . . 376
18.63.3.2 readRegister . . . 376
18.63.3.3 writeRegister . . . 376
18.64 I2C_LPC43xx Class Reference . . . 377
18.64.1 Detailed Description . . . 377
18.64.2 Constructor & Destructor Documentation . . . 377
18.64.2.1 I2C_LPC43xx . . . 377
18.64.3 Member Function Documentation . . . 377
18.64.3.1 init . . . 377
18.64.3.2 readRegister . . . 377
18.64.3.3 writeRegister . . . 377
18.65 I2CTouchController Class Reference . . . 378
18.65.1 Detailed Description . . . 378
18.65.2 Constructor & Destructor Documentation . . . 378
18.65.2.1 I2CTouchController . . . 378
18.65.2.2 ∼I2CTouchController . . . 379
18.65.3 Member Function Documentation . . . 379
18.65.3.1 init . . . 379
18.65.3.2 sampleTouch . . . 379
18.66 Image Class Reference . . . 379
18.66.1 Detailed Description . . . 380
18.66.2 Constructor & Destructor Documentation . . . 380
18.66.2.1 Image . . . 380
18.66.3 Member Function Documentation . . . 380
18.66.3.1 draw . . . 380
18.66.3.2 getAlpha . . . 380

18.66.3.3 getBitmap . . . 380
18.66.3.4 getSolidRect . . . 381
18.66.3.5 getType . . . 381
18.66.3.6 setAlpha . . . 381
18.66.3.7 setBitmap . . . 381
18.67 InternalFlashFont Class Reference . . . 381
18.67.1 Detailed Description . . . 382
18.67.2 Constructor & Destructor Documentation . . . 382
18.67.2.1 InternalFlashFont . . . 382
18.67.3 Member Function Documentation . . . 382
18.67.3.1 getKerning . . . 382
18.67.3.2 getPixelData . . . 383
18.68 Scanline::iterator Class Reference . . . 383
18.68.1 Detailed Description . . . 383
18.68.2 Constructor & Destructor Documentation . . . 383
18.68.2.1 iterator . . . 383
18.68.3 Member Function Documentation . . . 384
18.68.3.1 getCovers . . . 384
18.68.3.2 getNumPix . . . 384
18.68.3.3 next . . . 384
18.69 JSMOCHelper Class Reference . . . 384
18.69.1 Detailed Description . . . 385
18.69.2 Constructor & Destructor Documentation . . . 385
18.69.2.1 JSMOCHelper . . . 385
18.69.3 Member Function Documentation . . . 385
18.69.3.1 draw . . . 385
18.69.3.2 getCachedAbsX . . . 385
18.69.3.3 getCachedAbsY . . . 385
18.69.3.4 getCachedVisibleRect . . . 385
18.69.3.5 getHeight . . . 386
18.69.3.6 getWidget . . . 386
18.69.3.7 getWidth . . . 386
18.69.3.8 setWidget . . . 386
18.70 KerningNode Struct Reference . . . 386
18.70.1 Detailed Description . . . 386
18.71 Keyboard::Key Struct Reference . . . 387
18.72 Keyboard Class Reference . . . 387
18.72.1 Detailed Description . . . 389
18.72.2 Constructor & Destructor Documentation . . . 389
18.72.2.1 Keyboard . . . 389

18.72.2.2 ~Keyboard . . . 389
18.72.3 Member Function Documentation . . . 389
18.72.3.1 draw . . . 389
18.72.3.2 getBuffer . . . 389
18.72.3.3 getBufferPosition . . . 390
18.72.3.4 getCallbackAreaForCoordinates . . . 390
18.72.3.5 getCharForKey . . . 390
18.72.3.6 getKeyForCoordinates . . . 390
18.72.3.7 getKeyMappingList . . . 390
18.72.3.8 getLayout . . . 391
18.72.3.9 getType . . . 391
18.72.3.10handleClickEvent . . . 391
18.72.3.11setBuffer . . . 391
18.72.3.12setBufferPosition . . . 391
18.72.3.13setKeyListener . . . 391
18.72.3.14setKeymappingList . . . 392
18.72.3.15setLayout . . . 392
18.72.3.16setTextIndentation . . . 392
18.73Keyboard::KeyMapping Struct Reference . . . 392
18.74Keyboard::KeyMappingList Struct Reference . . . 392
18.75Keyboard::Layout Struct Reference . . . 393
18.76LCD Class Reference . . . 393
18.76.1 Detailed Description . . . 395
18.76.2 Constructor & Destructor Documentation . . . 395
18.76.2.1 ~LCD . . . 395
18.76.3 Member Function Documentation . . . 396
18.76.3.1 bitDepth . . . 396
18.76.3.2 blitCopy . . . 396
18.76.3.3 blitCopy . . . 396
18.76.3.4 copyFrameBufferRegionToMemory . . . 396
18.76.3.5 drawBorder . . . 397
18.76.3.6 drawGlyph . . . 397
18.76.3.7 drawGlyph . . . 397
18.76.3.8 drawHorizontalLine . . . 398
18.76.3.9 drawPartialBitmap . . . 398
18.76.3.10drawRect . . . 398
18.76.3.11drawString . . . 399
18.76.3.12drawStringLTR . . . 399
18.76.3.13drawStringRTL . . . 399
18.76.3.14drawTextureMapScanLine . . . 400

18.76.3.15 drawTextureMapTriangle . . . 400
18.76.3.16 drawVerticalLine . . . 401
18.76.3.17 fillRect . . . 401
18.76.3.18 init . . . 401
18.76.3.19 realX . . . 401
18.76.3.20 realY . . . 401
18.76.3.21 rotateRect . . . 402
18.76.3.22 stringWidth . . . 402
18.77 LCD16bpp Class Reference . . . 402
18.77.1 Detailed Description . . . 403
18.77.2 Member Function Documentation . . . 404
18.77.2.1 bitDepth . . . 404
18.77.2.2 blitCopy . . . 404
18.77.2.3 blitCopy . . . 404
18.77.2.4 blitCopyAlphaPerPixel . . . 405
18.77.2.5 blitCopyARGB8888 . . . 406
18.77.2.6 copyFrameBufferRegionToMemory . . . 406
18.77.2.7 drawGlyph . . . 406
18.77.2.8 drawPartialBitmap . . . 407
18.77.2.9 drawTextureMapScanLine . . . 407
18.77.2.10 fillRect . . . 407
18.77.2.11 init . . . 408
18.77.2.12 nextLine . . . 408
18.77.2.13 nextPixel . . . 408
18.78 LCD1bpp Class Reference . . . 408
18.78.1 Detailed Description . . . 409
18.78.2 Member Function Documentation . . . 409
18.78.2.1 bitDepth . . . 409
18.78.2.2 blitCopy . . . 410
18.78.2.3 blitCopy . . . 410
18.78.2.4 blitCopyRLE . . . 410
18.78.2.5 copyFrameBufferRegionToMemory . . . 411
18.78.2.6 drawGlyph . . . 411
18.78.2.7 drawPartialBitmap . . . 411
18.78.2.8 drawTextureMapScanLine . . . 412
18.78.2.9 fillMemory . . . 412
18.78.2.10 fillRect . . . 412
18.78.2.11 nextLine . . . 412
18.78.2.12 nextPixel . . . 413
18.79 LCD24bpp Class Reference . . . 413

18.79.1 Detailed Description . . . 414
18.79.2 Member Function Documentation . . . 414
18.79.2.1 bitDepth . . . 414
18.79.2.2 blitCopy . . . 414
18.79.2.3 blitCopy . . . 415
18.79.2.4 blitCopyARGB8888 . . . 415
18.79.2.5 copyFrameBufferRegionToMemory . . . 415
18.79.2.6 drawGlyph . . . 416
18.79.2.7 drawPartialBitmap . . . 416
18.79.2.8 drawTextureMapScanLine . . . 416
18.79.2.9 fillRect . . . 417
18.79.2.10 init . . . 417
18.79.2.11 nextLine . . . 417
18.79.2.12 nextPixel . . . 417
18.80 LED Class Reference . . . 418
18.80.1 Member Function Documentation . . . 418
18.80.1.1 get . . . 418
18.80.1.2 init . . . 418
18.80.1.3 off . . . 418
18.80.1.4 on . . . 419
18.80.1.5 toggle . . . 419
18.81 Line Class Reference . . . 419
18.81.1 Detailed Description . . . 420
18.81.2 Member Enumeration Documentation . . . 420
18.81.2.1 LINE_ENDING_STYLE . . . 420
18.81.3 Constructor & Destructor Documentation . . . 420
18.81.3.1 Line . . . 420
18.81.4 Member Function Documentation . . . 421
18.81.4.1 drawCanvasWidget . . . 421
18.81.4.2 getEnd . . . 421
18.81.4.3 getLineEndingStyle . . . 421
18.81.4.4 getLineWidth . . . 421
18.81.4.5 getMinimalRect . . . 422
18.81.4.6 getStart . . . 422
18.81.4.7 setEnd . . . 422
18.81.4.8 setLine . . . 423
18.81.4.9 setLineEndingStyle . . . 423
18.81.4.10 setLineWidth . . . 423
18.81.4.11 setStart . . . 424
18.81.4.12 updateEnd . . . 424

18.81.4.13 updateLineWidth . . . 424
18.81.4.14 updateStart . . . 425
18.82 ListLayout Class Reference . . . 425
18.82.1 Detailed Description . . . 426
18.82.2 Constructor & Destructor Documentation . . . 426
18.82.2.1 ListLayout . . . 426
18.82.2.2 ~ListLayout . . . 426
18.82.3 Member Function Documentation . . . 426
18.82.3.1 add . . . 426
18.82.3.2 getType . . . 426
18.82.3.3 insert . . . 427
18.82.3.4 remove . . . 428
18.82.3.5 removeAll . . . 428
18.83 LockFreeDMA_Queue Class Reference . . . 428
18.83.1 Detailed Description . . . 429
18.83.2 Constructor & Destructor Documentation . . . 429
18.83.2.1 LockFreeDMA_Queue . . . 429
18.83.3 Member Function Documentation . . . 429
18.83.3.1 first . . . 429
18.83.3.2 isEmpty . . . 429
18.83.3.3 isFull . . . 429
18.83.3.4 pop . . . 430
18.83.3.5 pushCopyOf . . . 430
18.84 Matrix4x4 Class Reference . . . 430
18.84.1 Detailed Description . . . 431
18.84.2 Constructor & Destructor Documentation . . . 431
18.84.2.1 Matrix4x4 . . . 431
18.84.3 Member Function Documentation . . . 431
18.84.3.1 concatenateXRotation . . . 431
18.84.3.2 concatenateXScale . . . 431
18.84.3.3 concatenateXTranslation . . . 431
18.84.3.4 concatenateYRotation . . . 432
18.84.3.5 concatenateYScale . . . 432
18.84.3.6 concatenateYTranslation . . . 432
18.84.3.7 concatenateZRotation . . . 432
18.84.3.8 concatenateZScale . . . 432
18.84.3.9 concatenateZTranslation . . . 433
18.84.3.10 getElement . . . 433
18.84.3.11 identity . . . 433
18.84.3.12 setElement . . . 433

18.84.3.13 setViewDistance . . . 433
18.85 MCUInstrumentation Class Reference . . . 434
18.85.1 Detailed Description . . . 434
18.85.2 Constructor & Destructor Documentation . . . 434
18.85.2.1 MCUInstrumentation . . . 434
18.85.2.2 ∼MCUInstrumentation . . . 434
18.85.3 Member Function Documentation . . . 435
18.85.3.1 getCCConsumed . . . 435
18.85.3.2 getCPUCycles . . . 435
18.85.3.3 getElapsedUS . . . 435
18.85.3.4 init . . . 435
18.85.3.5 setCCConsumed . . . 435
18.85.3.6 setMCUActive . . . 435
18.86 MoveAnimator< T > Class Template Reference . . . 436
18.86.1 Detailed Description . . . 437
18.86.2 Constructor & Destructor Documentation . . . 437
18.86.2.1 MoveAnimator . . . 437
18.86.2.2 ∼MoveAnimator . . . 437
18.86.3 Member Function Documentation . . . 437
18.86.3.1 cancelMoveAnimation . . . 437
18.86.3.2 getMoveAnimationDelay . . . 437
18.86.3.3 handleTickEvent . . . 438
18.86.3.4 isRunning . . . 438
18.86.3.5 setMoveAnimationDelay . . . 438
18.86.3.6 setMoveAnimationEndedAction . . . 438
18.86.3.7 startMoveAnimation . . . 438
18.87 MVPApplication Class Reference . . . 438
18.87.1 Detailed Description . . . 439
18.87.2 Constructor & Destructor Documentation . . . 439
18.87.2.1 MVPApplication . . . 439
18.87.2.2 ∼MVPApplication . . . 439
18.87.3 Member Function Documentation . . . 439
18.87.3.1 evaluatePendingScreenTransition . . . 439
18.87.3.2 handlePendingScreenTransition . . . 440
18.88 MVPHeap Class Reference . . . 440
18.88.1 Detailed Description . . . 440
18.88.2 Constructor & Destructor Documentation . . . 440
18.88.2.1 MVPHeap . . . 440
18.88.2.2 ∼MVPHeap . . . 441
18.89 NoDMA Class Reference . . . 441

18.89.1 Detailed Description . . . 441
18.89.2 Constructor & Destructor Documentation . . . 441
18.89.2.1 NoDMA . . . 441
18.89.3 Member Function Documentation . . . 441
18.89.3.1 flush . . . 441
18.89.3.2 getBlitCaps . . . 442
18.89.3.3 setupDataCopy . . . 442
18.89.3.4 setupDataFill . . . 442
18.89.3.5 signalDMAInterrupt . . . 442
18.90 NoTouchController Class Reference . . . 442
18.90.1 Detailed Description . . . 442
18.90.2 Constructor & Destructor Documentation . . . 443
18.90.2.1 ∼NoTouchController . . . 443
18.90.3 Member Function Documentation . . . 443
18.90.3.1 init . . . 443
18.90.3.2 sampleTouch . . . 443
18.91 NoTransition Class Reference . . . 443
18.91.1 Detailed Description . . . 443
18.91.2 Constructor & Destructor Documentation . . . 444
18.91.2.1 NoTransition . . . 444
18.91.2.2 ∼NoTransition . . . 444
18.91.3 Member Function Documentation . . . 444
18.91.3.1 handleTickEvent . . . 444
18.92 OSWrappers Class Reference . . . 444
18.92.1 Detailed Description . . . 444
18.92.2 Member Function Documentation . . . 444
18.92.2.1 giveFrameBufferSemaphore . . . 444
18.92.2.2 giveFrameBufferSemaphoreFromISR . . . 445
18.92.2.3 initialize . . . 445
18.92.2.4 signalVSync . . . 445
18.92.2.5 takeFrameBufferSemaphore . . . 445
18.92.2.6 tryTakeFrameBufferSemaphore . . . 445
18.92.2.7 waitForVSync . . . 445
18.93 Outline Class Reference . . . 445
18.93.1 Detailed Description . . . 446
18.93.2 Member Typedef Documentation . . . 446
18.93.2.1 OutlineFlags_t . . . 446
18.93.3 Constructor & Destructor Documentation . . . 446
18.93.3.1 Outline . . . 446
18.93.3.2 ∼Outline . . . 446

18.93.4 Member Function Documentation . . . 447
18.93.4.1 getCells . . . 447
18.93.4.2 getNumCells . . . 447
18.93.4.3 lineTo . . . 447
18.93.4.4 moveTo . . . 447
18.93.4.5 reset . . . 447
18.93.4.6 setMaxRenderY . . . 447
18.93.4.7 wasOutlineTooComplex . . . 447
18.94 PainterBW Class Reference . . . 448
18.94.1 Detailed Description . . . 448
18.94.2 Member Function Documentation . . . 448
18.94.2.1 bw . . . 448
18.94.2.2 renderNext . . . 449
18.94.2.3 setColor . . . 449
18.95 PainterRGB565 Class Reference . . . 449
18.95.1 Detailed Description . . . 450
18.95.2 Constructor & Destructor Documentation . . . 450
18.95.2.1 PainterRGB565 . . . 450
18.95.3 Member Function Documentation . . . 450
18.95.3.1 renderNext . . . 450
18.95.3.2 setColor . . . 450
18.96 PainterRGB565Bitmap Class Reference . . . 451
18.96.1 Detailed Description . . . 451
18.96.2 Constructor & Destructor Documentation . . . 452
18.96.2.1 PainterRGB565Bitmap . . . 452
18.96.3 Member Function Documentation . . . 453
18.96.3.1 renderInit . . . 453
18.96.3.2 renderNext . . . 453
18.96.3.3 setAlpha . . . 453
18.96.3.4 setBitmap . . . 453
18.97 PainterRGB888 Class Reference . . . 453
18.97.1 Detailed Description . . . 454
18.97.2 Constructor & Destructor Documentation . . . 454
18.97.2.1 PainterRGB888 . . . 454
18.97.3 Member Function Documentation . . . 454
18.97.3.1 renderNext . . . 454
18.97.3.2 setColor . . . 455
18.98 PainterRGB888Bitmap Class Reference . . . 455
18.98.1 Detailed Description . . . 456
18.98.2 Constructor & Destructor Documentation . . . 456

18.98.2.1 PainterRGB888Bitmap . . . 456
18.98.3 Member Function Documentation . . . 456
18.98.3.1 renderInit . . . 456
18.98.3.2 renderNext . . . 456
18.98.3.3 setAlpha . . . 457
18.98.3.4 setBitmap . . . 458
18.99 Pair< T1, T2 > Struct Template Reference . . . 458
18.99.1 Detailed Description . . . 458
18.99.2 Constructor & Destructor Documentation . . . 458
18.99.2.1 Pair . . . 458
18.99.2.2 Pair . . . 459
18.99.2.3 Pair . . . 460
18.100 Partition< ListOfTypes, NUMBER_OF_ELEMENTS > Class Template Reference . . . 460
18.100.1 Detailed Description . . . 461
18.100.2 Member Typedef Documentation . . . 461
18.100.2.1 SupportedTypesList . . . 461
18.100.3 Member Enumeration Documentation . . . 461
18.100.3.1 anonymous enum . . . 461
18.100.4 Constructor & Destructor Documentation . . . 461
18.100.4.1 Partition . . . 461
18.100.5 Member Function Documentation . . . 461
18.100.5.1 capacity . . . 461
18.100.5.2 element . . . 461
18.100.5.3 element . . . 462
18.100.5.4 element_size . . . 462
18.101 PixelDataWidget Class Reference . . . 462
18.101.1 Detailed Description . . . 463
18.101.2 Constructor & Destructor Documentation . . . 463
18.101.2.1 PixelDataWidget . . . 463
18.101.3 Member Function Documentation . . . 463
18.101.3.1 draw . . . 463
18.101.3.2 getSolidRect . . . 464
18.101.3.3 setAlpha . . . 464
18.101.3.4 setBitmapFormat . . . 464
18.101.3.5 setPixelData . . . 464
18.102 Point Struct Reference . . . 464
18.102.1 Member Function Documentation . . . 465
18.102.1.1 dist_sqr . . . 465
18.103 Point3D Struct Reference . . . 466
18.104 Point4 Class Reference . . . 466

18.104.1 Detailed Description . . . 466
18.104.2 Constructor & Destructor Documentation . . . 467
18.104.2.1 Point4 . . . 467
18.104.2.2 Point4 . . . 467
18.105 PreRenderable< T > Class Template Reference . . . 467
18.105.1 Detailed Description . . . 467
18.105.2 Constructor & Destructor Documentation . . . 468
18.105.2.1 PreRenderable . . . 468
18.105.3 Member Function Documentation . . . 468
18.105.3.1 draw . . . 468
18.105.3.2 isPreRendered . . . 468
18.105.3.3 preRender . . . 468
18.105.3.4 setupDrawChain . . . 468
18.106 Presenter Class Reference . . . 468
18.106.1 Detailed Description . . . 469
18.106.2 Constructor & Destructor Documentation . . . 469
18.106.2.1 ∼Presenter . . . 469
18.106.2.2 Presenter . . . 469
18.106.3 Member Function Documentation . . . 469
18.106.3.1 activate . . . 469
18.106.3.2 deactivate . . . 469
18.107 CWRUtil::Q10 Class Reference . . . 469
18.107.1 Detailed Description . . . 470
18.107.2 Constructor & Destructor Documentation . . . 470
18.107.2.1 Q10 . . . 470
18.107.2.2 Q10 . . . 470
18.107.3 Member Function Documentation . . . 470
18.107.3.1 operator int . . . 470
18.107.3.2 operator∗ . . . 470
18.107.3.3 operator+ . . . 471
18.107.3.4 operator- . . . 471
18.107.3.5 operator/ . . . 471
18.108 CWRUtil::Q15 Class Reference . . . 471
18.108.1 Detailed Description . . . 472
18.108.2 Constructor & Destructor Documentation . . . 472
18.108.2.1 Q15 . . . 472
18.108.3 Member Function Documentation . . . 472
18.108.3.1 operator int . . . 472
18.108.3.2 operator/ . . . 472
18.109 CWRUtil::Q5 Class Reference . . . 472

18.109.1 Detailed Description . . . 473
18.109.2 Constructor & Destructor Documentation . . . 473
18.109.2.1 Q5 . . . 473
18.109.2.2 Q5 . . . 473
18.109.3 Member Function Documentation . . . 474
18.109.3.1 operator int . . . 474
18.109.3.2 operator* . . . 474
18.109.3.3 operator* . . . 474
18.109.3.4 operator* . . . 474
18.109.3.5 operator+ . . . 475
18.109.3.6 operator- . . . 475
18.109.3.7 operator- . . . 475
18.109.3.8 operator/ . . . 475
18.109.3.9 operator/ . . . 475
18.109.3.10 0 . . . 476
18.110 Quadruple Class Reference . . . 476
18.110.1 Detailed Description . . . 477
18.110.2 Constructor & Destructor Documentation . . . 477
18.110.2.1 Quadruple . . . 477
18.110.2.2 Quadruple . . . 477
18.110.3 Member Function Documentation . . . 477
18.110.3.1 getElement . . . 477
18.110.3.2 getW . . . 477
18.110.3.3 getX . . . 478
18.110.3.4 getY . . . 478
18.110.3.5 getZ . . . 478
18.110.3.6 setElement . . . 478
18.110.3.7 setW . . . 478
18.110.3.8 setX . . . 478
18.110.3.9 setY . . . 478
18.110.3.10 setZ . . . 479
18.111 RadioButton Class Reference . . . 479
18.111.1 Detailed Description . . . 480
18.111.2 Constructor & Destructor Documentation . . . 480
18.111.2.1 RadioButton . . . 480
18.111.2.2 ~RadioButton . . . 480
18.111.3 Member Function Documentation . . . 480
18.111.3.1 draw . . . 480
18.111.3.2 getAlpha . . . 481
18.111.3.3 getCurrentlyDisplayedBitmap . . . 481

18.111.3.4 getDeselectionEnabled . . . 481
18.111.3.5 getSelected . . . 481
18.111.3.6 getSolidRect . . . 481
18.111.3.7 getType . . . 482
18.111.3.8 handleClickEvent . . . 482
18.111.3.9 setAlpha . . . 482
18.111.3.10 setBitmaps . . . 482
18.111.3.11 setDeselectedAction . . . 482
18.111.3.12 setDeselectionEnabled . . . 483
18.111.3.13 setSelected . . . 483
18.112 RadioButtonGroup< CAPACITY > Class Template Reference . . . 483
18.112.1 Detailed Description . . . 484
18.112.2 Constructor & Destructor Documentation . . . 484
18.112.2.1 RadioButtonGroup . . . 484
18.112.2.2 ~RadioButtonGroup . . . 484
18.112.3 Member Function Documentation . . . 485
18.112.3.1 add . . . 485
18.112.3.2 getDeselectionEnabled . . . 485
18.112.3.3 getRadioButton . . . 485
18.112.3.4 getSelectedRadioButton . . . 485
18.112.3.5 getSelectedRadioButtonIndex . . . 485
18.112.3.6 radioButtonClickedHandler . . . 485
18.112.3.7 radioButtonDeselectedHandler . . . 486
18.112.3.8 setDeselectionEnabled . . . 486
18.112.3.9 setRadioButtonDeselectedHandler . . . 486
18.112.3.10 setRadioButtonSelectedHandler . . . 486
18.112.3.11 setSelected . . . 486
18.113 Rasterizer Class Reference . . . 487
18.113.1 Detailed Description . . . 487
18.113.2 Member Enumeration Documentation . . . 488
18.113.2.1 anonymous enum . . . 488
18.113.2.2 anonymous enum . . . 488
18.113.2.3 FillingRule . . . 488
18.113.3 Constructor & Destructor Documentation . . . 488
18.113.3.1 Rasterizer . . . 488
18.113.4 Member Function Documentation . . . 488
18.113.4.1 calculateAlpha . . . 488
18.113.4.2 lineTo . . . 489
18.113.4.3 moveTo . . . 489
18.113.4.4 render . . . 489

18.113.4.5 reset . . . 489
18.113.4.6 setFillingRule . . . 489
18.113.4.7 setMaxRenderY . . . 490
18.114 Rect Class Reference . . . 490
18.114.1 Detailed Description . . . 491
18.114.2 Constructor & Destructor Documentation . . . 491
18.114.2.1 Rect . . . 491
18.114.2.2 Rect . . . 491
18.114.3 Member Function Documentation . . . 491
18.114.3.1 area . . . 491
18.114.3.2 bottom . . . 491
18.114.3.3 expandToFit . . . 491
18.114.3.4 includes . . . 492
18.114.3.5 intersect . . . 492
18.114.3.6 intersect . . . 492
18.114.3.7 isEmpty . . . 492
18.114.3.8 operator"!= . . . 492
18.114.3.9 operator& . . . 493
18.114.3.10 operator&= . . . 493
18.114.3.11 operator== . . . 493
18.114.3.12 right . . . 493
18.115 Renderer Class Reference . . . 493
18.115.1 Detailed Description . . . 494
18.115.2 Constructor & Destructor Documentation . . . 494
18.115.2.1 Renderer . . . 494
18.115.2.2 Renderer . . . 494
18.115.3 Member Function Documentation . . . 494
18.115.3.1 getRenderingBuffer . . . 494
18.115.3.2 render . . . 494
18.115.3.3 setRenderingBuffer . . . 495
18.116 RenderingBuffer Class Reference . . . 495
18.116.1 Detailed Description . . . 495
18.116.2 Constructor & Destructor Documentation . . . 496
18.116.2.1 ~RenderingBuffer . . . 496
18.116.2.2 RenderingBuffer . . . 496
18.116.2.3 RenderingBuffer . . . 496
18.116.3 Member Function Documentation . . . 496
18.116.3.1 attach . . . 496
18.116.3.2 getHeight . . . 497
18.116.3.3 getWidth . . . 497

18.116.3.4 getXAdjust . . . 497
18.116.3.5 inbox . . . 497
18.116.3.6 row . . . 497
18.116.3.7 row . . . 498
18.117 RGBA8 Struct Reference . . . 498
18.117.1 Detailed Description . . . 498
18.117.2 Constructor & Destructor Documentation . . . 499
18.117.2.1 RGBA8 . . . 499
18.117.2.2 RGBA8 . . . 499
18.117.2.3 RGBA8 . . . 499
18.118 ScalableImage Class Reference . . . 499
18.118.1 Detailed Description . . . 501
18.118.2 Member Enumeration Documentation . . . 501
18.118.2.1 ScalingAlgorithm . . . 501
18.118.3 Constructor & Destructor Documentation . . . 501
18.118.3.1 ScalableImage . . . 501
18.118.3.2 ~ScalableImage . . . 501
18.118.4 Member Function Documentation . . . 501
18.118.4.1 draw . . . 501
18.118.4.2 drawTriangle . . . 501
18.118.4.3 getAlpha . . . 502
18.118.4.4 getBitmap . . . 502
18.118.4.5 getScalingAlgorithm . . . 502
18.118.4.6 getSolidRect . . . 502
18.118.4.7 getType . . . 502
18.118.4.8 lookupRenderVariant . . . 503
18.118.4.9 setAlpha . . . 503
18.118.4.10 setBitmap . . . 503
18.118.4.11 setScalingAlgorithm . . . 503
18.119 Scanline Class Reference . . . 503
18.119.1 Detailed Description . . . 504
18.119.2 Constructor & Destructor Documentation . . . 505
18.119.2.1 ~Scanline . . . 505
18.119.2.2 Scanline . . . 505
18.119.3 Member Function Documentation . . . 505
18.119.3.1 addCell . . . 505
18.119.3.2 addSpan . . . 505
18.119.3.3 getNumSpans . . . 505
18.119.3.4 getY . . . 506
18.119.3.5 isReady . . . 506

18.119.3.6 reset . . . 506
18.119.3.7 resetSpans . . . 506
18.120 Screen Class Reference . . . 506
18.120.1 Detailed Description . . . 507
18.120.2 Constructor & Destructor Documentation . . . 508
18.120.2.1 Screen . . . 508
18.120.2.2 ~Screen . . . 508
18.120.3 Member Function Documentation . . . 508
18.120.3.1 add . . . 508
18.120.3.2 afterTransition . . . 508
18.120.3.3 bindTransition . . . 508
18.120.3.4 draw . . . 508
18.120.3.5 draw . . . 508
18.120.3.6 getRootContainer . . . 509
18.120.3.7 handleClickEvent . . . 509
18.120.3.8 handleDragEvent . . . 509
18.120.3.9 handleGestureEvent . . . 509
18.120.3.10 handleKeyEvent . . . 509
18.120.3.11 handleTickEvent . . . 509
18.120.3.12 SMOC . . . 510
18.120.3.13 remove . . . 511
18.120.3.14 setupScreen . . . 511
18.120.3.15 startSMOC . . . 511
18.120.3.16 tearDownScreen . . . 511
18.120.3.17 useSMOCDrawing . . . 511
18.120.3.18 usingSMOC . . . 511
18.121 ScrollableContainer Class Reference . . . 512
18.121.1 Detailed Description . . . 514
18.121.2 Constructor & Destructor Documentation . . . 515
18.121.2.1 ScrollableContainer . . . 515
18.121.2.2 ~ScrollableContainer . . . 515
18.121.3 Member Function Documentation . . . 515
18.121.3.1 add . . . 515
18.121.3.2 childGeometryChanged . . . 515
18.121.3.3 doScroll . . . 515
18.121.3.4 enableHorizontalScroll . . . 515
18.121.3.5 enableVerticalScroll . . . 516
18.121.3.6 getContainedArea . . . 516
18.121.3.7 getLastChild . . . 516
18.121.3.8 getType . . . 516

18.121.3.9 getXBorder . . . 516
18.121.3.10 getXScrollbar . . . 517
18.121.3.11 getYBorder . . . 517
18.121.3.12 getYScrollbar . . . 517
18.121.3.13 handleClickEvent . . . 517
18.121.3.14 handleDragEvent . . . 517
18.121.3.15 handleGestureEvent . . . 518
18.121.3.16 handleTickEvent . . . 518
18.121.3.17 invalidateScrollbars . . . 518
18.121.3.18 moveChildrenRelative . . . 518
18.121.3.19 reset . . . 518
18.121.3.20 setMaxVelocity . . . 518
18.121.3.21 setScrollbarPadding . . . 518
18.121.3.22 setScrollbarsAlpha . . . 519
18.121.3.23 setScrollbarsColor . . . 519
18.121.3.24 setScrollbarsVisible . . . 519
18.121.3.25 setScrollbarWidth . . . 519
18.121.3.26 setScrollThreshold . . . 519
18.122 SDLTouchController Class Reference . . . 519
18.122.1 Detailed Description . . . 520
18.122.2 Member Function Documentation . . . 520
18.122.2.1 init . . . 520
18.122.2.2 sampleTouch . . . 520
18.123 Shape< POINTS > Class Template Reference . . . 520
18.123.1 Detailed Description . . . 521
18.123.2 Constructor & Destructor Documentation . . . 521
18.123.2.1 ~Shape . . . 521
18.123.3 Member Function Documentation . . . 521
18.123.3.1 getCacheX . . . 521
18.123.3.2 getCacheY . . . 521
18.123.3.3 getCornerX . . . 522
18.123.3.4 getCornerY . . . 522
18.123.3.5 getNumPoints . . . 522
18.123.3.6 setCache . . . 522
18.123.3.7 setCorner . . . 523
18.124 AbstractShape::ShapePoint< T > Struct Template Reference . . . 523
18.124.1 Detailed Description . . . 523
18.125 Slider Class Reference . . . 523
18.125.1 Detailed Description . . . 525
18.125.2 Constructor & Destructor Documentation . . . 526

18.125.2.1 Slider . . . 526
18.125.3 Member Function Documentation . . . 526
18.125.3.1 getIndicatorPositionRangeSize . . . 526
18.125.3.2 getIndicatorRadius . . . 526
18.125.3.3 getType . . . 526
18.125.3.4 getValue . . . 526
18.125.3.5 getValueRangeSize . . . 526
18.125.3.6 handleClickEvent . . . 526
18.125.3.7 handleDragEvent . . . 526
18.125.3.8 positionToValue . . . 527
18.125.3.9 setBitmaps . . . 527
18.125.3.10 setBitmaps . . . 527
18.125.3.11 setNewValueCallback . . . 527
18.125.3.12 setStartValueCallback . . . 528
18.125.3.13 setStopValueCallback . . . 528
18.125.3.14 setupHorizontalSlider . . . 528
18.125.3.15 setupVerticalSlider . . . 529
18.125.3.16 setValue . . . 529
18.125.3.17 setValueRange . . . 529
18.125.3.18 setValueRange . . . 530
18.125.3.19 updateIndicatorPosition . . . 530
18.125.3.20 valueToPosition . . . 530
18.126 SlideTransition< templateDirection > Class Template Reference . . . 530
18.126.1 Detailed Description . . . 531
18.126.2 Constructor & Destructor Documentation . . . 531
18.126.2.1 SlideTransition . . . 531
18.126.2.2 ~SlideTransition . . . 531
18.126.3 Member Function Documentation . . . 532
18.126.3.1 handleTickEvent . . . 532
18.126.3.2 init . . . 532
18.126.3.3 initMoveDrawable . . . 532
18.126.3.4 tearDown . . . 532
18.126.3.5 tickMoveDrawable . . . 532
18.127 Snapper< T > Class Template Reference . . . 532
18.127.1 Detailed Description . . . 533
18.127.2 Constructor & Destructor Documentation . . . 533
18.127.2.1 Snapper . . . 533
18.127.2.2 ~Snapper . . . 533
18.127.3 Member Function Documentation . . . 533
18.127.3.1 handleClickEvent . . . 533

18.127.3.2 handleDragEvent . . . 534
18.127.3.3 setDragAction . . . 534
18.127.3.4 setSnappedAction . . . 534
18.127.3.5 setSnapPosition . . . 534
18.128 SnapshotWidget Class Reference . . . 534
18.128.1 Detailed Description . . . 535
18.128.2 Constructor & Destructor Documentation . . . 535
18.128.2.1 SnapshotWidget . . . 535
18.128.2.2 ~SnapshotWidget . . . 535
18.128.3 Member Function Documentation . . . 535
18.128.3.1 draw . . . 535
18.128.3.2 getAlpha . . . 536
18.128.3.3 getSolidRect . . . 536
18.128.3.4 getType . . . 536
18.128.3.5 makeSnapshot . . . 536
18.128.3.6 setAlpha . . . 536
18.129 LCD::StringVisuals Struct Reference . . . 536
18.129.1 Detailed Description . . . 537
18.129.2 Constructor & Destructor Documentation . . . 537
18.129.2.1 StringVisuals . . . 537
18.129.2.2 StringVisuals . . . 537
18.130 TextArea Class Reference . . . 538
18.130.1 Detailed Description . . . 539
18.130.2 Constructor & Destructor Documentation . . . 539
18.130.2.1 TextArea . . . 539
18.130.3 Member Function Documentation . . . 539
18.130.3.1 draw . . . 539
18.130.3.2 getAlpha . . . 540
18.130.3.3 getColor . . . 540
18.130.3.4 getIndentation . . . 540
18.130.3.5 getLinespacing . . . 540
18.130.3.6 getRotation . . . 540
18.130.3.7 getSolidRect . . . 540
18.130.3.8 getTextHeight . . . 541
18.130.3.9 getTextWidth . . . 541
18.130.3.10 getType . . . 541
18.130.3.11 getTypedText . . . 541
18.130.3.12 resizeToCurrentText . . . 541
18.130.3.13 setAlpha . . . 541
18.130.3.14 setBaselineY . . . 542

18.130.3.15 setColor . . . 543
18.130.3.16 setIndentation . . . 543
18.130.3.17 setLinespacing . . . 543
18.130.3.18 setRotation . . . 543
18.130.3.19 setTypedText . . . 543
18.130.3.20 setXBaselineY . . . 544
18.131 TextAreaWithOneWildcard Class Reference . . . 544
18.131.1 Detailed Description . . . 544
18.131.2 Constructor & Destructor Documentation . . . 545
18.131.2.1 TextAreaWithOneWildcard . . . 545
18.131.3 Member Function Documentation . . . 545
18.131.3.1 draw . . . 545
18.131.3.2 getTextHeight . . . 545
18.131.3.3 getTextWidth . . . 545
18.131.3.4 getType . . . 545
18.131.3.5 getWildcard . . . 546
18.131.3.6 setWildcard . . . 546
18.132 TextAreaWithTwoWildcards Class Reference . . . 546
18.132.1 Detailed Description . . . 547
18.132.2 Constructor & Destructor Documentation . . . 547
18.132.2.1 TextAreaWithTwoWildcards . . . 547
18.132.3 Member Function Documentation . . . 547
18.132.3.1 draw . . . 547
18.132.3.2 getTextHeight . . . 547
18.132.3.3 getTextWidth . . . 547
18.132.3.4 getType . . . 548
18.132.3.5 getWildcard1 . . . 548
18.132.3.6 getWildcard2 . . . 548
18.132.3.7 setWildcard1 . . . 548
18.132.3.8 setWildcard2 . . . 548
18.133 TextAreaWithWildcardBase Class Reference . . . 548
18.133.1 Detailed Description . . . 549
18.133.2 Constructor & Destructor Documentation . . . 549
18.133.2.1 TextAreaWithWildcardBase . . . 549
18.133.3 Member Function Documentation . . . 549
18.133.3.1 calculateTextHeight . . . 549
18.134 TextProvider Class Reference . . . 549
18.134.1 Detailed Description . . . 550
18.134.2 Constructor & Destructor Documentation . . . 550
18.134.2.1 TextProvider . . . 550

18.134.3 Member Function Documentation . . . 550
18.134.3.1 getNextChar . . . 550
18.134.3.2 initialize . . . 550
18.135 Texts Class Reference . . . 550
18.135.1 Detailed Description . . . 551
18.135.2 Member Function Documentation . . . 551
18.135.2.1 getLanguage . . . 551
18.135.2.2 getText . . . 551
18.135.2.3 setLanguage . . . 551
18.136 TextureMapper Class Reference . . . 551
18.136.1 Detailed Description . . . 555
18.136.2 Member Enumeration Documentation . . . 555
18.136.2.1 RenderingAlgorithm . . . 555
18.136.3 Constructor & Destructor Documentation . . . 555
18.136.3.1 ~TextureMapper . . . 555
18.136.4 Member Function Documentation . . . 555
18.136.4.1 applyTransformation . . . 555
18.136.4.2 draw . . . 556
18.136.4.3 drawTriangle . . . 557
18.136.4.4 getBitmap . . . 557
18.136.4.5 getBitmapPositionX . . . 557
18.136.4.6 getBitmapPositionY . . . 557
18.136.4.7 getBoundingRect . . . 558
18.136.4.8 getCameraDistance . . . 558
18.136.4.9 getCameraX . . . 558
18.136.4.10 getCameraY . . . 558
18.136.4.11 getOrigoX . . . 558
18.136.4.12 getOrigoY . . . 558
18.136.4.13 getOrigoZ . . . 558
18.136.4.14 getRenderingAlgorithm . . . 559
18.136.4.15 getScale . . . 559
18.136.4.16 getSolidRect . . . 559
18.136.4.17 getType . . . 559
18.136.4.18 getX0 . . . 559
18.136.4.19 getX1 . . . 559
18.136.4.20 getX2 . . . 560
18.136.4.21 getX3 . . . 560
18.136.4.22 getXAngle . . . 560
18.136.4.23 getY0 . . . 560
18.136.4.24 getY1 . . . 560

18.136.4.25 getY2 . . . 560
18.136.4.26 getY3 . . . 560
18.136.4.27 getYAngle . . . 561
18.136.4.28 getZ0 . . . 561
18.136.4.29 getZ1 . . . 561
18.136.4.30 getZ2 . . . 561
18.136.4.31 getZ3 . . . 561
18.136.4.32 getZAngle . . . 561
18.136.4.33 lookupRenderVariant . . . 561
18.136.4.34 setAlpha . . . 562
18.136.4.35 setBitmap . . . 563
18.136.4.36 setBitmapPosition . . . 563
18.136.4.37 setBitmapPosition . . . 563
18.136.4.38 setCamera . . . 563
18.136.4.39 setCameraDistance . . . 563
18.136.4.40 setOrigo . . . 563
18.136.4.41 setOrigo . . . 564
18.136.4.42 setRenderingAlgorithm . . . 564
18.136.4.43 setScale . . . 564
18.136.4.44 updateAngles . . . 564
18.136.4.45 updateXAngle . . . 564
18.136.4.46 updateYAngle . . . 564
18.136.4.47 updateZAngle . . . 565
18.137 TextureSurface Struct Reference . . . 565
18.138 ToggleButton Class Reference . . . 565
18.138.1 Detailed Description . . . 566
18.138.2 Constructor & Destructor Documentation . . . 566
18.138.2.1 ToggleButton . . . 566
18.138.3 Member Function Documentation . . . 566
18.138.3.1 forceState . . . 566
18.138.3.2 getState . . . 566
18.138.3.3 getType . . . 566
18.138.3.4 handleClickEvent . . . 566
18.138.3.5 setBitmaps . . . 567
18.139 TouchArea Class Reference . . . 567
18.139.1 Detailed Description . . . 568
18.139.2 Constructor & Destructor Documentation . . . 568
18.139.2.1 TouchArea . . . 568
18.139.3 Member Function Documentation . . . 568
18.139.3.1 draw . . . 568

18.139.3.2 getSolidRect . . . 568
18.139.3.3 getType . . . 568
18.139.3.4 handleClickEvent . . . 568
18.139.3.5 handleDragEvent . . . 569
18.139.3.6 setPressedAction . . . 569
18.140 TouchCalibration Class Reference . . . 569
18.140.1 Detailed Description . . . 569
18.140.2 Member Function Documentation . . . 569
18.140.2.1 setCalibrationMatrix . . . 569
18.140.2.2 translatePoint . . . 570
18.141 TouchController Class Reference . . . 571
18.141.1 Detailed Description . . . 571
18.141.2 Constructor & Destructor Documentation . . . 571
18.141.2.1 ~TouchController . . . 571
18.141.3 Member Function Documentation . . . 571
18.141.3.1 init . . . 571
18.141.3.2 sampleTouch . . . 571
18.142 Transition Class Reference . . . 572
18.142.1 Detailed Description . . . 572
18.142.2 Constructor & Destructor Documentation . . . 572
18.142.2.1 Transition . . . 572
18.142.2.2 ~Transition . . . 572
18.142.3 Member Function Documentation . . . 573
18.142.3.1 handleTickEvent . . . 573
18.142.3.2 init . . . 573
18.142.3.3 isDone . . . 573
18.142.3.4 setScreenContainer . . . 573
18.142.3.5 tearDown . . . 573
18.143 TypedText Class Reference . . . 573
18.143.1 Detailed Description . . . 574
18.143.2 Constructor & Destructor Documentation . . . 574
18.143.2.1 TypedText . . . 574
18.143.3 Member Function Documentation . . . 574
18.143.3.1 getAlignment . . . 574
18.143.3.2 getFont . . . 575
18.143.3.3 getId . . . 575
18.143.3.4 getText . . . 575
18.143.3.5 getTextDirection . . . 575
18.143.3.6 hasValidId . . . 575
18.143.3.7 registerTexts . . . 575

18.143.3.8 registerTypedTextDatabase . . . 575
18.144 TypedText::TypedTextData Struct Reference . . . 576
18.144.1 Detailed Description . . . 576
18.145 UIEventListener Class Reference . . . 576
18.145.1 Detailed Description . . . 577
18.145.2 Constructor & Destructor Documentation . . . 577
18.145.2.1 ~UIEventListener . . . 577
18.145.3 Member Function Documentation . . . 577
18.145.3.1 handleClickEvent . . . 577
18.145.3.2 handleDragEvent . . . 577
18.145.3.3 handleGestureEvent . . . 577
18.145.3.4 handleKeyEvent . . . 577
18.145.3.5 handlePendingScreenTransition . . . 577
18.145.3.6 handleTickEvent . . . 578
18.146 Unicode Class Reference . . . 578
18.146.1 Member Typedef Documentation . . . 579
18.146.1.1 UnicodeChar . . . 579
18.146.2 Member Function Documentation . . . 579
18.146.2.1 atoi . . . 579
18.146.2.2 itoa . . . 579
18.146.2.3 snprintf . . . 579
18.146.2.4 snprintf . . . 580
18.146.2.5 snprintfFloat . . . 580
18.146.2.6 snprintfFloat . . . 581
18.146.2.7 snprintfFloats . . . 581
18.146.2.8 snprintfFloats . . . 582
18.146.2.9 strlen . . . 582
18.146.2.10 strlen . . . 583
18.146.2.11 strncmp . . . 584
18.146.2.12 strncmp_ignore_white_spaces . . . 584
18.146.2.13 strncpy . . . 584
18.146.2.14 strncpy . . . 585
18.146.2.15 toa . . . 585
18.147 Vector< T, capacity > Class Template Reference . . . 585
18.147.1 Detailed Description . . . 586
18.147.2 Constructor & Destructor Documentation . . . 586
18.147.2.1 Vector . . . 586
18.147.3 Member Function Documentation . . . 586
18.147.3.1 add . . . 586
18.147.3.2 clear . . . 586

18.147.3.3 contains . . . 587
18.147.3.4 isEmpty . . . 588
18.147.3.5 maxCapacity . . . 588
18.147.3.6 operator[] . . . 588
18.147.3.7 operator[] . . . 588
18.147.3.8 remove . . . 588
18.147.3.9 removeAt . . . 589
18.147.3.10 reverse . . . 589
18.147.3.11 size . . . 589
18.148 Vector4 Class Reference . . . 589
18.148.1 Detailed Description . . . 589
18.148.2 Constructor & Destructor Documentation . . . 590
18.148.2.1 Vector4 . . . 590
18.148.2.2 Vector4 . . . 590
18.148.3 Member Function Documentation . . . 590
18.148.3.1 crossProduct . . . 590
18.149 View< T > Class Template Reference . . . 590
18.149.1 Detailed Description . . . 590
18.149.2 Constructor & Destructor Documentation . . . 591
18.149.2.1 View . . . 591
18.149.3 Member Function Documentation . . . 591
18.149.3.1 bind . . . 591
18.150 Widget Class Reference . . . 591
18.150.1 Detailed Description . . . 592
18.150.2 Constructor & Destructor Documentation . . . 592
18.150.2.1 Widget . . . 592
18.150.2.2 ~Widget . . . 592
18.150.3 Member Function Documentation . . . 592
18.150.3.1 getLastChild . . . 592
18.150.3.2 getType . . . 592
18.151 ZoomAnimationImage Class Reference . . . 593
18.151.1 Detailed Description . . . 595
18.151.2 Member Enumeration Documentation . . . 596
18.151.2.1 States . . . 596
18.151.3 Constructor & Destructor Documentation . . . 596
18.151.3.1 ZoomAnimationImage . . . 596
18.151.3.2 ~ZoomAnimationImage . . . 596
18.151.4 Member Function Documentation . . . 596
18.151.4.1 getAlpha . . . 596
18.151.4.2 getAnimationDelay . . . 596

18.151.4.3 getLargeBitmap . . . 596
18.151.4.4 getScalingMode . . . 597
18.151.4.5 getSmallBitmap . . . 597
18.151.4.6 getType . . . 597
18.151.4.7 handleTickEvent . . . 597
18.151.4.8 isRunning . . . 597
18.151.4.9 setAlpha . . . 597
18.151.4.10 setAnimationDelay . . . 597
18.151.4.11 setAnimationEndedCallback . . . 598
18.151.4.12 setBitmaps . . . 598
18.151.4.13 setCurrentState . . . 598
18.151.4.14 setDimension . . . 598
18.151.4.15 setHeight . . . 598
18.151.4.16 setPosition . . . 599
18.151.4.17 setScalingMode . . . 599
18.151.4.18 setWidth . . . 599
18.151.4.19 startZoomAndMoveAnimation . . . 599
18.151.4.20 startZoomAnimation . . . 600
18.151.4.21 updateRenderingMethod . . . 600
18.151.4.22 updateZoomAnimationDeltaXY . . . 600

Index **601**

Chapter 1

TouchGFX Manual

Introduction

Welcome to the TouchGFX Manual! The sections contained within embody the complete documentation of the TouchGFX graphics framework and the toolchain that supports it. If you are new to TouchGFX we recommended that you, as a start, read the following topics to get a solid understanding of the framework and how to get from source code to running application.

1. What is TouchGFX and the philospohy behind it? (Further down this section).
2. The TouchGFX Distribution discusses the contents, structure and licensing model of the TouchGFX distribution.
3. Development Environment shows you how to setup a TouchGFX development environment.
4. PC Simulator teaches you to build and execute applications for the PC simulator, including a Quickstart for the impatient.
5. Examples contains a list of ready to build examples to try out either on the PC simulator or on target hardware.
6. Creating a New Application shows you how to create a new application from scratch using gcc or Microsoft Visual Studio.
7. Application Development enlightens you in the ways of TouchGFX application development now that you have read the above topics. Keep the `API documentation` handy.
8. Target Hardware now that your finished application is ready to be deployed.

Several *advanced* topics go beyond standard application development and deployment.

1. Target Configurations contains details on how *board-* and *OS-packages* are used in conjunction to construct build configurations for hardware platforms.
2. RT Operating System (touchgfx/os/) is important for those interested in running a different OS than what TouchGFX provides (FreeRTOS).
3. Performance Measurement Add on Cortex-M MCU instrumentation to your project in an effort to read out cpu cycles, elapsed time, and MCU load.
4. Text Format shows you how to enable Hardware-accelerated text rendering on ST devices. This setting is used by TouchGFX applications where applicable.
5. Advanced contains topics on the fundamental workings of TouchGFX such as TFT Controller timings and how to do performance measurements by reading out CPU cycles from Cortex-M CPUs.

6. Porting Guide reveals the innards of the TouchGFX HAL and provides you with a complete guide to porting TouchGFX to platforms other than the ones immediately supported.

The remaining parts of this manual go into detail about the contents of a TouchGFX distribution and how to develop, build, execute and download TouchGFX applications. Advanced topics are available for those who require a different configuration than what is immediately provided by the standard TouchGFX distribution.

What is TouchGFX?

TouchGFX is a framework written in C++ that enables you to create modern graphical user interfaces on embedded systems with limited resources. That means a smartphone-like look and feel on hardware an order of magnitude less powerful than smartphones. TouchGFX is unique in its ability to provide the same user interface experience people have come to expect today, in applications that for reasons of unit cost, power consumption or otherwise cannot use the hardware platforms normally associated with that type of UI. TouchGFX achieves its smart-phone-like look and feel by making use of cutting-edge graphics invalidation routines (*Occlusion Culling*) and by applying clever rendering algorithms and drawing mechanisms.

To give you an easy-to-understand picture of the structure of the framework, here is a teaser diagram depicting the different layers of TouchGFX: Core, Application, OS- and Hardware Abstraction Layers and how these fit together. We will revisit the diagram later in this manual and leave you with an initial impression for now.

TouchGFX was designed to make application development as flexible and effortless as possible for application programmers by offering, amongst other things, the following:

1. An Object-Oriented and extendable API
2. A tool chain for generating application assets (bitmaps and fonts) and building PC Simulator applications
3. Said PC simulator for prototyping and easy debugging
4. A Built-in (but optional) scheme for efficient memory allocation
5. An architecture that cleanly separates the responsibilities of the GUI from the business logic of the application

TouchGFX is able to run on a multitude of hardware platforms. A natural fit is any Cortex-M based platform, with the exact parameters, of course, depending on display resolution, memory latency and bandwidth, etc. Example platforms are:

MCU	Clock speed	Display
Cortex-M0	24-48MHz	320x240 QVGA
Cortex-M3	48-64MHz	320x240 QVGA
Cortex-M3/M4	96-120MHz	480x272 WQVGA
Cortex-M3/M4	120-204MHz	800x480 WVGA
Cortex-M3/M4	180-204MHz	1024x768 XGA
Cortex-M7	200MHz	1024x768 XGA

TouchGFX can run on systems where the TFT controller is integrated in the MCU with frame buffers in external memory, or on systems that use an LCD with integrated frame buffer and controller. In some cases a Cortex-M0 based solution could also be sufficient. The framework comes with built-in support for a number of evaluation boards and MCUs. For a complete list see Supported Hardware.

The resource requirements of TouchGFX depends, of course, on the size and complexity of the specific application. Below are figures for the usage of the framework itself as well as typical requirements for UI implementations. The lower bound represent user interfaces like the simple examples included in the framework distribution. The higher bound represent very complex applications with ∼200 different screens.

Int. RAM
10-20 KB (framework+stack) 1-15 KB (widgets)
Int. ROM
20 KB (framework) 1-40 KB (screen definitions, UI logic)
Ext. RAM
1-3 frame buffers in external RAM / integrated on display
Ext. Flash
1-8 MB for graphics data, fonts, text and translations.
Operating System
OS independent. Can run with (one task, two semaphores) or without (not recommended).

Background and Philosophy

The development of TouchGFX started in 2009 where existing solutions were found to be either unsuitable for the types of hardware we were concerned with, or took an approach to user interface design and implementation we felt were not quite up to date. While other solutions are certainly capable of delivering great-looking UIs, they simply cannot run on cost-efficient, low-power hardware. Other alternatives are comparable to TouchGFX in resource requirements but with concepts like resizable and minimizable windows, focus, cursors and standard look and feel widgets seem more targeted towards creating interfaces reminiscent of desktop PCs rather than smartphones.

A key design decision of TouchGFX is to prefer prerasterized (bitmap) graphics as opposed to arithmetical (vector) graphics. Apart from text rendering, almost everything is done with bitmaps. This means that the DMA can be used extensively when rendering, thereby significantly reducing the MCU load. We can also heavily optimize occlusion culling by using offline-generated data from the bitmap conversion tool. In essence, this is what enables TouchGFX to run well on the target platform range described in *"What is TouchGFX?"*.

Another important design decision was to not include standard look and feel widgets. That means there is no built-in definition of what e.g. a button should look like. The graphics associated with the button must be supplied as part of the application thus giving it a customized look and feel, tailored to the specific application and the client's brand and style guides. By taking this approach, leaving out standard graphics, the memory footprint is reduced and better tailors the framework to be customizable to a very high degree.

Last, but not least, the framework was designed with extreme focus on memory usage and MCU load. Very rarely does an application consist of just a user interface, so there must be plenty of resources available for the system to do "real work" as well. The footprint of TouchGFX is very low and it is capable of rendering even very complex full-screen transition effects with an MCU load typically less than 15%.

Chapter 2

The TouchGFX Distribution

2.1 Updating your TouchGFX installation

TouchGFX uses a semantic versioning scheme. This means that an update *within* same *major* release can be done by rebuilding your applications against the new distribution. A bump to the MAJOR verison number could possibly involve changes to your application. Given a version number `MAJOR.MINOR.PATCH`, the semantic versioning scheme is defined as follows:

1. Increment MAJOR version when introducing changes that make incompatible API changes
2. Increment MINOR version when adding functionality in a backwards-compatible manner, and
3. Increment PATCH version when making backwards-compatible bug fixes.

2.2 Distribution types

Three different types of TouchGFX distributions exist, each with their own license scheme. Each of these schemes affect the contents of the distribution in a number of ways. See Precompiled libraries (touchgfx/lib/) for more information on the various libraries available and the supported platforms. The table below goes through each of these versions.

Type	Description
Evalulation	The *Evaluation* copy of TouchGFX uses pre-built libraries for both the core API and the HAL. IAR project configurations are postfixed with `-eval`.
Standard	*Standard* refers to a version that uses pre-built libraries for the core API and supplies source code for the HAL. IAR project configurations are postfixed with `-std`.
Full	*Full* refers to a version that includes full source for every part of TouchGFX, including tools, core API and HAL. IAR project configurations for this version do not have a postfix.

2.3 Main Component Model

A TouchGFX application follows the layered model that can be seen in the figure below. The TouchGFX distribution provides, depending on the distribution, either pre-build libraries or source code for both the TouchGFX Core and

OSAL/HAL layers. For the application layer only demos and examples are provided. This is the layer that should be implemented for an application specific behaviour.

The HAL layer holds the porting for specific boards. The distribution includes standard drivers for MCU, DMA, and Touch, as well as full HAL layers (Board Packages) for a variety of evaluation boards. If your specific target board uses any of the supported components the drivers should be directly reusable in the needed customized port.

The distribution also supplies components for the OSAL layer in form of OS wrappers for selected operating systems. If you use one of the supported operating systems you should be able to simply reuse the supplied OS wrapper in your application and just adjust the OS configuration file.

Figure 2.1: Model of the main components of a TouchGFX application.

2.4 Supported Hardware

To get TouchGFX running on your custom board you need to port TouchGFX' hardware abstraction layer (HAL) to it. The TouchGFX distribution, however, includes a driver package containing helpful components to use in your porting process. There are several benefits to choosing a hardware platform that includes component that are directly supported, or similar to the supported hardware setups. One of these benefits is that the driver package helps the process of porting your application to the specific board, making the process much faster. It is, of course, possible to make a custom port for other microcontrollers as well.

Below is a list of the microcontrollers currently supported.

Manufacturer	MCU
NXP	LPC1788
NXP	LPC1853
NXP	LPC1857
NXP	LPC4088
NXP	LPC4353
NXP	LPC4357
ST	STM32F429
ST	STM32F439

ST	STM32F469
Freescale	K70

The TouchGFX distribution also includes complete board packages for a number of official demo boards. With one of these in hand, you can have your first TouchGFX running on target within minutes, since TouchGFX comes with predefined projects that are ready to be compiled and downloaded to any of the boards.

Board	MCU	Compiler
FDI uEZGUI-1788-70WVT	NXP LPC1788	IAR
TouchGFX demo board 4.3"	NXP LPC4353, NXP LPC4350	IAR, Keil, GCC
TouchGFX demo board 5.7"	NXP LPC4353	IAR, Keil, GCC
TouchGFX demo board 7.0"	NXP LPC4353	IAR, Keil, GCC
ST STM32429I-EVAL1	ST STM32f429	IAR, Keil, GCC
ST STM324x9I-EVAL-5.7"	ST STM32f4x9	IAR, Keil, GCC
ST STM32F429I-DISCO	ST STM32f429	IAR, Keil, GCC
ST STM32F469I-EVAL	ST STM32F469	IAR, Keil, GCC
ST STM32F469I-Discovery	ST STM32F469	IAR, Keil, GCC
Embedded Artists LPC4357DevKit	NXP LPC4357	IAR, Keil, GCC
Embedded Artists LPC4088DisplayModule	NXP LPC4088	IAR, Keil, GCC

2.5 TouchGFX File Structure

The TouchGFX distribution includes a set of different files. The following sections describes the main groups of the TouchGFX file structure.

The listing below depicts the overall folder structure of a TouchGFX distribution.

```
+-- app
|   +-- demo
|   +-- example
|   \-- template
|
+-- doc
+-- touchgfx
|   +-- board
|   +-- config
|   +-- framework
|   +-- lib
|   \-- os
|
\-- changelog.txt
\-- known_issues.txt
```

2.5.1 Applications (app/)

The `app` folder contains a number of TouchGFX applications.

- `demo` contains comprehensive TouchGFX applications that give insight into creating and structuring more complex graphical interfaces. For more detail see Demos.
- `example` contains example applications that showcases the widget library provided by the framework. For more detail see Examples.
- `template` contains a template application `EmptyApplication` that provides an empty shell for application developers to quickly get started. As the application places no restrictions on screen dimensions or otherwise, it can be targeted at all available hardware platforms in TouchGFX.

```
+-- demo
```

```
|   +-- touchgfx_demo2014_240x320
|   +-- touchgfx_demo2014_480x272
|   \-- home_automation
|       \-- home_control_demo
|
|-- example
|   +-- animation_example
|   +-- button_example
|   +-- custom_transition_example
|   +-- ...
|   \-- transition_example
|
\-- template
    \-- EmptyApplication
```

2.5.2 Documentation (doc/)

Contains this document.

2.5.3 TouchGFX (touchgfx/)

The `touchgfx` folder contains all the source code and binaries you will need in the development of new and existing touchgfx applications for both simulator and target hardware platforms.

The folder `config` contains files used by the TouchGFX build environment and will not be covered here in detail.

```
+-- touchgfx
|   +-- board
|   +-- config
|   +-- framework
:   +-- lib
:   \-- os
```

2.5.3.1 Board Packages (touchgfx/board/)

The `board` folder contains what is called *board packages* that contain any source code required to initialize hardware, OS and the TouchGFX framework itself for a specific target platform supported by TouchGFX. These board packages are provided for a number of evaluation boards that allow users to concentrate on developing applications.

The file `BoardConfiguration.cpp` file of the TouchGFX provided board packages tie hardware and software configuration together allowing it to run TouchGFX applications on the target board in question. `bsp` (Board support) contains source code for anything that is not delivered by the board vendor in question. The `vendor` folder typically contains vendor provided source code related to board bringup. The following listing shows the folder structure for the `Mjolner` board package located under `board/Mjolner/`. As we support the Mjolner line of boards in more than one configuration (4.3" and 7.0"), more than one `BoardConfiguration.cpp` is provided.

```
+-- TouchGFXDemoBoardRev1.1-4353
|   +-- flashloader
|   +-- include
|   |   +-- bsp
|   |   \-- vendor
|   |
|   +-- source
|   |   +-- bsp
|   |   +-- vendor
|   |   +-- BoardConfiguration_4_3.cpp
:   :   +-- BoardConfiguration_7_0.cpp
:   :   \-- GPIO.cpp
```

Board configurations define a very simple interface that can be used to expose methods for initializing hardware and software. Any method of initialization could be used, but a generic interface allow applications in TouchGFX to share more common code across target platforms that all use the same Operating System.

The *GPIO* class (touchgfx::GPIO) defines an interface used internally for debugging purposes. To that end, the class defines a list of helpful types that are signaled by the core framework. By assigning a GPIO on a target hardware platform to each of these types we can measure timings using blinking LEDs or an oscilloscope.

```
typedef enum
{
  VSYNC_FREQ,
  RENDER_TIME,
  FRAME_RATE,
  MCU_ACTIVE
} GPIO_ID;
```

See Target Configurations for information on how these board packages are used in target build configurations.

2.5.3.2 Framework (touchgfx/framework/)

The `framework` folder contains files required to create TouchGFX applications. In the listing below, the two components of interest are the `platform` folder, containing platform specific interfaces and `touchgfx` containing the core API. The contents of `platform` an advanced topic and can be studied in Platform Abstraction and Porting Guide. The `tools` folder contains tools for handling assets such as images, fonts, and texts. Read more about these in Assets and Tools.

```
+-- framework
|   +-- include
|   |   +-- common
|   |   +-- mvp
|   |   +-- platform
|   |   \-- touchgfx
|   |
|   \-- tools
|       +-- fontconvert
:       +-- imageconvert
:       \-- textconvert
```

2.5.3.3 Precompiled libraries (touchgfx/lib/)

The `lib` folder contains precompiled core libraries (`core`), libraries for the evaluation boards (`board`), TouchGFX application libraries for Windows (`win`) and Linux (`linux`) and some vendor libraries used in the TouchgGFX simulator (`sdl`).

```
+-- lib
|   +-- board
|   +-- core
|   +-- linux
|   +-- sdl
|   \-- win
|
\-- os
```

The complete list of permutations of platforms and compilers for `core`:

Platform	Folder	Compiler
Windows	lib/win	gcc and Visual Studio 2010/2012/2013
Linux	lib/linux	gcc
ARM Cortex-M3	lib/core/cortex_m3/IAR	IAR
ARM Cortex-M3	lib/core/cortex_m3/Keil	Keil
ARM Cortex-M4F	lib/core/cortex_m4F/IAR	IAR

ARM Cortex-M4F	lib/core/cortex_m4F/Keil	Keil
ARM Cortex-M4F	lib/core/cortex_m4F/gcc	gcc

2.5.3.4 RT Operating System (touchgfx/os/)

TouchGFX requires, as a minimum, one thread for the GUI and two semaphores that govern access to resources central to the framework. TouchGFX can run with almost any OS by implementing the oswrapper class appropriately. The folder `os` contains a single *OS package* (FreeRTOS 7.6.0) that is used by all target board configurations in TouchGFX. The structure of The FreeRTOS OS package is as in the following listing, where `License/` and `Source/` are vendor supplied folders.

```
\-- os
    +-- FreeRTOS7.6.0
        +-- License
        +-- Source
        \-- OSWrappers.cpp
```

`OSWrappers.cpp` is the implementation of an OS Abstraction Layer (OSAL) required for every OS package. The layer specifies methods for wrapping methods related to controlling access to vital framework resources. While the documentation for `OSWrappers` (touchgfx::OSWrappers) specifies what each method should be doing, an example implementation can be found in the FreeRTOS7.6.0 package. The OSAL allows users to create new packages for, and use with TouchGFX, any OS that offers thread and semaphore handles.

To use an OS package in your project, include the source for the OS as well as your new `OSWrappers.cpp` for compilation in your project. If you are running an evaluation copy of TouchGFX, FreeRTOS7.6.0 will be included in the provided libraries.

See Target Configurations for details on how to use an OS package in your project.

Chapter 3

Development Environment

The easiest way to setup a TouchGFX development environment is to use one of the few concrete methods supplied by TouchGFX, as explained below. The guidelines in *Custom* detail the tool-chain required to setup a custom environment, on Linux, for instance.

TouchGFX MinGW	Visual Studio	Custom Environment
TouchGFX ships with a simple MSYS/MinGW shell with the required tools and compilers installed and configured for compiling on Windows. Extract the contents from the self-extracting zip to the root of your `C:` drive, and you are ready to start building and running applications (See Build and Execute).	TouchGFX supplies solutions and libraries for Visual Studio 2010, 2012, and 2013. If you wish to use either version to develop TouchGFX applications see Visual Studio and Visual Studio Project Creation for further details.	If you use Linux, or for any reason wish to setup your own development environment, the tools you need to install are: 1. `gcc` 4.9.1+ 2. `make` 3.8.1+ 3. `ruby` 1.9.x 4. `roo` (Excel interface Ruby gem) 5. GCC compiler for ARM processors (See GCC Building applications with GNU GCC compiler for ARM) 4.9.3

Warning

If using the TouchGFX MinGW development environment for Windows, avoid using spaces in any of the foldernames in your project as this may caused undefined MinGW behavior. This warning is also issued by the MinGW shell upon startup.

Note

Linux - libraries only provided for 64-bit systems.

Figure 3.1: The TouchGFX MinGW Shell

Chapter 4

Build and Execute

This section describes how you build and run your code on both the PC Simulator and on target hardware. Quickstart lets you dive right in, while the remaining sections go into more detail about building for the PC simulator and target hardware.

4.1 Quickstart

If you are anxious to dive right in without further ado and experience TouchGFX in action, navigate to `app/example/button_example` from your TouchGFX installation folder and execute the two commands below. This will build and run the button example application in the TouchGFX simulator using *gcc*. A working TouchGFX development environment (See Development Environment) is a pre-requisite.

```
$ make -f simulator/gcc/Makefile
$ ./build/bin/simulator.exe
```

You can also build and run the example code in Visual Studio 2010/2012/2013. To do this, build the solution file located in `app/example/button_example/simulator/msvs/`. Other pre-made applications can be found in the `app/` directory. See Examples and Demos. If you are using Visual Studio 2010 you must update it with Service Pack 1 (SP1) for Visual Studio 2010. It is available through Windows Update or directly from `Microsoft's webpage`

Deploying an application to a hardware platform is discussed in detail in Target Hardware which covers downloading to all the supported evaluation boards (see Supported Hardware).

4.2 Make

TouchGFX makes heavy use of Make to convert application assets to code, build simulator applications and applications for target hardware. The Makefiles of TouchGFX provide four simple commands:

1. `clean` - Cleans application build and generated assets.
2. `all` - Generates assets and builds application (Default target)
3. `assets` - Generates assets
4. `flash` - Flash application to target hardware (Using ST-Link CLI)

Here's a few useful make commands:

```
# Clean and generate only assets
$ make -f simulator/gcc/Makefile clean assets
```

```
# Clean, generate assets and build simulator application
$ make -f simulator/gcc/Makefile clean all
```

```
# Clean, generate assets and build application for specific target
$ make -f target/ST/STM324x9I-CUBE/gcc/Makefile clean all
```

```
# flash an application with ST-Link
$ make -f target/ST/STM32429I-EVAL1/gcc/Makefile flash
```

The `flash` command can only be used on a windows platform and if ST-Link is properly installed.

```
$ make -f target/ST/STM32469I-EVAL/gcc/Makefile flash
Linking build/bin/target.elf
Producing additional output formats...
  target.hex   - Combined internal+external hex
  intflash.elf - Internal flash, elf debug
  intflash.hex - Internal flash, hex
  extflash.bin - External flash, binary
STM32 ST-LINK CLI v2.3.0
STM32 ST-LINK Command Line Interface

ST-LINK SN : 066CFF495351885087243857
ST-LINK Firmware version : V2J24M11
Connected via SWD.
SWD Frequency = 4000K.
Target voltage = 3.2 V.
Connection mode : Normal.
Device ID:0x434
Device flash Size : 2048 Kbytes
Device family :STM32F469x/F479x

Loading file...
Flash Programming:
  File : build/bin/intflash.hex
  Address : 0x08000000
Memory programming...
                                                  100%
Memory programmed in 5s and 0ms.
Programming Complete.

MCU Reset.

STM32 ST-LINK CLI v2.3.0
STM32 ST-LINK Command Line Interface
```

Figure 4.1: Flash application using Make

The flash command might fail with the following error, due to the external flash being mounted as an USB device in windows.

```
  Device Name          : MT25QL512A_STM32756G-EVAL
  Device Start Address : 0x90000000
  Device Size          : 0x04000000
  Device Type          : Nor Flash
Memory programming...
                                                    7%
STLink USB communication error
Error occured during erase operation!
Programming Complete.

STLink USB communication error
target/ST/STM32756G-EVAL/gcc/Makefile:210: recipe for target '_flash_' failed
make[1]: *** [_flash_] Error 2
eval/app/demo/touchgfx_demo2015/target/ST/STM32756G-EVAL/gcc/Makefile:66: reci
make: *** [flash] Error 2
```

Figure 4.2: ST-Link CLI communication error

Disabling the USB device driver for your particular board in Windows Device Manager should fix the issue:

Figure 4.3: Flash using Make command

4.3 PC Simulator

TouchGFX comes with a PC-based simulator that can run on both Windows and Linux platforms. Please note that only 64-bit Linux is supported. TouchGFX is platform independent, so when running the simulator, all code being executed, except for the code in specific HAL implementations, is exactly the same as when running on a target platform making application development seamless from simulator to target hardware.

The simulator provides an easy way of getting started without having a target hardware platform available. In addition, it provides a build-chain which automatically generates the source code representation of images, fonts, texts and translations to be included in the application. While using the simulator is optional, you will get the code generation of assets done automatically instead of manually having to execute the conversion tools, so we recommend keeping the simulator around.

Since the TouchGFX build utilize dependency-checking mechanism to examine the assets folders, adding a new image, modifying an image or changes to the text spreadsheet will automatically result in the appropriate source code being generated next time you compile.

When creating your own applications, as long as you base your file and directory structure on that of the template_application, you should easily be able to generate assets and compile everything by typing `make -f simulator/gcc/Makefile` in the application root directory or by using the Visual Studio solution located in `simulator/msvs`.

Note: If you do not wish to use our toolchain for compiling your source code, you can use the command

```
$ make -f simulator/gcc/Makefile assets
```

to generate asset output and not compile anything. You could then use your favorite IDE to compile and link the sources.

4.3.1 MinGW / GCC

To build and run a TouchGFX application in the MinGW shell do the following (using the button_example as example).

Launch the development environment shell, navigate to the place where TouchGFX is located and cd into the `app/example/button_example` directory. To compile, simply type

```
$ make -f simulator/gcc/Makefile
```

This process will generate the assets used in the application, and produce a binary of the simulator placed in `build/bin`. To execute the simulator, type

```
$ ./build/bin/simulator.out
```

4.3.2 Visual Studio

Visual Studio 2010/2012/2013 is an alternative to using make. All supplied applications (in `app/<type>/<app_name>/simulator/msvs/`) comes with a complete Visual Studio solution that is ready to compile and run.

Assets are automatically generated when compiling in Visual Studio.

Note: If you are trying to compile a Visual Studio 2012 project in Visual Studio 2010 you will get a compile error stating that "Specified platform toolset (v110) is not installed or invalid". To fix this right-click on the project in the Soulution Explorer and select "properties". Select Configuration Properties/General. Here you can change the Platform Toolset from v110 to v100. Then rebuild.

Note: Also note, that the supplied a Visual Studio project files are all distributed as Visual Studio 2010 projects. If you are using a newer version of Visual Studio, either choose to upgrade the project when you load it the first time (Visual Studio will tell you that "one or more projects use an earlier version of the Visual C++ compiler and libraries"), or select the project in the Solution Explorer and from the menu, select "PROJECT", "Update VC++ projects...". Remember to rebuild after updating the project.

4.4 Keil Compiler Cross-Version Compatibility

Keil projects for Cortex-M7 based targets (STM32F746G-DISCO and STM32756G-EVAL) are based on Keil uVision 5. Project files based on the new .uvprojx format are not compatible with Keil uVision 4. When one, illegally, attempts to open an .uvprojx Keil 4, the following error occurs:

Figure 4.4: Opening a Keil 5 project with Keil 4

Opening a keil 4 project (.uvproj) with Keil 5 is allowed but produces the following prompt to migrate to the new format or to install legacy support.

Figure 4.5: Opening a Keil 4 project with Keil 5

4.5 Target Hardware

TouchGFX currently supports three compilers for microcontroller targets:

1. IAR C/C++ Compiler for ARM
2. Keil Compiler for ARM (uVision 4 and 5 IDE)
3. GNU GCC Compiler for ARM

For a list of supported hardware and compilers see Supported Hardware.

The section GCC Building applications with GNU GCC compiler for ARM shows you how to build applications using GNU GCC compiler for ARM and also goes into detail about how to program each board. In general, for ST based boards, the ST-Link flashing utility is always used.

1. TouchGFX demo board 4.3" (NXP LPC4350, No internal flash) - See Programming TouchGFX demo board 4.3" LPC4350 when Compiling with GCC
2. TouchGFX demo board 4.3"/5.7"/7.0" (NXP LPC4353, Internal+External flash) - See Programming TouchGFX demo board 4.3"/7.0" LPC4353 when Compiling with GCC
3. STM32429I-EVAL1 4.3" (ST STM32F4, Internal+External flash) - See Programming ST Boards when Compiling with GCC
4. STM324x9I-EVAL 5.7" (ST STM32F4, Internal+External flash) - See Programming ST Boards when Compiling with GCC
5. STM32F429I-DISCO (ST STM32F4, No external flash) - See Programming ST Boards when Compiling with GCC
6. STM32F746G-DISCO (ST STM32F7, Internal+External flash) - See Programming ST Boards when Compiling with GCC
7. STM32756G-EVAL (ST STM32F7, Internal+External flash) - See Programming ST Boards when Compiling with GCC

By inspecting the structure of the `target/` folder for any application, we may learn about which targets that application is intended to support. If an application was designed for a 4.3" display, any targets with 7.0" displays would typically not be represented here. The application `app/template/EmptyApplication` is an empty application shell that contains every target supported by TouchGFX. Each vendor folder contains supported target configuration from that vendor. Each target contains application project configurations for supported compilers.

```
+-- target
|   +-- EmbeddedArtists
|   |       +-- gcc
|   |       \-- IAR
|   |       \-- Keil
|   +-- FDI
|   +-- Mjolner
|   \-- ST
```

4.5.1 Building and downloading applications with IAR

Every IAR workspace file in the TouchGFX distribution incorporates a flash-loader configuration for the specific target it is designed for. For instance, if we navigate to the Button Example application at *app/example/button_example* we can open up an IAR target configuration for the STM32429I-EVAL1 board at `target/ST/STM32429I-EVAL1/IAR/application.eww` and prepare to download the *Button Example* application. To download the application simply attach a debugger of choice (must be supported by IAR) to the target hardware and configure the debugger in the projects configuration; In the image below we are using IAR's own i-jet debugger.

All TouchGFX libraries and applications have been built and tested using IAR Embedded Workbench for ARM version 6.60.1.5104 and compatibility beyond that point has not been thoroughly tested. IAR workspace files are compatible with version 7.20.

Note

STM32756G-EVAL: Currently, TouchGFX does not support an IAR flash loader for the *external* flash of the STM32F7 based EVAL boards. The IAR projects are currently configured to only program the internal flash and you should see an error during target programming related to a missing flashloader algorithm for any data starting at 0x90000000. To flash the external flash, open the application .hex binary produced by IAR (i.e. `/target/ST/STM32756G-EVAL/IAR/Debug/Exe/application`.hex) and flash using ST's "ST-Link" flash utility. This procedure is described in Programming ST Boards when Compiling with GCC.

4.5.2 Building and downloading applications with Keil uVision

The TouchGFX distribution offers support for both Keil uVision4, with preconfigured settings for flashing both the internal and external flash regions. Like with IAR, you will find a uVision project (.uvproj file) in the target folder for a given application. Please note that not all boards have support for Keil at the moment. Please refer to Supported Hardware for details on which boards support Keil out of the box. To deploy an example on an evaluation board, simply open the uVision project file, hit F7 to compile the code and press the "Load" button afterwards. You will then need to either start a debugging session from within uVision or simply power cycle the target to get the application running.

The Keil projects are preconfigured to use the J-Link debugger. This can be changed in project options, as shown below. But please note that in uVision, the flash programming algorithms for the target are tied to the chosen debugger, so if you change debugger, you need to also (re)configure the flash programming algorithms to match those used when J-link is selected.

Figure 4.6: Configuration of debuggger and flash algorithms in Keil uVision4

The screenshots above are from a specific target (LPC4357DevKit). The exact flash algorithms used vary among boards, so the example above is just a guideline. All evaluation boards have the correct flash algorithms installed

for the J-Link debugger, so examine that setting for correct flash algorithms for other debuggers.

4.5.2.1 Downloading Applications for ST ST3249xI eval board (4.3" and 5.7" display) with Keil uVision4

The Keil uVision projects for the ST evaluation boards use a custom flash programming algorithm for the external NOR flash, which must be copied to your uVision installation directory in order to be able to flash your application. You will find the flash loader algorithm here:

```
touchgfx_install_dir\app\template\EmptyApplication\target\ST\STM324x9I-CUBE\Keil\M29W128GL-ST.FLM
```

This file must be copied to the following folder (assuming that Keil uVision4 is installed in C:\Keil):

```
C:\Keil\ARM\Flash
```

After copying the flash algorithm file, please make sure that your uVision project settings correctly specify the use of this flash algorithm, and that the "RAM for algorithm" size is large enough (at least 0x2800 bytes):

Figure 4.7: Configuration of debuggger and flash algorithms in Keil uVision for ST eval boards

4.5.2.2 Downloading Applications for ST STM32756G-EVAL and STM32F746G-DISCO boards with Keil uVision5

The Keil uVision projects for the STM32756G-EVAL and STM32F746G-DISCO boards use a custom flash programming algorithm for the external NOR flash (QSPI), which must be copied to your uVision installation directory in order to be able to flash your application. You can find the flash loader algorithms here:

```
touchgfx_install_dir\app\template\EmptyApplication\target\ST\STM32F746G-DISCO\STM32F7_Disco_Micron.flm
touchgfx_install_dir\app\template\EmptyApplication\target\ST\STM32756G-EVAL\STM32F7xx_QSPI_Micron.flm
```

This file must be copied to the following folder (assuming that Keil uVision5 is installed in C:\Keil_v5):

```
C:\Keil_v5\ARM\Flash
```

Note

Updating your device packs through the Keil package manager would also give you access to the above flashloaders. For isntance, The flashloader for STM32756G-EVAL exists in device ST pack 2.2.0

```
i.e. C:\Keil_v5\ARM\Pack\Keil\STM32F7xx_DFP\2.2.0\Flash
```

After copying the flash algorithm file, please make sure that your uVision project settings correctly specify the use of this flash algorithm, and that the "RAM for algorithm" size is large enough (at least 0x2800 bytes. For EVAL, 0x5000). The F7 based boards are configured to use ST-Link per default.

Figure 4.8: Configuration of flash algorithms

4.5.3 GCC Building applications with GNU GCC compiler for ARM

TouchGFX offers GCC support for several hardware platforms (See Supported Hardware), but does not provide or support any flash loaders officially. The MinGW development environment for Windows (See Development Environment) supports and comes with a pre-installed GNU GCC compiler suite for ARM processors. Those who don't use the MinGW development environment can download the GCC compiler suite `here`. The version used by TouchGFX is currently 4.8.4 20140725 (labeled 4.8-2014-q3).

To build an application for i.e. the STM32429I-EVAL1 board, navigate to an application that supports this target (indicated by the contents of the target/ folder for any application) and execute the following command that will generate code from application assets and build the application.

```
$ make -f target/ST/STM32429I-EVAL1/gcc/Makefile
```

The `build/bin` folder now contains the following files:

- `application.map`
- `extflash.bin`
- `intflash.elf`
- `intflash.hex`
- `target.elf`
- `target.hex`

The build process outputs binaries for both internal and external flash for easier download, but also outputs a combined binary (`target.elf`) if you have access to a tool that can flash both internal and external flash from a single .elf file. While you may use whatever tools you wish, but we supply a few methods below.

Note

Binaries for external flash are not produced for targets with no external flash.

4.5.4 Programming TouchGFX demo board 4.3" LPC4350 when Compiling with GCC

If using the GNU GCC compiler for ARM, boards with NXPs LPC4350 MCU (No internal flash) can flash an external, "Common Flash Interface" (CFI)-compliant NOR flash by using SEGGERs J-Link GDB Server (This requires a SEGGER J-Link debugger). The following describes, in detail, one process for flashing LPC4350 based boards, using the TouchGFX demo board 4.3", LPC4350 based, evaulation board as an example. The programming method described in this section can be generalized for other boards using the LPC4350 MCU and has a CFI compliant NOR flash.

The SEGGER J-Link GDB server can be downloaded `here` and the documentation can be found `here`.

Let's start off by compiling the "Custom transition example" into an .elf file that we can program onto the evaluation board. From the root of our TouchGFX installation we execute the following commands:

```
$ cd app/example/custom_transition_example
$ make -r -f target/Mjolner/TouchGFXDemoBoardRev1.1-4350-4.3/gcc/Makefile
> Linking build/bin/target.elf
```

The following is an example GDB configuration file that sets up our GDB client with a few common settings for programming and debugging within the TouchGFX environment. Place this configuration in ~/.gdbinit (If using the touchgfx windows development environment, this resolves to `/c/touchgfx-env/MinGW/msys/1.0/home/{user}`). Execute these commands manually if you want to avoid a central configuration.

```
set auto-load safe-path /

# Connect to the GDB Server
target remote :2331
file build/bin/target.elf
monitor reset
monitor halt
monitor endian little
monitor speed auto
monitor flash device LPC4350_M4
```

These commands perform the following steps automatically:

1. Connects to a GDB server listening on port 2331 (Default port)

2. Loads the `target.elf` file for the application we've just built using the `file` command. The argument provided to `file` is relative to the application root folder.

3. Resets the CPU and halts it so that we may read register values

4. Ensure little-endianness

5. Sets JTAG speed to auto

6. Lets GDB know which device we're flashing. This, combined with the information from the .elf file will tell GDB about memory regions.

Configure the connection to the SEGGER JLink GDB server:

Start the SEGGER Jlink GDB Server and await incoming connections from GDB client:

Connect to the GDB server using a `gdb` client. By not specifying any configuration file argument, `gdb` will default to ~/.gdbinit.

```
$ gdb
> GNU gdb (GDB) 7.6.1
```

The GDB server accepts the incoming connection:

Execute the `load` command to initiate programming of the `build/bin/target.elf` file using SEGGERs J-Link flash-loader. Information about flash memory regions is read from the .elf file and no further configuration is required.

```
(gdb) load
>Loading section .text, size 0x95c8 lma 0x14000000
>Loading section .rodata, size 0x2a84 lma 0x140095c8
>Loading section ExtFlashSection, size 0xc8c00 lma 0x1400c04c
>Loading section .init_array, size 0x10 lma 0x140d4c4c
>Loading section .ARM.exidx, size 0x8 lma 0x140d4c5c
>Loading section .data, size 0x8bc lma 0x140d4c64
>Start address 0x14007ae1, load size 873760
>Transfer rate: 12548 KB/sec, 15064 bytes/write.
```

The GDB server will start downloading data based on the above load information.

SEGGER J-Link GDB Server V4.92
File Help
GDB 1 client @ 127.0.0.1
J-Link Connected
CPU LPC4350_M4, Halted
Initial JTAG speed 10 kHz
Current JTAG speed 2000 kHz
3.25 V
Little endian
Localhost only
Stay on top
Show log window
Generate logfile
Verify download
Init regs on start
Log output:
Clear log
Downloading 16128 bytes @ address 0x14000000 - Verified OK
Downloading 16160 bytes @ address 0x14003F00 - Verified OK
Downloading 6056 bytes @ address 0x14007E20 - Verified OK
Downloading 10884 bytes @ address 0x140095C8 - Verified OK
Downloading 16228 bytes @ address 0x1400C04C - Verified OK
Downloading 16352 bytes @ address 0x1400FFB0 - Verified OK
Downloading 16352 bytes @ address 0x14013F90 - Verified OK
Downloading 16352 bytes @ address 0x14017F70 - Verified OK
Downloading 16352 bytes @ address 0x1401BF50 - Verified OK
Downloading 16352 bytes @ address 0x1401FF30 - Verified OK
Downloading 16352 bytes @ address 0x14023F10 - Verified OK
Downloading 16352 bytes @ address 0x14027EF0 - Verified OK
Downloading 16352 bytes @ address 0x1402BED0 - Verified OK
Downloading 16352 bytes @ address 0x1402FEB0 - Verified OK
SEGGER J-Link V4.92 - Flash download (896 KB)
Compare 100.0% 0.283s
Erase 100.0% 1.158s
Program 45.0% 2.880s
Verify 0.0%
Cancel
Programming range 0x14088000 - 0x1408BFFF (16 KB) 4.321s
Downloading 16352 bytes @ address 0x14067CF0 - Verified OK
Downloading 16352 bytes @ address 0x1406BCD0 - Verified OK
Downloading 16352 bytes @ address 0x1406FCB0 - Verified OK
Downloading 16352 bytes @ address 0x14073C90 - Verified OK
Downloading 16352 bytes @ address 0x14077C70 - Verified OK
Downloading 16352 bytes @ address 0x1407BC50 - Verified OK
Downloading 16352 bytes @ address 0x1407FC30 - Verified OK
Downloading 16352 bytes @ address 0x14083C10 - Verified OK
Downloading 16352 bytes @ address 0x14087BF0 - Verified OK
Downloading 16352 bytes @ address 0x1408BBD0 - Verified OK
Downloading 16352 bytes @ address 0x1408FBB0 - Verified OK
Downloading 16176 bytes @ address 0x14093B90 - Verified OK
Downloading 16064 bytes @ address 0x14097AC0 - Verified OK
Downloading 16048 bytes @ address 0x1409B980 - Verified OK
Downloading 16048 bytes @ address 0x1409F830 - Verified OK
Downloading 16064 bytes @ address 0x140A36E0 - Verified OK
Downloading 16064 bytes @ address 0x140A75A0 - Verified OK
Downloading 15952 bytes @ address 0x140AB460 - Verified OK
Downloading 15824 bytes @ address 0x140AF2B0 - Verified OK
Downloading 15760 bytes @ address 0x140B3080 - Verified OK
Downloading 15760 bytes @ address 0x140B6E10 - Verified OK
Downloading 16032 bytes @ address 0x140BABA0 - Verified OK
Downloading 16304 bytes @ address 0x140BEA40 - Verified OK
Downloading 16352 bytes @ address 0x140C29F0 - Verified OK
Downloading 16352 bytes @ address 0x140C69D0 - Verified OK
Downloading 16352 bytes @ address 0x140CA9B0 - Verified OK
Downloading 16336 bytes @ address 0x140CE990 - Verified OK
Downloading 8940 bytes @ address 0x140D2960 - Verified OK
Downloading 16 bytes @ address 0x140D4C4C - Verified OK
Downloading 8 bytes @ address 0x140D4C5C - Verified OK
Downloading 2236 bytes @ address 0x140D4C64 - Verified OK
1706 KB downloaded
2 JTAG devices, IRLen=8

After programming of the external flash has finished, reset the CPU (`monitor reset`) and take it out of halt (`cont`).

```
(gdb) monitor reset
>Resetting target
(gdb) cont
>Continuing.
```

The application should now be running on target. Here's the GDB server log in it's entirety:

```
>Resetting target
>Target endianess set to "little endian"
>Select auto target interface speed (2000 kHz)
>Selecting device: LPC4350_M4
(gdb) load
>Loading section .text, size 0x95c8 lma 0x14000000
>Loading section .rodata, size 0x2a84 lma 0x140095c8
>Loading section ExtFlashSection, size 0xc8c00 lma 0x1400c04c
>Loading section .init_array, size 0x10 lma 0x140d4c4c
>Loading section .ARM.exidx, size 0x8 lma 0x140d4c5c
>Loading section .data, size 0x8bc lma 0x140d4c64
>Start address 0x14007ae1, load size 873760
>Transfer rate: 12548 KB/sec, 15064 bytes/write.
(gdb) monitor reset
>Resetting target
(gdb) cont
>Continuing.
```

4.5.5 Programming TouchGFX demo board 4.3"/7.0" LPC4353 when Compiling with GCC

If using the GNU GCC compiler for ARM, boards with NXPs LPC4353 MCU can flash an external, "Common Flash Interface" (CFI)-compliant NOR flash by using Segger J-Flash software with a J-Link debugger. The following describes, in detail, one process for flashing LPC4353 based boards, using the TouchGFX demo board 4.3", LPC4353 based, evaluation board as an example. The programming method described in this section can be generalized for other boards using the LPC4353 MCU and has a CFI compliant NOR flash.

The SEGGER J-Flash software can be downloaded `here`.

Initial setup:

To enable flashing of the LPC4353 demo board you have to add the following lines the J-Flash files:

Add

```
"NXP;LPC4353_M4 (SPIFI);LPC4353_M4 SPIFI;0E0000FF;0;0;0;4BA00477;14000000;10000000;A000"
```

to the end of the `etc/jflash/mcu.csv` file.

Add

```
"NXP;LPC4353_M4 SPIFI;0;0;1;56;0;32;1;0;0;0;0;0;0;0;0;0;0;0;0;"20000[512];"
```

to the end of the `etc/jflash/Flash.csv` file

Flashing:

1. Open J-Flash and select "Create a new project".
2. Select "Options / Project Settings". In the "Project Settings" pop-up select the CPU tab. Choose the "Device" radio button and press the "..." button to select a new device. Type NXP and select the LPC4353_M4 entry. Press Apply and Ok to close the pop-up.

3. Select "Target / Erase Chip (F4)"
4. Do the same as in step 2. except that you now select the LPC4353_M4(SPIFI) entry.

5. Select "Target / Erase Chip (F4)". J-Flash might complain about unknown version of the device. In that case press "yes" and select NXP LPC4353_M4.

6. Flash external flash from address 14000000. To do this: Select "File / Open data file". Select the extflash.bin (default placed in `build/bin/`). Select "Target / Program (F5)".
7. Repeat step 2.
8. Flash internal flash: Select "File / Open data file". Select the intflash.hex (default placed in `build/bin/`). Select "Target / Program (F5)".
9. Power cycle the target.

And the flashing should be completed.

4.5.6 Programming ST Boards when Compiling with GCC

ST provides a Windows based utility, ST-Link, for flashing the internal and external flash of ST boards. You can download the `STM32 ST-Link Utility here`. The process below outlines how to use the utility to program the internal and external flashes of the ST boards supported by TouchGFX. ST-Link will automatically detect the type of device connected and know how to flash the internal flash, but the external flashloader must be configured manually. Currently, TouchGFX supports the following boardpackages:

1. STM32469I-DISCO
2. STM32469I-EVAL
3. STM324x9I-CUBE
4. STM32756G-EVAL
5. STM32F429I-DISCO
6. STM32F746G-DISCO

Instead of programming each flash individually, you may load the combined internal+external hex binary, target.hex. This will automatically detect the memory ranges to program, and select a suitable (if configured) flashloader to use.

```
[target.hex] opened successfully.
Address Ranges [0x08000000 0x08010364] [0x90000000 0x9000CA28]
```

The procedure below outlines how to flash the internal and external individually. The accompanying screenshots are based on the STM324x9I-EVAL board.

Program the external flash:

1. Select "Add External Loader" from the "External Loader" menu.
2. Select the appropriate flashloader in the list of supported boards (i.e. MT25QL512A_STM32756G-EVAL for STM32756G-EVAL and N25Q128A_STM32F746G-DISCO for STM32F746G-DISCO).
3. Open the appropriate binary (`extflash.bin`). The start address is automatically provided.
4. Select "Program" from the External flash loader menu.

Program the internal flash :

1. Open the appropriate binary (`intflash.hex`). The start address is automatically provided.
2. Select "Program" from the "Target" menu.

Note

Consider having only a single external flashloader configured in ST-link active at any given time.

Figure 4.9: Step1 (Flash external): Add External Flash loader

Figure 4.10: Step 2 (Flash external): Select External Flash loader

Figure 4.11: Step 3 (Flash external): Select Program

Figure 4.12: Step 4 (Flash external): Download binary

Figure 4.13: Step 1 (Flash internal): Select Program

Figure 4.14: Step 2 (Flash internal): Download binary

Chapter 5

Application Development

This page describes the key points of TouchGFX application development. New to TouchGFX? If so, you will most likely find the initial sections that follow this introduction useful as they outline the basics of application development:

1. Application File Structure detailing the contents of a standard TouchGFX application
2. Application assets and their role in application development

To get you started on application development, here is a common development scenario:

1. Creating a New Application Shows you how to create a new application and shows you how to configure the location of your TouchGFX framework installation relative to the location of your application.
2. Project Configuration goes through the various application specific settings available to developers
3. Examples offers insight into how to construct simple applications with the standard TouchGFX widget library such as buttons, transitions, lists, texts, containers etc.

If you feel ready to go further than introductory application development:

1. Concepts talks about events, memory allocation, custom widgets and containers, and how to use texts, fonts and images.
2. Miscellaneous covers minor topics and tips and tricks.

5.1 Application File Structure

As a general rule, the application examples provided with TouchGFX are not a part of the actual TouchGFX framework, but are simply provided as part of the distribution. By default, an application in TouchGFX conforms to the following folder structure which is natively supported by the build configurations in the distribution. Deviating from this folder structure without modifications to the build configurations provided by TouchGFX can potentially cause build-errors.

```
+-- assets
|   +-- fonts
|   +-- images
|   \-- texts
|
+-- config
|   +-- gcc
|   \-- msvs
|
+-- generated
|   +-- fonts
```

```
|   +-- images
|   \-- texts
|
+-- gui
+-- platform
+-- simulator
\-- target
```

5.1.1 Application assets

Typically, an application makes use of a number of image, font and textual *assets*. By convention, asset files should be placed by the user in their respective folders in `assets` of the root of an applications folder. The Font-, Image- and Textconversion tools of TouchGFX expect assets to conform to this guideline and any assets not located in the appropriate folder will either not be processed, or cause a failure during conversion.

A prerequisite to successful compilation of either of the supplied applications in TouchGFX is code generation for the applications assets: *images*, *fonts*, *texts*.

Any code generated by these converter tools is placed in the folder `generated`, mirroring the structure of the `assets` folder. For generated application assets only, the folder structure is as seen below.

```
+-- generated
|   +-- fonts
|   |   +-- include
:   :   \-- src
:   :
```

Notice, contrary to the rest of the framework, that the `fonts` folder (and also `images` and `texts`) is a self contained component with source code and header files that must be added to the list of include directories for a build configuration that wishes to use it. This allows you to place these *application specific* components, where ever you desire as long as you make sure to modify your build configuration. If you inspect any of our IAR build configurations you would see the following include folders related to assets (similar for Keil uVision).

```
$PROJ_DIR$\..\..\..\..\generated\fonts\include
$PROJ_DIR$\..\..\..\..\generated\images\include
$PROJ_DIR$\..\..\..\..\generated\texts\include
```

5.1.1.1 Font assets

Font assets are located in the folder `assets/fonts/`. When compiling an application, the Font Converter will convert these font assets to source code, which is output to the `generated/fonts` directory. See Font Converter for details on fonts and texts.

5.1.1.2 Image assets

Image assets are located in the folder `assets/images`. When compiling an application, the Image Converter will convert the image assets to source code, which is output to the `generated/images` directory. When this code is a part of project, images are made available to application developers through an identifier based on the original file name of the asset. For details on how images are converted and how to use images in the source code of your application, see Image Converter. Also be sure to have a look at the Examples, which all use bitmap graphics.

5.1.1.3 Text assets

TouchGFX provides an easy way of defining text strings, languages and translations for use in applications. The `assets/texts` folder contains an Excel spreadsheet called `texts.xlsx` containing text definitions for a specific application. Source code based on these definitions is generated to the `generated/texts` directory by the

TouchGFX `Text` Converter. See Text Converter for details on using texts in your application, as well as the Text Example for a concrete example of using texts and translations in an application.

5.1.2 Application configuration

The files in the `config` folder contain *user defined* and application specific settings that used as input to the TouchGFX build environment for various compilers (currently available for Microsoft Visual Studio and GCC). Please see Project Configuration for further details on choosing the settings for your build/project to, among other things, achieve optimized target performance.

5.1.3 Graphical User Interface

The `gui` folder of any application provided by TouchGFX contains files related to the graphical user interface. TouchGFX requires knowledge, at compile-time, about the amount of RAM a user interface is going to use up. Such a memory allocation scheme (See Memory Allocation) is conveniently provided by TouchGFX in the form of a well defined architectural pattern helpful in constructing applications with graphical user interfaces (See Model-View-↩ Presenter). The main concepts of this architecture are:

- The `model` is an interface defining what data should be presented or acted upon.
- The `view` is a passive interface that displays data from the model and routes events (i.e. user commands) to the presenter.
- The `presenter` acts upon the model and the view. It retrieves data from the model, processes it, and displays it in the view .

You'll see that the naming of code files for all applications in TouchGFX take their name according to the above principles. For details on building a graphical interface using the widget library of TouchGFX, see Concepts.

5.1.4 Platform Configuration

Any common files related to a target platform, such as operating systems configurations are, by convention, placed by TouchGFX in the `platform` folder. We provide an optional and free operating system `FreeRTOS` (See RT Operating System (touchgfx/os/) on how to replace the standard OS supplied by TouchGFX). The folder `platform/os` contains the file `FreeRTOSConfig.h`, located under `platform/os` is included and used by IAR, Keil and GCC target build configurations throughout the framework. Among other things, the FreeRTOS configuration defines OS stack- and heapsize, hook usage, interrupt priority levels and maps FreeRTOS interrupt handlers to CMSIS standard names.

5.1.5 PC Simulator

The folder `simulator` in any application contains PC simulator build configurations for GCC and Microsoft Visual Studio. See PC Simulator for more on details on how to build and execute simulator applications.

5.1.6 Target

The `target` folder for any of the supplied sample applications contains build configurations for target hardware that is able to run that particular application which might require certain display dimensions not supported by all targets. As a rule, any graphical user interface code designed for certain display dimensions will run on any board that supports those dimensions if a valid target build configuration for those targets exists. The `target` folder of the *template* application called `EmptyApplication` is an empty shell designed to help people get started on application development and contains build configurations for all demo boards supported by TouchGFX. As such, the example and demo applications provided by TouchGFX do not contain all targets supported by the framework because some of those applications are designed for a specific display resolution. For instance, every example is designed for a 480 x 272 (4.3") resolution. Three supported boards support these dimensions:

1. Embedded Artists LPC4357DevKit
2. Mjolner TouchGFXDemoBoardRev1.1-4353-4.3
3. ST STM32429I-EVAL1

See Board Packages (touchgfx/board/) for an overview of how such target build configurations make use of readily available hardware and software initialization in the shape of what is called *board packages*.

5.2 Creating a New Application

The application located under `app/template/EmptyApplication`, as previously mentioned, is a basic, empty application that can serve as a basis for new applications. It contains build configurations for all of the evaluation boards supported by TouchGFX (IAR, Keil or GCC) and one for for the PC simulator (MinGW or MSVS). While `EmptyApplication` contains build configurations for every supported evaluation board your choice of compiler might not be supported for that board. For a complete list see Supported Hardware. As a general rule, IAR is supported by all boards, while GCC and Keil support is limited.

Navigate to the template folder from the root of your TouchGFX installation, `app/template/`. The folder `empty_application` can be copied, renamed to something suitable and help you get started on application development on PC simulator or any of the supported targets. See Project Configuration to configure your application relative to your TouchGFX installation folder along with other application specific features.

```
cp -r empty_application your_application_folder/new_application
```

After copying the template application `empty_application` make sure the TouchGFX framework path in the configuration of the application is correct. To test the application, build the PC simulator:

```
cd your_application_folder/new_application
make -f simulator/gcc/Makefile
./build/bin/simulator.out
```

As for the target configurations supported by the template application, keep in mind that not all targets support the same display dimensions; An application developed for 320x240 QVGA would not look right on a larger screen. So, remember to remove any non-supported targets from the `target/` folder.

Moving an application around on disk might affect any of the projects projects of an application in that you might need to re-add all files that have changed location on disk. You may also need to update your relative paths the C/C++ and Assembler pre-processors along with references to libraries used by the linker.

5.2.1 Visual Studio Project Creation

Creating new projects for use with Visual Studio can be handled in almost the same way as the general case.

- Copy the template application `EmptyApplication` from the folder `app/template/`. This will give you a basic Visual Studio solution with a default project.
- If you wish, you can rename all the Visual Studio specific files (.sln, vcxproj, .filters) to the name of the new application, e.g. NewApplication
- Open the solution and project file, `NewApplication.sln` and `NewApplication.vcxproj` in an editor.
- Replace all occurrences of TemplateApplication with NewApplication
- **Note** : If the NewApplication project is going to be included in solutions with other projects, one should replace the project GUID (found in the project file) in the project and solution files with a newly generated GUID, look for

  ```
  <ProjectGuid>{6E061874-2DC8-4BF4-9B83-12FE838C5E20}</ProjectGuid>
  ```

5.2.1.1 Starting from scratch

Creating an entirely new project in Visual Studio and setting it up to adhere to the TouchGFX standard is also an option:

- Create an empty, no precompiled header, C++ Win32 console application project
- Copy the `TemplateApplication.props` property sheet to the new project, and add this property sheet to the project using the Property Manager
- Add the property sheet found in `framework/config/msvs/touchgfx.props` to the project using the Property Manager
- Add the build customization found in `framework/config/msvs/prebuild_touchgfx.targets` to the project
- Add all *.hpp and *.cpp files (from `/gui` and `/simulator`) to the project
- Adjust the properties found in the newly added property sheets. The TouchGFXReleasePath should point to where the framework folder is located
- Add the assets folder and the images, texts and fonts folder below this
- *Optional* Set up filters for header and source files
- Build and run

5.3 Project Configuration

When generated in an appropriate format, image- and textual assets, can greatly enhance the performance of some systems. This section describes how to, among other things, achieve such a performance boost through application configuration. Currently, TouchGFX supports asset generation through `gcc` and Microsoft Visual Studio projects. Any other supported compiler, such as IAR, have to rely on one of the mentioned compilers for asset generation.

The following table gives you an overview of the possible configuration options for supported compilers.

Property	Description	GCC name	MSVS name
Framework path	This is a required property to let an application know where to find the TouchGFX installation. See Project Configuration.	`touchgfx_path`	`TouchGFXRelease↩Path`
Opaque Image Format	This property determines the format of solid images. See Image Output Format.	`opaque_image_↩format`	`OpaqueImage↩OutputFormat`
Non-opaque Image Format	This property determines the format of non-solid images. See Image Output Format.	`non_opaque_↩image_format`	`NonOpaqueImage↩OutputFormat`
Alpha Dither	This property determines whether the alpha channel of bitmaps should be dithered. See Alpha Dithering.	`alpha_dither`	`AlphaDitherImage`

Screen Orientation	Let's you change the orientation of generated assets. See Screen Orientation	`screen_orientation`	`RotateImage90`
Text Data Format	Let's you change the data format of generated text assets. See Text Format.	`text_data_format`	`TextDataFormat`
Additional C/C++ compiler flags	For users who want to supply additional compiler flags such as optimizations and symbols.	`user_cflags`	*not supported*

The following listings show a very common application configuration for each of the supported compilers with standard orientation of images, images without alpha-channel and standard text data format.

5.3.1 GCC

Application specific GCC configurations from the folder are located at the root of each application in the folder `config/gcc`. These configurations are automatically imported by the Makefile for that same application.

```
# Relative location of the TouchGFX framework from root of application
touchgfx_path := ../../../touchgfx

# Optional additional compiler flags
user_cflags :=

# Settings for image converter output format
alpha_dither := yes
opaque_image_format := RGB565
non_opaque_image_format := RGB565

# Settings for image converter screen orientation (empty string = default value, -rotate90 rotates the image 9
screen_orientation :=

#Settings for Hardware accelerated text rendering on ST32F4 devices. Must correspond to value of bitsPerPixel
#If A4 blitcap is enabled for target specific HAL the fontconverter must generate compliant data format, other
text_data_format :=
```

5.3.2 Microsoft Visual Studio 2010/2012

All MSVS configurations in TouchGFX import the following application specific configuration from the `config/msvs` located at the root of the application: `config/msvs/Application.props`. These properties can also be edited using the MSVS IDE through the menu

```
Property Manager > project_name > Debug|Win32 > project_name icon > Common Properties > User Macros
```

A Property group called `UserMacros` is defined in the .props file used by the Visual Studio project for the application, mirroring the settings from the `GCC` configuration above.

```
<PropertyGroup Label="UserMacros">
  <RotateImage90>false</RotateImage90>
  <TouchGFXReleasePath>$(ProjectDir)\..\..\..\touchgfx</TouchGFXReleasePath>
  <AlphaDitherImage>yes</AlphaDitherImage>
  <OpaqueImageOutputFormat>RGB565</OpaqueImageOutputFormat>
  <NonOpaqueImageOutputFormat>RGB565</NonOpaqueImageOutputFormat>
  <TextDataFormat></TextDataFormat>
</PropertyGroup>
```

5.3.3 Screen Orientation

Screen orientation must be configured if the user interface orientation (portrait or landscape) is different from the native orientation of the TFT display itself. A TFT display is said to be native portrait if its width is smaller than its height. Please refer to Switching Display Orientation for details on how to correctly set this property.

5.3.4 Image Output Format

The image converter takes application image assets and convert them into a specific format based on two application configuration settings set by the user *opaque image format* and *non-* opaque *image format*. Depending on the target platform, the value of these properties can help optimize performance drastically.

The supported image formats are:

- `RGB565`
- `ARGB8888`
- `BW`
- `BW_RLE`

The default output format for solid images is `RGB565` and the default for images containing transparency is `RGB565` plus an alpha channel array. This format is optimized to give the best performance for most hardware setups. The `ARGB8888` format is supported by any platform in TouchGFX using the STMicroelectronics ST-STM32F429/439 microcontroller, enabling the ChromArt accelerator. Referring to Application configuration, we would configure gcc and msvs builds, respectively, to generate images in the RGB565 format by supplying the following value `RGB565` to both solid images (opaque) and transparent images (non-opaque).

The BW and BW_RLE settings are for monochrome displays only. See Configuring TouchGFX for 1BP↩P Monochrome for details.

Note that if you do not have any good reason to choose a different output format you should always use the default RGB565 setting.

GCC:

```
# Settings for image converter output format
opaque_image_format := RGB565
non_opaque_image_format := RGB565
```

MSVS:

```
<OpaqueImageOutputFormat>RGB565</OpaqueImageOutputFormat>
<NonOpaqueImageOutputFormat>RGB565</NonOpaqueImageOutputFormat>
```

MSVS property configuration can by selecting the Property Manager and right-clicking the *project-name/Debug*|↩*Win32/project-name* icon to bring up the property pages for the project. Select "Common Properties/User Macros". A similar setting is available for the release build.

5.3.5 Alpha Dithering

Since the color channels are only 5/6 bits, having an 8-bit alpha channel will cause many blend operations to yield the same resulting color, thereby causing banding if the alpha channel is a gradient. With alpha dithering enabled, the alpha channel of each bitmap will be dithered from 8 to 5 bits. It is recommended to enable this setting as it will make gradients in the alpha channel appear smoother.

5.3.6 Text Format

The *text format* property of the application configuration can be set to enable hardware rendering of 4bpp *(bits per pixel)* or 8bpp text if your target is a `Chrom-ART accelerated ST platform`. By default, any such platform supported by TouchGFX is already *capable* of rendering in either of these formats but it is up to the *user* configuring the project to enable this feature by setting the appropriate value for the property relating to *text data format* in the application configuration.

Both the ST platforms supported by TouchGFX have ChromeArt capabilities (STM32F4 series) that support hardware rendering of texts if fonts are generated in the following two data formats:

1. A4 - Requires additional application configuration to alter the format output by the font converter
2. A8 - Requires *no* additional application configuration

The following configurations would yield hardware accelerated rendering of 4bpp TTF fonts on STM32F4 based platforms. For this configuration to have any effect, the text typograph sheet in *assets/texts/texts.xlsx* should specify 4bpp for the desired font as specified in the table below, mimicking a row from the typography sheet. Likewise for 8bpp, except that no value for *text data format* in the application configurationis required.

Typograhy name	Font	Size	Bpp
Headline	RobotoCondensed-↩ Reguar.ttf	40	4

GCC:

```
#Setting for Hardware accelerated text rendering on ST32F4 devices. Possible values: A4 or no value
text_data_format := A4
```

MSVS:

```
<TextDataFormat>A4</TextDataFormat>
```

Note

Any attempts, by the DMA controller, to make transfers in 4bpp mode using font data non-compliant with the A4 format will result in a DMA Controller Configuration Error.

5.4 Assets and Tools

A TouchGFX application includes a set of asset files as described in Application assets. These assets are converted at compile time to internal C++ formats by various external converter tool. The converter tools are each described below. The converted files are all placed in the `generated/` folder.

5.4.1 Font Converter

The TouchGFX Font Converter accepts TrueType (.ttf), OpenType (.otf) fonts and Glyph Bitmap Distribution Format (.bdf). Simply place the font in the `assets/fonts/` folder and the font will be available for reference in the text database (see Text Database).

Please note that a TouchGFX license does not in any way provide licenses for commercial use of any TrueType or OpenType fonts.

5.4.1.1 Kerning support

Kerning is the process of adjusting the spacing between characters in a proportional font, usually to achieve a visually pleasing result. The TouchGFX font converter supports kerning by using the kerning information in the supplied font.

5.4.2 Image Converter

The following image formats are supported.

- 24-bit bmp images (alpha channel not supported)
- PNG images (24-bit RGB or 32-bit RGBA colors)

The .png and .bmp images located in the assets/images folder or subfolders hereof are converted and placed in the generated/images folder. The conversion does Floyd-Steinberg dithering in order to reduce color banding in the generated 16 bit bitmaps.

The image conversion process supports opaque, transparent and semitransparent pixels when converting PNG images. Beware to use only what is necessary since the format has a huge impact on the overall performance and memory usage.

- Images containing only opaque pixels take up the smallest amount of memory and can in almost all cases be drawn efficiently by the hardware.
- Images containing only opaque or fully transparent pixels take up the same amount of memory as the opaque case, but must in most cases be drawn by software. And thus is more demanding on the MCU.
- Images containing semitransparent pixels take up more memory and will in almost all cases have to be drawn by software. It is uncommon in hardware to support per pixel alpha transparency, thus the software has to do a lot of work to read, blend and write pixels. Therefore it is advisable to limit the use of per pixel alpha transparency due to performance issues.

To know more on how to use the generated images in the application read the section Using Images.

In the section Image Output Format you can read more on how to configure the output format of the Image Converter.

To change the placement of bitmap data (internal vs. external flash), see Changing the Location of Bitmap Data

5.4.3 Text Converter

The Text Converter is an external tool that converts the information in the text spreadsheet (see below) to an internal C++ format used both by the simulator and on target. The tool is an integrated part of the build tool-chain and will be executed automatically when building the simulator. The Text Converter issues an error if the text spreadsheet violates any of the syntax rules. The Text Converter is not executed if the text spreadsheet has not been updated since last build. Beware that no warnings will be issued in this case. Either change or touch the text.xlsx file or rebuild if you need the Text Converter to be re-executed.

5.4.3.1 Text Database

The handling of text strings and translations in TouchGFX application are done in the `assets/texts/texts.xlsx` Excel spreadsheet. Please note that other programs than Excel can be used for editing the spreadsheet such as OpenOffice and LibreOffice.

This spreadsheet consists of two sheets. One defining the typographies used in the application and one defining the text strings and translations that are used.

The Typography Sheet

The typography sheet lets you define all typographies that should be used in the application. A typography describes a specific text layout and consists of four values.

- **Typography Name:** The name which you should reference the typography by in the Translation sheet
- **Font:** The name of the font to use for the given typography. This should match the name of a font located in the `assets/fonts/` directory. It must include the complete file name including extension. Example: miso-regular.ttf

- **Size:** The font size of the typography.
- **Bpp:** Bits per pixel. The number of bits that are used per pixel to represent the font. Legal values are `1`, `2`, `4`, `8`.

Note: If the Font is a Glyph Bitmap Distribution Format font (.bdf), not all font sizes can be rendered with the font. If the given size in the typography sheet does not match with the given font, the font convert utility will write a report with the supported font sizes, e.g. "Number of fixed sizes: 1 (15)" signifying that there is only one supported font size in the font, and that size is 15. Updating the size in the sheet to one of the supported sizes will solve the problem.

The Translation Sheet

The translation sheet lets you define all text strings used in the application. A row represents one text string to be used in the application and includes the typography, alignment, and translations to all supported languages.

One row consists of the following values:

- **Text ID:** The id of the string. This is the id that should be used to reference the text string in the application code. This is done by including the `texts/TextKeysAndLanguages.hpp` file which defines all the strings and languages. Notice that all text ids are prepended with T_ and converted to uppercase in the C++ definition.
- **Typography Name:** Specifies which typography the text string should use as default. This must match a typography name from the Typography sheet.
- **Alignment:** Specifies the default horizontal alignment of the text string. Legal values are `LEFT`, `RIGHT`, and `CENTER`.
- **Direction:** Specifies the default text direction of the text string. Legal values are `LTR` (Left-to-Right) and `RTL` (Right-to-Left), the default being `LTR`. The direction column can be omitted if `LTR` should be the default direction for all rows. The `RTL` option is primarily used for Arabic, Hebrew or other languages written from right to left.
- **Language column:** A language column is a column for a specific translation of all text strings. You can define as many language column that you want, by adding new columns with language names. A language name must be uppercase and contain 1 to 3 letters. E.g. `GB`, `DK`, `USA`, `CH`.
- **Language specific typography:** It is possible to override the default typography used for a specific language. This is done by introducing a column with the format language-TYPOGRAPHY e.g. `GB-TYPOGRAPHY`. Legal values are typographies defined in the Typography sheet. This is particular useful if the language uses characters that are not included in the font of the default typography e.g. Chinese characters. If this value is left empty for a specific text string, the default typography is used.
- **Language specific alignment:** It is possible to override the default alignment used for a specific language. This is done by introducing a column with the format language-ALIGNMENT e.g. `GB-ALIGNMENT`. Legal values are `LEFT`, `RIGHT`, and `CENTER`. If this value is left empty for a specific text string, the default alignment is used.
- **Language specific text direction:** It is possible to override the default text direction used for a specific language. This is done by introducing a column with the format language-DIRECTION e.g. `GB-DIRECTI↩ON`. Legal values are `LTR` and `RTL`. If this value is left empty for a specific text string, the default direction is used.

For examples of how to use the Text Database see `example/text_example` and `example/arabic_↩text_example`.

Character Memory Optimization

TouchGFX is optimized for low memory consumption. By analysing the characters used for a specific typography the number of generated characters (in internal C++ format) are minimized to the characters that are actually used by the application.

Wildcards

It is possible to use run time values as part of text strings. This is possible by use of wildcards in the text strings. These are specified in the given format <∗>, where the ∗ represents an optional helping text which will not be included in the resulting text string. It is possible to have up to two wildcards in one text string. All translations for a given text must contain the same number of wildcards. The wildcard values are inserted at run time in the application C++ code. Example of wildcard usage: "The temperature is <insert_temperature>°". One detail to notice is that due to the character memory optimization the only characters that are generated for a specific typography is the one used in texts having this typography. So to make sure that all characters that are to be used for wildcard values are generated, it is suggested to make a dummy text row having all the characters that are needed by wildcards for a specific typography. Example of dummy text: "-0123456789". The wildcard format "<∗>" can be escaped by using backslash notation like this: "\<not a wildcard\>". This will result in the literal text "<not a wildcard>" being used in the application. See the text_example application for a wildcard example.

5.5 Concepts

This section discusses some of the underlying concepts of TouchGFX, some which are important to application developers.

5.5.1 Model-View-Presenter

Model-View-Presenter (MVP) is a common way of dividing responsibilites in UI applications. It separates the concerns of setting up the graphical elements UI, reacting to changes and maintaining UI state permanently. In the general case, a diagram of an MVP architecture can be seen here:

Figure 5.1: Generic MVP architecture

TouchGFX comes with a minimal baked in (but optional) variant of MVP. The TouchGFX flavour of MVP has one model and multiple pairs of views and presenters, one pair for each screen in the application. Using the built-in MVP mechanism you get the added benefit of a Memory Allocation scheme which helps reduce internal memory use of your application.

5.5.1.1 Model

The model takes care of the overall UI application state. The model will communicate with the rest of the system (the backend) in order to obtain the state needed for the UI. In this sense the model acts as a gateway to the rest of the system.

When changes occur in the model the current active Presenter is notified of the change. This is done via methods in the ModelListener interface in the application.

5.5.1.2 View

A View is responsible for setting up the graphical elements of the UI. A View intercepts user events (e.g. a Click↩ Event) and takes action, usually delegating to the Presenter. The View knows about the corresponding Presenter enabling the view to inform the presenter of changes.

5.5.1.3 Presenter

The Presenter orchestrates the Model and the View. The Presenter handles logic related to the UI elements. The Presenter is informed when changes occur in the Model and then takes action to synchronize the View appropriately. The Presenter is informed when changes occur in the View and then takes action to synchronize the Model appropriately.

5.5.1.4 FrontendApplication

An application has a FrontendApplication. See the examples in Examples. The FrontendApplication provides methods for transitioning between different concrete views, e.g gotoSomeScreen.

When the application needs to go to a screen the current Presenter instructs the FrontendApplication to change screens. In order to accomplish this switch of screens the FrontendApplication remembers the screen to go to. The actual switch will be postponed to the next tick. This is done to preserve the integrity of memory.

5.5.1.5 Switching Screens

Switching between screens is handled by the FrontendApplication. A switch is done in two steps. The first step, as described in FrontendApplication, remembers the screen to switch to. The second step actually performs the switch. The second step is performed on the next tick.

The actual switch is done using makeTransition located in MVPApplication.hpp. This switches to the requested view and presenter using a Transition.

First makeTransition cleans up the current view and presenter as well as the last performed transition. Then a new transition, view and presenter is created and initialized. These are then connected and made aware of the model. And the switch is complete.

5.5.1.6 Class Diagram, TouchGFX MVP

The following class diagram depicts how the MVP elements are related in TouchGFX applications:

Figure 5.2: TouchGFX MVP

5.5.2 Drawables

In TouchGFX, all built-in and custom widgets and containers must derive from the Drawable class. This base class contains a pure virtual draw function which must be implemented. It also provides a range of common functionality like setting the size and position of the drawable.

This chapter deals with understanding the usage, hierarchy and types of Drawables that comes with TouchGFX. The Custom Widgets and Containers chapter describes how to implement your own custom drawables. There are two distinct direct subclasses of Drawable, namely Widget and Container, organised according to the composite design pattern:

Figure 5.3: The composite pattern of Drawable, Widget and Container

What this basically means is that Widget and all its derivatives are *leaf* nodes, whereas you can add other drawables to Container and anything deriving from Container. The collection of available drawables in TouchGFX (as well as the ones you create yourself) will therefore either derive from Widget or Container, depending on whether it is able to contain other drawables.

5.5.2.1 Adding Drawables to a View

A View contains a single root Container to which the drawables to be displayed can be added. This is done by calling the add function of the view. The order in which drawables are added determines their Z-order. To remove a previously added drawable, call remove.

Internally, each drawable contains a pointer to the next drawable (called nextSibling), thus forming a linked list. The Container subclass additionally provides a pointer to its first child to form a hierachical structure. The following diagram shows an example of a simple scene graph:

```
1) container.add(bgImage)
2) container.add(myContainer)
3) container.add(keyboard)
4) container.add(text2)
5) myContainer.add(button1)
6) myContainer.add(text1)
```

Figure 5.4: Scene graph composition through linked lists

By using this approach each Drawable automatically provides the memory needed for the scene graph and it scales to any number of drawables. Although not displayed here, each Drawable also has a parent pointer so the root container can be reached from any drawable in the scene.

The Z-order of the drawables in the example is from highest (front) to lowest (back) is:

1. text2
2. keyboard
3. text1
4. button1
5. myContainer
6. bgImage
7. container

5.5.2.2 Coordinate Systems

Coordinates can be expressed in one of two ways. The first is *absolute* coordinates where the following is true:

- Upper left corner is (0, 0)
- Upper right corner is (WIDTH-1, 0)
- Lower left corner is (0, HEIGHT-1)
- Lower right coner is (WIDTH-1, HEIGHT-1)

The low-level classes such as LCD operate solely on absolute coordinates, so in order to actually draw anything, coordinates must be converted to absolute.

Instead the entire drawable hierarchy uses *relative* coordinates throughout, which simply means coordinates relative to the parent drawable (ie. the offset from the position of the parent). The root container of a view is always positioned at (0, 0), so the drawables added to the root container will have coordinates that are the same in both relative and absolute. So, to place an Image in (50, 50) absolute for example, it can either be given the position of (50, 50) and added to the root container, or it could be added to a different container located at, say, (30, 20) and given the coordinates (20, 30). The Drawable class provides a function that can translate its relative coordinates to absolute. This is done by traversing up the scene graph, adding the x and y values at each level until the root container is reached.

Note: relative coordinates can be negative.

Relative coordinates were chosen because they generally makes life much easier when implementing a user interface. The conversion to absolute coordinates when drawing or when handling touch input events is done entirely by the TouchGFX framework and is completely transparent to the application developer.

5.5.2.3 Size and Position

All drawables have a position expressed as X and Y in relative coordinates as explained above. In addition, all drawables have a width and a height. For built-in widgets that operate on bitmaps like Image and Button, the width and height is automatically inferred from the size of the bitmap, so only the position needs to be set. The width and height dictates the visible area of a widget. Although it would not make much sense in most cases, an Image for instance can be set to a smaller width or height than the source bitmap, which would cause it to be clipped. Containers, being drawables too, also have a width and a height which effectively act as a viewport. That is, if you add drawables to a container and position them outside the region expressed by the container's width and height, those child drawables will be clipped. Viewports can be a useful feature and are used in ScrollableContainer where the size of the scrollable container denotes the visible area and scrolling automatically being enabled when the child drawables are larger than that area.

5.5.2.4 Limitations on the number of drawables

Due to memory optimization reasons it is only possible to have 150 visible drawables on a screen at one time. Any exceeding drawables will not be shown.

5.5.3 Events

5.5.3.1 Input Events

TouchGFX supports three event types that originate from touch screen input, as well as a generic event type that can originate from other sources. These are described in the following.

A ClickEvent is issued when the touch screen detects a change in whether or not it is being pressed. Depending on the screen coordinates where the event occured, it will automatically be routed to the correct Drawable - that is, the drawable with the highest Z-order (the front-most drawable) that are both visible and touchable and are covering those coordinates. This notification will happen through a call to the Drawable::handleClickEvent function, which takes a ClickEvent object as parameter. By examining the parameter, the coordinates can be obtained. Upon the reception of the event, the coordinates have automatically been translated to the coordinate system of the drawable. That means, a click event in the top left corner of a drawable will report coordinates (0,0). The type of click event is also included in the ClickEvent object. There are three types:

- PRESSED which means that a new touch was detected at the specified coordinates.
- RELEASED which means that a touch is no longer detected.
- CANCEL which means that a touch has been dragged outside the border of the drawable.

The default implementation of handleClickEvent does nothing. Custom widgets can override this, of course. Most commonly, drawables that need to respond to click events derive from AbstractButton. A user-defined callback which will trigger if a "click" is detected (a PRESSED event followed by a RELEASED event with no intermittent CANCEL events received) can be assigned to such a subclass.

The rules for receiving click events are as follows:

- Only drawables that are *visible* (Drawable::setVisible, default true) AND *touchable* (Drawable::setTouchable, default false) are eligible for receiving touch events.
- Click events are not propagated to children of a drawable with visible set to false.
- Click events are propagated to children of a drawable with touchable set to false.
- The PRESSED event will always be issued to the highest z-order (front-most) drawable which is covering the coordinate area and satisfies the rules above. No other drawables are notified.
- The RELEASED event will always be issued to the exact same drawable that received the PRESSED event, regardless of whether the coordinates at the time of detecting the release are inside the area of that drawable. No other drawables are notified.
- The CANCEL event is likewise only sent to the drawable that received the PRESSED notification.

5.5.3.2 Drag Events

A drag event will be reported if the user moves the finger/stylus on the touch screen. Drag events will therefore always occur between a ClickEvent::PRESSED event and ClickEvent::RELEASED event. Drag events are sampled at each tick, so there will most likely be a series of drag events reported when moving the finger. A DragEvent is issued to the appropriate drawable by calling the Drawable::handleDragEvent function. In the object passed as parameter, the coordinates of where the drag began (labeled oldX/oldY) as well as where the drag ended (labeled newX/newY) can be obtained.

Receiving drag events follows the same rules as click events. One exception is that the receiving drawable is actually determined at the time of a ClickEvent::PRESSED, not each time a drag occurs.

All subsequent drag events until a ClickEvent::RELEASED is detected will be reported to the receiving drawable found at the time of the ClickEvent::PRESSED.

Drawables can easily be made draggable (following the movement of the drag event) by using the Draggable mixin.

5.5.3.3 Gesture Events

A gesture event is a more complicated event than a simple drag. Currently TouchGFX supports only one type of gesture, namely swipe.

A swipe in this context probably means what you think it means, but the actual definition in TouchGFX is that it is a drag immediately followed by releasing the touch. That is, the input mechanism is interpreted as being "in motion" when releasing, indicating that the user interface should perform some type of scrolling with a gradual deceleration. Like with drags, gesture events will happen in between ClickEvent::PRESSED and ClickEvent::RELEASED events.

A gesture is issued to the appropriate drawable by calling the function Drawable::handleGestureEvent. The default implementation does nothing. The event object will contain the coordinates of the gesture, again in the coordinate system of the receiving drawable. Additionally the gesture event object contains a velocity which, for swipes, is an indication of the length of the underlying drag that caused the gesture to fire. A small velocity can be interpreted as a "flick", whereas a higher value should likely be interpreted as a normal swipe.

The ScrollableContainer will respond to gesture events by performing a scroll relative to the reported velocity.

Gesture events will always be sent to the drawable that received the ClickEvent::PRESSED event at the time the touch screen was initially pressed, regardless of whether the coordinates where the gesture was performed intersects with that drawable.

5.5.3.4 Other Input Events

Aside from the three touch-screen based events described above, there is a generic event type called a *KeyEvent* which can be used to signal other forms of input events in an application-specific manner. Possible sources for events of this type could be physical pushbuttons, an attached keyboard or something else entirely. TouchGFX in itself does not produce, nor process any events of this type. But the Application class, from which your concrete FrontendApplication derives, has a function called handleKeyEvent, which will delegate the event to the currently active View. Thus, you can override this function and perform the appropriate actions, like displaying a dialog or notifying the presenter.

Note: A pointer to the Application class can be obtained from anywhere through the getInstance function, so any code can call the handleKeyEvent function.

5.5.3.5 Timers

Timer ticks can be useful in the UI for creating animations or displaying something for a certain amount of time. The framework provides a timer tick that can be used for these purposes in two ways, described in the following.

All containers and widgets have a handleTickEvent function, which is an empty virtual function of the Drawable base class. This function can be overridden to perform the actions necessary at each tick. For performance reasons, this function is not normally called though, unless explicitly requested. By calling Application::registerTimerWidget with a pointer to the specific Drawable, that drawable will have its handleTickEvent function called at every tick from that point on. Unsurprisingly, calling Application::unregisterTimerWidget will cause the drawable to no longer receive tick events. Up to 10 drawables can be registered at any one time.

Note: When switching to another screen, the list of drawables that receive tick events is automatically cleared (since those drawables have effectively ceased to exist).

Alternatively, the timer tick could be handled in the active View. This is useful if you want to perform actions across multiple drawables, or e.g. notify the Presenter. Not unlike the case with drawables, all views also have a handle↩TickEvent function which in the base implementation does nothing. But since there is always only one view active, this function will be called automatically at each tick without the need to register for it.

5.5.3.6 External Events

It is frequently necessary to propagate other event types to the UI. This could be, for instance, a change in battery level, current time changed, ADC sample events or messages from other RTOS tasks in the system. Usually the recommended way of doing this is to make the Model (see Model-View-Presenter) check for the event. The default Model class distributed with the examples and template will already contain a tick() function, which is called automatically every tick. You can implement the Model::tick function such that the source of your events (hardware layer, OS mailbox, ..) is polled. If an event occurs that must be propagated to the UI, simply make use of the fact that the model always has a pointer to the currently active Presenter (through the ModelListener interface), on which it can call a function matching the event. It would then be up to the active presenter to decide whether or not that event should result in UI changes, or if the event can be disregarded. For instance, a "current time changed" event would only be of interest to presenters associated with screens that actually display the current time. Other presenters can leave that particular event function unimplemented.

5.5.4 Memory Allocation

In principle you can use any memory allocation scheme you wish in TouchGFX. For convenience we have included a default memory allocation scheme that operates on the Model-View-Presenter architecture. It is somewhat complex internally, but very effective and should be relatively easy to use. The allocation scheme uses static memory allocation only - that means you will know at compile-time exactly how much RAM your user interface is going to require.

Recall from the MVP section that the user interface consists of a Model and a number of Presenter-View pairs. The Model is always accessible, so the appropriate amount of memory (sizeof(Model)) is reserved. But only one Presenter-View pair is active at a time. By use of C++ templates, the size of the biggest presenter and the biggest

view is determined at compile time, and memory for the largest presenter and the largest view is reserved. When switching to a new screen (presenter-view pair), the new presenter and view are then simply placement new'ed into these two memory blocks. The drawables for any given view are placed as members of that view, so the memory needed for them will be included in the memory reserved for the view. Aside from presenters and views, Transitions are also allocated in this manner.

The FrontendHeap is the application-specific class that defines the memory blocks needed for the UI. Since it needs to calculate the sizes of the biggest presenter, the biggest view and the biggest transition, it contains a list of the used types for each of these that needs to be maintained. That is, if you add a new presenter and a new view, you need to update the lists accordingly. If you want to understand the gritty details of how the largest type is calculated, have a look at Meta.hpp, but you will also get by fine by simply copy-pasting when inserting new entries into the lists. The following snippet is taken from the FrontendHeap class in Custom Transition Example which contains five presenter-view pairs and, incidentally, also five transitions.

```
/*
 * A list of all view types. Must end with meta::Nil.
 * @note All view types used in the application MUST be added to this list!
 */
typedef meta::TypeList< MainView,
                        meta::TypeList< LeftView,
                        meta::TypeList< RightView,
                        meta::TypeList< DownView,
                        meta::TypeList< UpView,
                        meta::Nil > > > >
                      > ViewTypes;

/*
 * A list of all presenter types. Must end with meta::Nil.
 * @note All presenter types used in the application MUST be added to this list!
 */
typedef meta::TypeList< MainPresenter,
                        meta::TypeList< LeftPresenter,
                        meta::TypeList< RightPresenter,
                        meta::TypeList< DownPresenter,
                        meta::TypeList< UpPresenter,
                        meta::Nil > > > >
                      > PresenterTypes;

/*
 * A list of all transition types. Must end with meta::Nil.
 * @note All transition types used in the application MUST be added to this list!
 */
typedef meta::TypeList< NoTransition,
                        meta::TypeList< SlideTransition<Transition::LEFT>,
                        meta::TypeList< SlideTransition<Transition::RIGHT>,
                        meta::TypeList< SlideTransition<Transition::DOWN>,
                        meta::TypeList< SlideTransition<Transition::UP>,
                        meta::Nil > > > >
                        > TransitionTypes;
```

Maintaining the list above is all you need to worry about in terms of memory allocation. But (as stated in the comments) it is **important** to keep these lists updated, otherwise horrible crashes will ensue.

Purely for reference, it can be mentioned that the member variables of FrontendHeap represents what is actually allocated:

```
Partition< PresenterTypes, 1 > presenters;
Partition< ViewTypes, 1 > views;
Partition< TransitionTypes, 1 > transitions;
FrontendApplication app;
Model model;
```

That is, three memory blocks for the presenters, views and transitions plus the actual FrontendApplication and Model objects. So by doing a sizeof(FrontendHeap) you will know how much memory your user interface requires.

5.5.5 Custom Widgets and Containers

TouchGFX provides built-in widgets and containers, such as Button, Image and Container, which can be used off-the-shelf to create a GUI (see Widgets, Containers, Mixins, Canvas Widgets for the complete list). However, an application might require some functionality or appearance that is not provided by the built-in widgets and containers. In this case, custom widgets and containers can be implemented and used to get the desired look and feel.

5.5.5.1 Tweaking Existing Widgets

One way of creating custom widgets in TouchGFX, is by changing the functionality of the built-in widgets. This is relatively straightforward since it is done by the using C++ inheritance to inherit from one of the built-in widgets. Thus one can take advantage of the possibilities offered by inheritance, such as accessing public and protected members and functions and overriding virtual functions. Almost all functions are virtual and members are protected, so the framework can be modified to a very large degree by overriding.

An example of such a custom widget can be seen in the Custom Widget Example where a custom button is implemented and used.

The code snippet below shows the header file for the custom widget *FadeButton* which is implemented in the Custom Widget Example.

```
#ifndef FADEBUTTON
#define FADEBUTTON

#include <touchgfx/widgets/Button.hpp>
using namespace touchgfx;

class FadeButton : public Button
{
public:
    FadeButton(bool initiallyVisible);

    void fadeOut();
    void fadeIn();

    virtual void handleTickEvent();

private:
    bool fadingOut;
};

#endif // FADEBUTTON
```

Notice how the *FadeButton* inherits from the Button class to get the built-in functionality of the the Button. This means that the *FadeButton* gets the behavior that can be expected by a button, such as click callbacks, the possibility to set the positions of the button etc.

The *FadeButton* then adds custom behaviour by adding the *fadeOut()* and *fadeIn()* functions and by overriding the inherited handleTickEvent() function. The implementation of *fadeOut()* and *fadeIn()* can be seen in the following code snippet, that shows how they call registerTimerWidget(Drawable∗ w) to make the framework call the handle↩TickEvent() function for every tick.

```
void FadeButton::fadeOut()
{
    fadingOut = true;
    Application::getInstance()->registerTimerWidget(this);

    // Deactivate during fade out
    setActive(false);
}

void FadeButton::fadeIn()
{
    fadingOut = false;
    Application::getInstance()->registerTimerWidget(this);
}
```

In the code snippet below it can be seen how the *handleTickEvent()* implementation alters the value of the inherited alpha member to obtain a fading effect.

```
void FadeButton::handleTickEvent()
{
    if(fadingOut)
    {
        if(alpha <= 2) // Be careful of not underrunning datatype
        {
            alpha = 0;
            Application::getInstance()->unregisterTimerWidget(this);
        }
        else
        {
```

```
            alpha -= 3;
        }
    }
    else // fading in
    {
        if(alpha >= 253) // Be careful of not overrunning datatype
        {
            alpha = 255;
            Application::getInstance()->unregisterTimerWidget(this);
            // Activate after fade in is finished
            setActive(true);
        }
        else
        {
            alpha += 3;
        }
    }
    // Force redrawing of FadeButton, since changing alpha does not yield redraw.
    invalidate();
}
```

5.5.5.2 Custom Container

Custom containers can be used to implement a custom widget by letting the custom container contain and aggregate other widgets. The custom container can then implement functions that forward calls to the widgets it contains. Imagine that we need a widget which is a normal button that also displays a text below the button image. This is a simple case of aggregating already existing functionality (as provided by Button and TextArea), so we can simply create a container with those two widgets as members:

```
#include <touchgfx/Button.h>
#include <touchgfx/TextArea.h>
#include <touchgfx/Container.h>

class MyButtonWithLabel : public Container
{
public:
  MyButtonWithLabel()
  {
    // The constructor simply adds the two widgets as children of this inherited Container
    add(btn);
    add(txt);
  }
  void setBitmaps(const Bitmap& bmpReleased, const Bitmap& bmpPressed)
  {
    btn.setBitmaps(bmpReleased, bmpPressed);

    // set the Y-coordinate of the textarea so it appears 2 pixels below the button image.
    txt.setY(bmpReleased.getHeight() + 2);
  }

  void setLabelText(TypedText t)
  {
    txt.setTypedText(t);
  }
  void setAction(SignalBase1< const AbstractButton& >& signal) { btn.setAction(signal); }
private:
  Button btn;
  TextArea txt;
};
```

Notice how *MyButtonWithLabel* inherits from Container and how it uses the inherited add(Drawable& d) function to add the widgets.

5.5.5.3 Custom Widget

If the desired look and feel or functionality cannot be achieved by one of the previously described functions, one can implement a custom widget by inheriting from the abstract Widget base class. Creating a custom widget by inheriting from Widget, provides the least amount of off-the-shelf functionality while it provides the most freedom when implementing custom widgets.

It enables one to control exactly how the the framework draws the custom widget and thereby how it looks. The implementation of how the custom widget is drawn can also affect how well drawing the widget performs, and thus the performance of the framework. Furthermore, inheriting from Widget enables the custom widget to save RAM

by leaving out members that might otherwise be unnecessarily inherited by using one of the other approaches for creating custom widgets.

When implementing a custom widget by inheriting from the Widget, it is necessary to at least implement draw and getSolidRect since these are pure virtual functions in the Drawable class. This is in contrast with deriving from Container where you need not worry about these functions as they will simply delegate the requests to the contained drawables, which will ultimately be leafs (widgets). The two functions are explained in detail in the following.

getSolidRect

The getSolidRect function exists as part of the rendering algorithm of TouchGFX in order to do effective occlusion culling where anything hidden in the background is skipped when drawing. In essence it is a contract in which the widget guarantees that the area returned by getSolidRect is a completely solid rectangle (no transparent pixels, no alpha-blending). That is, it informs the framework that it is not necessary to draw anything in the background in this area, since it will be overshadowed by this particular widget anyway. The implementation of such a function is completely indifferent to the particular Z-ordering and what other widgets might be on the screen. It simply answers the question *"If I call draw on you, what is the largest rectangular area that you are going to cover such that any background widget will not be visible?"*. It is merely an optimization – all widgets could return an empty Rect and the user interface would still look the same. The performance would be much worse though, since a user interface typically consists of several layers of widgets and without this optimization the framework would be forced to draw all the layers, including pixels that are not visible because something else covers it.

Widgets that display bitmaps can use the information automatically generated by the imageconverter for their get↩ SolidRect implementation. For example, this is the implementation of getSolidRect from the Image widget:

```
Rect Image::getSolidRect() const
{
  if (alpha < 255)
    return Rect(0,0,0,0); //If we are alpha-blending, we are not solid.

  return bitmap.getSolidRect();
}
```

For widgets that are not based on bitmaps but instead draw lines or individual pixels, getSolidRect can just return an empty area. This is typically not a problem because the amount of pixels drawn in this manner is likely to be low. For example, a TextArea draws only a fraction of the pixels actually covered by the width and height of the widget. So it returns simply an empty area, meaning the entire area covered by this widget will also be drawn by the background widget, and the pixels representing the actual text will therefore be drawn twice.

draw

The draw(const Rect& invalidatedArea) const function is what actually makes something appear on the screen by manipulating the pixels in the frame buffer. This is typicall done through the LCD interface, which contains functions for drawing bitmaps, texts, rectangles and lines. It is also possible, through the lockFrameBuffer function in H↩ AL, to obtain a pointer directly to the actual frame buffer itself, allowing direct manipulation of pixels. There are two important things to note regarding the draw function. It is *const*, which means that it can not modify member variables. This is by design because the framework might choose to call draw on a widget in several partial steps (or not at all). Therefore any assumptions about when and how the draw function is called are likely to be false, so updating state information inside this function would often lead to unexpected results.

The second thing to consider is the argument *invalidatedArea* passed to the draw function. This is a rectangle describing the subregion of the widget to draw. If the entire widget is to be drawn, the invalidatedArea will be (0, 0, width of widget, height of widget). The implementation of draw *must* respect the invalidatedArea by only drawing inside that area. If a draw function draws outside that area it will result in visible errors in Z-order on the display, even if the pixels being drawn are still inside the area of the widget. As an example, here is the draw function of Image:

```
void Image::draw(const Rect& invalidatedArea) const
{
    Rect meAbs;
    translateRectToAbsolute(meAbs);
    Rect dirtyBitmapArea = bitmap.getRect() & invalidatedArea; // operator& overloaded to mean intersection
       of two areas
    LCD::drawPartialBitmap(bitmap, meAbs.x, meAbs.y, dirtyBitmapArea, alpha);
}
```

Notice that a dirtyBitmapArea is calculated as the intersection of the bitmap area with the invalidated area, and only

that result is being passed to drawPartialBitmap.

5.5.6 Canvas Widgets

Canvas Widgets and the Canvas Widget Renderer are a powerful and versatile add-on to TouchGFX which provides nice smooth, anti-aliased drawing of geometric shapes using relatively little memory while maintaining high performance. However, rendering geometrical shapes must be seen as a quite expensive operation and can easily strain the microcontrollers resources if not used carefully.

The Canvas Widget Renderer (hereafter referred to as CWR) is a general graphics engine providing optimized drawing primitives, automatically eliminating most superfluous drawings. CWR is used by TouchGFX for drawing complex geometric shapes. Geometric shapes are defined by Canvas Widgets. TouchGFX comes with a number of supported Canvas Widgets but just like normal widgets you can make your own custom Canvas Widget to match your needs. Where a Canvas Widget defines the geometric shape of a figure to be drawn by the CWR, the actual color of each pixel inside the figure is defined by an associated Painter class. Again, TouchGFX comes with a number of Painters but you can make your own custom Painters to match your needs.

See CanvasWidget Example for an example.

Figure 5.5: Example of Canvas Widget objects

5.5.6.1 Using Canvas Widgets

Other widgets in TouchGFX have their sizes set automatically. A bitmap widget, for example, will automatically get the width and height of the contained bitmap. It is therefore enough use setXY() on the bitmap widget to place the bitmap on the display.

Canvas Widgets do not have a default size which can be determined automatically and set initially. Care must be taken to not only position, but also size the widget correctly, otherwise the width and height of the Canvas Widget will be to zero, and nothing will be drawn on the display.

So, instead of using *setXY()*, use setPosition() to place and size the canvas widget. See also Custom Canvas Widgets below for an example on how to create and use a custom canvas widget.

Once the position and size of the Canvas Widget has been set, a geometrical shape can be drawn inside it. The coordinate system will have (0, 0) in the upper left corner of the widget (*not* the display), the X axis stretches to the right and the Y axis stretches downwards. See The Coordinate System for further information.

5.5.6.2 Memory allocation and usage

To produce nice anti-aliased complex geometrical shapes requires additional memory. For this CWR has to have a special allocated memory buffer that is used during rendering. CWR, as the rest of TouchGFX, has no dynamic memory allocation.

The memory buffer should be allocated and setup in `target/main.cpp` (and `simulator/main.cpp`) as follows – here are snippets from `target/main.cpp`. See `app/example/canvas_widget_example` for further details.

```
#define CANVAS_BUFFER_SIZE (3600)
```

Is defined in the beginning of main.cpp and the following code in main():

```
// Setup the CanvasWidgetRenderer. ONLY needed if you use CanvasWidgets
// in your application. The CANVAS_BUFFER_SIZE can be adjusted to match
// your needs in performance vs. RAM usage. Read more on this in the
// TouchGFX Manual.
static uint8_t canvasBuffer[CANVAS_BUFFER_SIZE];
CanvasWidgetRenderer::setupBuffer(canvasBuffer, CANVAS_BUFFER_SIZE);
```

The amount of CWR memory needed depends on the maximum size of the shapes that are to be drawn in the application. You can, however, reserve less memory than the maximum shape requires. To handle this situation the CWR split up drawings of shapes into smaller frame buffer parts resulting in slightly longer rendering time, as shapes in these cases will sometimes have to be rendered more than once. It is possible to investigate and fine-tune the memory consumption closer when running in simulator mode.

Note

Fine tuning the memory buffer size can easily be done from your program by a call to *touchgfx::CanvasWidgetRenderer::getUsedBufferSize()* and *touchgfx::CanvasWidgetRenderer::getMissingBufferSize()*. The sum of these two values is the maximum amount of memory that was required by CWR for the specific execution of the application. This number is easy to monitor with the simulator by simply changing

```
#define CWR_OUTPUT 0
```

to

```
#define CWR_OUTPUT 1
```

in *touchgfx::CanvasWidgetRenderer* "CanvasWidgetRenderer.hpp". Now updates to the CWR memory requirements will be printed in the console window.

Setting the correct memory buffer size is a trade off between memory and performance (rendering time). A good starting value is usually around 3000, but using the above technique, a better value can often be determined. If the shape is too complex and the allocated memory buffer is too small, part of the shape will not be drawn (some vertical pixel lines will be skipped) and it is possible that nothing is drawn at all. In any case rendering time will increase a lot.

If the memory buffer is far too small, CWR will fail. If this happens in debug mode CWR will assert and in release mode CWR will not draw on the display (since the buffers are setup for each draw request, it is possible that sometimes nothing is draw and sometimes parts of the geometrical shapes are drawn with some one or more vertical lines missing).

5.5.6.3 The Coordinate System

The coordinate system in TouchGFX is normally used to address pixels for positioning bitmaps on the display. Bitmaps, texts and other graphic elements are all placed in a coordinate system, where (0,0) is the upper left hand pixel, the x-axis extends to the right and the y-axis extends downwards. In CWR it is not enough to be able to address pixels using integers, though this might be enough in special cases, this is far from enough in general. To demonstrate this, consider a circle with line width 1, which must fit precisely inside a box of 5 by 5 pixels. The center

of this circle must be at (2.5, 2.5) and the radius must be 2, so fractions are required for the center coordinate. Similarly, if the circle should fit inside a box of 6 by 6 pixels, the center must be at (3, 3) and the radius must be 2.5, so here fractions are required for the radius.

This new way of addressing coordinates for drawing graphics, means that the center of the pixel at (0,0) has CWR coordinate (0.5, 0.5). Hence, the box containing the pixel in the upper left corner of the screen has the following outline: (0,0) -> (1,0) -> (1,1) -> (0,1) -> (0,0).

Though this might seem confusing at first, it quickly becomes very natural. Where the coordinate system for bitmaps address the pixels, the same coordinate for Canvas Widgets address the *gap* just before and above the pixel.

5.5.6.4 Custom Canvas Widgets

Implementing a custom Canvas Widget requires an implementation of a new class with the following functions:

```
virtual bool drawCanvasWidget(const Rect& invalidatedArea) const;
virtual Rect getMinimalRect() const;
```

The `drawCanvasWidget()` must draw whatever the custom widget needs to draw and `getMinimalRect()` should return the actual rectangle in the Widget which contains the geometrical shape.

Note

> The reason for having `getMinimalRect()` is that a geometrical shape can be moved around inside its widget and it is often impractical to resize and reposition the widget whenever the shape changes to only invalidate the smallest possible area.

A dummy implementation of `getMinimalRect()` could simply `"return rect;"`, that is the size of the widget, but that would cause the entire area covered by the canvas widget to be redrawn, and not just the part of the canvas widget containing the getometrical shape. Very often, the geometrical shape occupies only a small part of the canvas widget, for example the hands of a clock as can be seen in the touchgfx_2014 demo.

Canvas Widgets all use the Canvas class, which encapsulates the Canvas Widget Renderer as described above. CWR has many optimizations applied automatically, though awareness of your geometrical shape in relation to the invalidated area, and avoiding unnecessary drawing outside the invalidated area, is always a good way to boost performance.

A rough implementation of a diamond shaped square inside a 10x10 box could look something like this:

```
class Box10x10 : public CanvasWidget
{
public:
  virtual Rect getMinimalRect() const
  {
    return Rect(0,0,10,10);
  }
  virtual bool drawCanvasWidget(const Rect& invalidatedArea) const
  {
    Canvas canvas(this, invalidatedArea);
    canvas.moveTo(5,0);
    canvas.lineTo(10,5);
    canvas.lineTo(5,10);
    canvas.lineTo(0,5);
    return canvas.render(); // Shape is automatically closed
  }
};
```

Note

> Again, be careful that `getMinimalRect()` returns to correct rectangle, or the graphics on screen might be wrong.

In order to see the Box10x10 on the display, the color must be set using Box10x10::setPainter() inherited from CanvasWidget. Also, the Box10x10 must be placed and sized correctly. This could look similar to this:

In the header file declare

```
Box10x10 box;
PainterRGB565 myPainter;
```

and in the code you should have something like this:

```
myPainter.setColor(Color::getColorFrom24BitRGB(0xFF, 0x0, 0x0));
box.setPosition(100,100,10,10);
box.setPainter(myPainter);
add(box);
```

The color of the Box10x10 is determined by the Painter currently set on the box object and is handled entirely by the super class. Read more about Painters below.

See Canvas Widgets for all supported Canvas Widgets.

5.5.6.5 Painters

A Painter defines a coloring scheme to fill a Canvas Widget object. TouchGFX comes with a set of predefined painter classes, but custom painters can easily be implemented.

In order to implement a custom Painter, care must be taken to never write outside the frame buffer. Such a bug in a custom Painter can result in serious crashes. Here is an example of a custom Painter which we will use to paint an object red, only function renderNext() needs to be implemented. See AbstractPainter for more information.

```
class Red : public AbstractPainterRGB565
{
public:
    virtual bool renderNext(uint8_t &red, uint8_t &green, uint8_t &blue, uint8_t &alpha)
    {
        red = 0xFF;
        green = 0x00;
        blue = 0x00;
        alpha = 0xFF;
    }
};
```

To paint the box object from above red, put this in the header file:

```
Box10x10 box;
Red redPaint;
```

and put this in the code:

```
box.setPosition(100,100,10,10);
box.setPainter(redPaint);
add(box);
```

Please note that it is possible to override more methods to create special painters (e.g. renderInit()), however, TouchGFX has some generic painters which covers most uses, see Painters for all supported Painters.

5.5.7 Mixins

When developing a TouchGFX applications it is sometimes necessary to apply a particular feature to a lot of different widgets and containers e.g. the ability to get dragged around the screen. A useful method for doing this is to use mixin classes.

A mixin is a fragment of a class in the sense that it is intended to be composed with other classes or mixins and not intended for standalone use.

TouchGFX comes with a set of mixin classes that can be used together with any widget and container. One example is the Draggable mixin class.

```
template<class T>
class Draggable : public T
{
public:
    Draggable() : T()
    {
        T::setTouchable(true);
    }

    virtual ~Draggable() { }

    virtual void handleDragEvent(const DragEvent& evt)
    {
        T::handleDragEvent(evt);
        T::moveRelative(evt.getDeltaX(), evt.getDeltaY());
    }

private:

};
```

The only thing the mixin does is to ensure that the template element is touchable and that its position is updated when receiving drag events. Now if you simply declare your widget or container as the template argument for the mixin class it will receive the new feature from the mixin class.

An example of how to declare your image as draggable:

```
Draggable<Image> myDraggableImage;
```

See Drag Example for usage of the touchgfx::Snapper and touchgfx::Draggable mixins.

5.5.8 Using Text and Fonts

Displaying texts is a built in capability of TouchGFX. A TypedText in TouchGFX is a combined entity of the text contents itself and the typography of the text. The typography contains, the font and font size of the text and the bits per pixel (bpp) used in anti aliasing the glyphs of the font.

TypedTexts are specified in the `assets/texts/texts.xlsx` Excel file. The texts.xlsx file supports specifying texts for any language needed. For a complete specification read Text Converter.

Text Example illustrates the key aspects of text displaying in TouchGFX.

5.5.8.1 Using Texts in Code

In order to display a TypedText on the screen we use the TextArea widget.

```
// in the .hpp file
TextArea txt;

// in the .cpp file
txt.setTypedText(TypedText(T_MAIN_TEXT));
txt.setPosition(...);
txt.setColor(...)
...
add(txt);
```

We set the TypedText of the TextArea and set the position and color. Finally the TextArea is added to the current view.

5.5.8.2 Wildcards

TypedTexts can contain "wildcards", making texts with dynamic content possible. This is specified in the texts.xlsx file by using the special syntax <...>. This corresponds closely to having a %s in a regular string in C++.

Using wildcard text in an application is done by managing and updating a local touchgfx::Unicode::UnicodeChar array.

```
// in the .hpp file
TextAreaWithOneWildcard txt;
Unicode::UnicodeChar txtBuffer[10];

// in the .cpp file
...
txt.setWildcard(txtBuffer);

// updating the value
Unicode::snprintf(txtBuffer, 10, "%d", 42);
txt.invalidate();
```

See Text Example for a complete example.

Note that you have to make sure that all characters used in a wildcard are represented in the texts.xlsx file for that wildcards specific typography. As an example if you need to display digits, make sure that characters corresponding to the digits 0 to 9 are present in the texts.xlsx file (for that typography).

Also note that two consecutive `%` chars will be replaced by one `%` char in wildcard strings.

5.5.8.3 Switching Language

TouchGFX supports multi language interfaces. The current language used in the interface can be changed by

```
Texts::setLanguage(...);
```

5.5.8.4 Conversion Process

Texts are automatically transformed from Excel (.xlsx) to an internal .cpp format. Likewise fonts are transformed from TrueType/OpenType/Glyph Bitmap Distribution Format (.ttf/.otf/.bdf) to an internal .cpp format. This is done when invoking make or make assets and utilizes the supplied textconvert and fontconvert tools. See Assets and Tools for more information.

The texts located in the assets/texts folder are converted and placed in the generated/texts folder. The fonts located in the assets/fonts folder are converted and placed in the generated/fonts folder.

The generated texts are all inserted into a database. The texts are named after the keys in the Excel file and can be found in the generated file `generated/texts/include/texts/TextKeysAndLanguages.hpp`.

All a developer has to worry about is placing the needed texts in the `assets/texts/texts.xlsx` file. The rest will be handled by the TouchGFX tooling.

Note: Some Glyph Bitmap Distribution Format (.bdf) files have been seen to contain lines with leading spaces causing problems for the font convert utility. The .bdf font file can simply be edited using any text editor and those leading spaces can be removed thereby correcting the .bdf font file.

5.5.9 Using Images

Images are essential since a lot of the graphical elements in a typical TouchGFX application are based on images. The following sections provides an overview of how an image is represented in TouchGFX and how it is used.

5.5.9.1 Using Images in Code

The Bitmap class is a proxy for images stored in the application specific bitmap database that holds the auto generated images. The Bitmap can be used to build a GUI by using it with the following classes:

- Image
- AnimatedImage
- Button
- ToggleButton

A Bitmap object is instantiated by passing the constructor an ID that identifies the bitmap in the bitmap database the Bitmap should be a proxy for.

Image is one of the classes using the Bitmap. To use an Image, and thus a Bitmap, it is necessary to include the following:

```
#include <touchgfx/widgets/Image.hpp>
#include <BitmapDatabase.hpp>
```

Assuming that the BitmapDatabase contains the ID BITMAP_BACKGROUND_ID, it is possible to use this ID to create an Image as shown below:

```
Image someImage(Bitmap(BITMAP_BACKGROUND_ID));
```

When *someImage* is drawn by the framework, it will now show the bitmap image that BITMAP_BACKGROUND_ID identifies.

TouchGFX supports referencing the same bitmap multiple times and thus it is valid to do the following:

```
Image someImage(Bitmap(BITMAP_BACKGROUND_ID));
Image anotherImage(Bitmap(BITMAP_BACKGROUND_ID));
```

As described in Image Converter images may include alpha blended pixels. Furthermore it is possible to dynamically set a transparency level, the alpha value, of an entire image when using images.

```
Image image;
...
image.setAlpha(123);
```

5.5.9.2 The Conversion Process

Images are automatically transformed from one of the supported image formats to the internal 16 bit cpp representation. This is done when building in Visual Studio, invoking make or make assets and utilizes the supplied imageconvert tool. For detailed information see Image Converter.

The generated bitmaps are all inserted into a database (`generated/images/include/Bitmap↩Database.hpp`). The bitmaps are named after the input image files. As an example the assets/images/background.↩bmp file is converted to the BITMAP_BACKGROUND_ID bitmap. Another example is assets/images/icons/open.↩png being converted to BITMAP_OPEN_ID bitmap. Note that the directory name 'icons' is not included in the name of the generated bitmap.

As a consequence all a developer has to worry about is placing the needed image files in the assets/images folder. The rest will be handled by the TouchGFX tooling.

5.5.10 Using Callbacks

A Callback in TouchGFX is essentially a function pointer wrapper which enables the registration of a function to be called when something happens. For instance, the Button standard widget can be configured to call a function when the button is clicked. Naturally the callback mechanism can also be used for custom widgets in the application.

The reason TouchGFX does not use simple function pointers is that we want the function to be called to be able to reside as a member function of a user-created class. In C++ pointers to member functions require knowledge of the class from the caller, which is problematic for standard widgets. For instance, consider that a standard Button needs to invoke a member function of your custom View subclass, of which it has no knowledge.

To deal with this, a template-based abstraction has been implemented. It is based on two parts: a GenericCallback placed in the caller (e.g. Button), and a Callback placed in the callee (e.g. your custom view). The GenericCallback does not require knowledge of the class type of the member function, meaning it can be used in standard widgets. Callbacks can be configured to call functions of zero, one or two parameters.

The syntax of callbacks is a bit cumbersome but the added value that the function being called can be placed in a member function usually makes it worthwhile. For instance, a button clicked handler will very often need to manipulate other widgets in the View or notify the Presenter, both of which requires access to other View member variables. Using a normal function pointer to a global function instead of a Callback will require some other mechanism of manipulating the view from outside. You are of course free to employ whatever callback mechanism you wish in your custom widgets and views.

The following is an example of how to declare the two portions of a callback and glue them together:

```
#include <touchgfx/Callback.hpp>
// This is a widget capable of firing an event. You can register a callback to catch the event.
class MyWidget : public Widget
{
public:
  MyWidget() : action(0) {} //Make sure to initialize listener pointer to zero
  // This function registers the callback
  void setAction(GenericCallback<>& callback) { action = &callback; }

  // This function calls the callback, if one is registered.
  void fire()
  {
    if (action && action->isValid())
    {
      // Call registered function
      action->execute();
    }
  }
private:
  //An untyped pointer to the typed Callback located in callee
  GenericCallback<>* action;
};
```

On the receiver side, where the function to be called is located, it looks like this:

```
#include <touchgfx/Callback.hpp>
class MyView : public View
{
public:
  MyView() :
    cb(this, &MyView::handleEvent) // Initialize callback with pointer to object containing member
       function, and address of member function.
  {
    myWidget.setAction(cb); //Register callback
  }
  void handleEvent()
  {
    //This function gets called when event is fired from MyWidget
  }
private:
  MyWidget myWidget;
  Callback<MyView> cb; //This callback is for a function placed in MyView
};
```

As mentioned, callbacks are also defined for calling functions with one or two parameters. An example where the function takes an integer argument is as follows:

```
  GenericCallback<int>* action;
  void setAction(GenericCallback<int>& callback) { action = &callback; }
  void fire(int eventCode)
  {
    if (action && action->isValid())
    {
      action->execute(eventCode);
    }
  }

...

  MyView() : cb(this, &handleEvent) {}
  Callback<MyView, int> cb;
  void handleEvent(int value) {}
  myWidget.setAction(cb);
```

5.6 Widgets, Containers, Mixins, Canvas Widgets

The TouchGFX distribution provides a set of built-in widgets, containers, mixins, Canvas Widgets and Painters. The complete set is:

5.6.1 Widgets

- AbstractButton
- AnimatedImage
- Box
- Button
- ButtonWithIcon
- ButtonWithLabel
- Image
- Keyboard
- RadioButton
- RadioButtonGroup
- ScalableImage
- SnapshotWidget
- TextArea
- TextAreaWithOneWildcard
- TextAreaWithTwoWildcards
- TextureMapper
- ToggleButton
- TouchArea

5.6.2 Containers

- Container
- ListLayout
- ScrollableContainer
- Slider
- ZoomAnimationImage

5.6.3 Mixins

- ClickListener
- Draggable
- FadeAnimator
- MoveAnimator
- PreRenderable
- Snapper

To read more on extending or creating widget, containers and mixins please read Custom Widgets and Containers.

5.6.4 Canvas Widgets

- Circle
- Line
- Shape

5.6.5 Painters

- PainterBW
- PainterRGB565
- PainterRGB565Bitmap

To read more on extending or creating canvas widgets and painters please read Canvas Widgets.

More widgets, containers, and mixins can be found in the TouchGFX Open Repository (see TouchGFX Open Repository).

5.7 TouchGFX Open Repository

TouchGFX includes an open source community that shares custom widget, containers, and other components that can be used free of charge either directly in your application or as inspiration. The source code is directly accessible from the GitHub `web page`.

The TouchGFX Open Repository can be found `here`.

The TouchGFX team would like to encourage you to use the repository and contribute with your own great ideas and widgets whenever possible.

5.8 Miscellaneous

5.8.1 Screenshots

The simulator supports taking screenshots of the current screen. Use the F3 key to do this. The screenshots will be saved in the `screenshots/` folder.

Chapter 6

Examples

This page describes the examples included in the TouchGFX distribution located under the folder `app/example/`. To compile and run each example please see Build and Execute. The examples will run on both PC simulator (Windows and Linux) and on selected supported demo boards.

6.1 Animation Example

The Animation Example shows how to display, start and stop an animation via a range of images. The example shows how to configure the animation to a particular frame rate and how to reverse the order of the animation.

Figure 6.1: The animation in progress

6.1.1 Framework Elements Used

- AnimatedImage
- ButtonWithLabel
- Image
- Callback

6.1.2 Example Location

`app/example/animation_example/`

6.2 Arabic Text Example

The Arabic Text Example shows how to use the Right-to-Left language option to display texts in arabic. The example also shows how to mix Right-to-Left and Left-to-Right texts in one text.

Figure 6.2: Displaying Arabic Text

6.2.1 Framework Elements Used

- TextArea
- TextAreaWithOneWildcard
- ButtonWithLabel
- Image
- Callback

6.2.2 Example Location

`app/example/arabic_text_example/`

6.3 Button Example

The Button Example shows how to handle clicks of a Button. This is done by attaching a Callback to the button. The example shows how to reuse a callback between multiple buttons. And further shows how to interact with the presenter. A bit of logic to enable and disable buttons is done by the presenter.

Figure 6.3: After a few clicks of the up button

6.3.1 Framework Elements Used

- Image
- Button
- TextAreaWithOneWildcard
- Callback

6.3.2 Example Location

`app/example/button_example/`

6.4 CanvasWidget Example

The Canvas Widget Example shows how to draw and update geometric shapes using some of the build in Canvas Widgets.

Figure 6.4: Canvas Widgets in action

6.4.1 Framework Elements Used

- Circle
- Line
- Shape
- CanvasWidgetRenderer

6.4.2 Example Location

`app/example/canvas_widget_example/`

6.5 Custom Transition Example

The Custom Transition example shows how to implement custom transitions and how these are used. The visual effect of the transition implemented in this example, resembles two doors being closed and opened from both sides of the screen. Both doors closes to hide the screen the transition is coming from. The doors then open again to reveal the screen the transition is going to. The doors are implemented by moving Images across the screen. A SnapshotWidget is used to capture a snapshot of the screen it transitions away from, which is used when the doors are closing.

Figure 6.5: Snapshot taken while doors are closing

6.5.1 Framework Elements Used

- Transition
- SnapshotWidget
- Image
- Callback
- ButtonWithLabel

6.5.2 Example Location

`app/example/custom_transition_example/`

6.6 Custom Widget Example

The Custom Widget example shows how to implement custom widgets. In the example a Fade Button is created. A Fade Button will fade it self in or out using alpha transparency. This is done by registering for timer events when the Fade Button is clicked. The Fade Button will update its alpha transparency when handling timer events.

Figure 6.6: One button fading out, the other fading in

6.6.1 Framework Elements Used

- Image

- Callback

6.6.2 Example Location

`app/example/custom_widget_example/`

6.7 Display Orientation Example

The Display Orientation Example illustrates how to make dynamically display orientation changes. When pressing the rotate button, the orientation is changed and the view reinitialized.

Figure 6.7: View in landscape mode

Figure 6.8: View in portrait mode

6.7.1 Framework Elements Used

- Button
- Image
- HAL::setDisplayOrientation

6.7.2 Example Location

`app/example/display_orientation_example/`

6.8 Drag Example

The Drag Example illustrates a way to create draggable elements. This is, in this case, done by mixing the Snapper mix-in into a Button. The snapper will set the button draggable and receive callbacks when the button is dragged. When clicked the snapper will display a highlighted image to give visual feedback that a drag is taking place. When the drag is ended the snapper receives another callback and can decide whether to snap to a new position or not, displaying the appropriate element at the appropriate location. The Drag Example comes in both a 16 and a 24 bit version.

Figure 6.9: The element is dragged over the drop zone

6.8.1 Framework Elements Used

- Snapper
- Button
- Callback
- DragEvent
- Image

6.8.2 Example Location

`app/example/drag_example/`

6.9 Dynamic Bitmap Example

The Dynamic Bitmap Example shows how to access bitmap images stored on the SD card on the STM32F746 Discovery board and how to transfer these to the TouchGFX bitmap cache. Note that the Dynamic Bitmap Example only runs on the ST STM32F746 Discovery board.

Figure 6.10: Displaying an image that has been cached from the SD card

6.9.1 Framework Elements Used

- Image
- ScalableImage

6.9.2 Example Location

`app/example/dynamic_bitmap_example/`

6.10 List Layout Example

The List Layout Example shows how to use the ListLayout container to easily organize elements in a list. It uses a custom container to handle the layout of each list element. The example also shows how to propagate events, in this case button click event, to the view from inside the list.

Figure 6.11: List is being displayed and ready for scrolling

6.10.1 Framework Elements Used

- ListLayout
- Container
- ScrollableContainer
- Image
- Button
- Callback

6.10.2 Example Location

`app/example/list_layout_example/`

6.11 Monochrome Example

This is a very simple example, showing TouchGFX configured for 1 bit per pixel (monochrome) graphics. As such it does not present anything new in terms of framework elements, but it can serve as an example of how to configure your application for using 1BPP graphics.

Figure 6.12: Monochrome application

6.11.1 Framework Elements Used

- Box
- Button
- TextArea

6.11.2 Example Location

`app/example/monochrome_example/`

6.12 Move and Fade Example

The MoveFadeExample shows how to use the two mixin classes MoveAnimator and ZoomAnimator. This example is a good example of how to use mixins and how to react on callback.

Figure 6.13: Move and Fade Animation example 1

Figure 6.14: Move and Fade Animation example 2

6.12.1 Framework Elements Used

- MoveAnimator
- FadeAnimator
- Image
- Box

6.12.2 Example Location

`app/example/move_fade_example/`

6.13 Keyboard Example

The Keyboard Example shows how to implement an on screen keyboard by using the Keyboard widget. The keyboard implemented in this example, has both alpha and numeric keys, as well as a backspace key, a space key and a caps lock key.

The example covers how keyboard layouts are created and used to define the visual appearance of the keyboard. Furthermore, it covers how Keyboard mappings are set up, and how these can be changed at runtime to change the keys displayed on the keyboard. The implementation of custom keys, such as caps lock and backspace are also covered in the example.

Figure 6.15: The keyboard with the numeric key mapping enabled.

6.13.1 Framework Elements Used

- Keyboard
- KeyMapping
- Layout
- Callback
- Container
- TextArea

6.13.2 Example Location

`app/example/keyboard_example/`

6.14 Pixel Data Example

The Pixel Data Example shows how to use the PixelDataWidget to display a runtime calculated buffer of pixel data.

Figure 6.16: Pixel Data Example.

6.14.1 Framework Elements Used

- PixelDataWidget

6.14.2 Example Location

`app/example/pixel_data_example/`

6.15 Radio Button Example

The Radio Button Example shows how to group RadioButtons together in a RadioButtonGroup. It also shows how to react on selected and deselected events.

Figure 6.17: Radio Button Example.

6.15.1 Framework Elements Used

- RadioButton

- RadioButtonGroup

6.15.2 Example Location

`app/example/radio_button_example/`

6.16 Scale and Zoom Example

The scale and zoom example shows an example use of how to ScalableImage and ZoomAnimationImage.

The example shows two ScalableImages that are scaled down to 76x76 pixels from a 200x200 pixels bitmap.

The top image is rendered with the bilinear interpolation ScalingAlgorithm the bottom image the nearest neighbor ScalingAlgorithm. To the right is a ZoomAnimationImage that is zoomed and animated across the background.

Figure 6.18: ScalableImages and a ZoomAnimationImage

Figure 6.19: ScalableImages and a ZoomAnimationImage

6.16.1 Framework Elements Used

- ScalableImage

- ZoomAnimationImage

6.16.2 Example Location

`app/example/scale_zoom_example/`

6.17 Scrollable Container Example

The Scrollable Container Example shows an example use of a ScrollableContainer. The container is filled with text via a TextArea. The text is too large to fit the container, so the container is made scrollable. This is done automatically by the framework. Scrollbars are displayed appropriately.

Figure 6.20: The text is being scrolled

6.17.1 Framework Elements Used

- ScrollableContainer
- Image
- TextArea

6.17.2 Example Location

`app/example/scrollable_container_example/`

6.18 Slider Example

The Slider Example shows how to use both horizontal and vertical sliders. The sliders controls the transparency levels of the images. The vertical slider changes the value on release whereas the horizontal slider changes it at each slider drag and click.

Figure 6.21: Using sliders in TouchGFX

6.18.1 Framework Elements Used

- Slider

- Image

6.18.2 Example Location

`app/example/slider_example/`

6.19 Text Example

The Text Example shows how to use and display texts. Texts are specified in the assets/texts/texts.xlsx file. These texts along with the fonts in the assets/fonts directory are automatically transformed into resources in the generated directory.

Texts have a font, size, alignment and color. Texts can be either static or dynamic. Dynamic texts have wildcards corresponding to %s in regular C++ strings.

The example shows a static text and a dynamic text.

Figure 6.22: The static headline and content and the dynamic counter text

6.19.1 Framework Elements Used

- TextArea
- TextAreaWithOneWildcard
- ButtonWithLabel
- Image
- Callback

6.19.2 Example Location

`app/example/text_example/`

6.20 Text Rotate Example

The Text Rotate Example shows how to write text in one of four directions. Text is normally written left to right, being rotated 0 degrees, but it is possible to rotate the text by 90, 180 or 270 degrees at runtime, which implies that the text direction is not specified in the assets/texts/texts.xlsx file, but is set using a function call at runtime.

The example shows a static text rotated 90 degrees counterclockwise.

Figure 6.23: The text can be written rotated

6.20.1 Framework Elements Used

- TextArea
- ButtonWithLabel
- Image
- Callback

6.20.2 Example Location

`app/example/text_rotate_example/`

6.21 Texture Mapper Example

The Texture Mapper Example shows how to easily rotate an image around both the X, Y and Z axises.

Figure 6.24: Rotating images in both 3D and 2D using the Texture Mapper

6.21.1 Framework Elements Used

- TextureMapper

- Image

6.21.2 Example Location

`app/example/texture_mapper_example/`

6.22 Transition Example

The Transition Example shows how to use transitions. Transitions are the mechanism used for changing between screens.

The example uses SlideTransition. The SlideTransition is one of the built in transitions.

The transition will grab a snapshot of the screen initially and then perform a soft horizontal or vertical slide.

In the example, the snapshot slides off to one side while the new screen slides in.

Figure 6.25: The sliding change from one screen to the other

6.22.1 Framework Elements Used

- SlideTransition
- MVPApplication
- Image
- Button
- Callback

6.22.2 Example Location

`app/example/transition_example/`

Chapter 7

Demos

This page describes the demos included in the TouchGFX distribution located under `app/demo/`. The demos will run on both PC simulator (Windows and Linux) and on selected demo boards that support the demos resolution. Source code for all demos can be found in the `app/demo/` folder, for each application respectively.

7.1 Compiling and running

The demos all support Make and Visual Studio build for the PC simulator and IAR, Keil or GCC builds for selected evaluation boards. This is done in the usual way, see Build and Execute for more information.

7.2 Demo 2015 (5.7" - 640x480)

The TouchGFX demo 2015 5.7" (`app/demo/touchgfx_demo2015`) is a quite extensive demo showing some of the graphical possibilities that TouchGFX offers.

The full source code and all image files are included in the release and should serve as inspiration and as an introduction on how to make complex graphical elements and views. The demo has a carousel menu that lets you choose between five separate independent demos. The carousel menu can be changed to a menu structure with animating boxes.

Note

Since the focus of this demo is to demonstrate how to make complex graphics, most of the screen do not use data from the model class, as would a real application. Instead, they mostly use hard-coded values as input.

Figure 7.1: Carousel menu (Demo 2015)

Figure 7.2: Alpha blending demo screen (Demo 2015)

7.3 Demo 2015 (4.3" - 480x272)

The TouchGFX demo 2015 4.3"(`app/demo/touchgfx_demo2015_480x272_8MB`) shows some of the graphical possibilities that TouchGFX offers while running on very small devices. This demo is designed to run on the lower range of Cortex M3/M4 e.g. 120 MHz NXP LPC4088.

The demo consist of a full screen menu that the user can swipe through and choose one of the small independent demos.

Figure 7.3: swipe menu (Demo 2015 480x272)

Figure 7.4: Graph demo screen (Demo 2015 480x272)

Figure 7.5: Text and language demo screen (Demo 2015 480x272)

7.4 Demo 2014 (4.3" - 480x272)

The TouchGFX demo 2014 (`app/demo/touchgfx_demo2014_480x272`) is a quite extensive demo showing some of the graphical possibilities that TouchGFX offers.

The full source code and all image files are included in the release and should serve as inspiration and as an introduction on how to make complex graphical elements and views. The demo is divided into many small screens that each contain a separate independent demo. Each demo can be selected from the main page, which contains a scrollable menu.

Note

Since the focus of this demo is to demonstrate how to make complex graphics, most of the screen do not use data from the model class, as would a real application. Instead, they mostly use hard-coded values as input. One exception to this is the HomeAutomation screen. The *HomeAutomationView* uses the presenter and model in the correct way. The data is, of course, still hardcoded (in the model) since the demo naturally does not have any real input or output.

Figure 7.6: The List Navigation screen (Demo 2014)

Figure 7.7: The Animated Graphics screen (Demo 2014)

Figure 7.8: The Home Automation screen (Demo 2014)

7.5 Demo 2014 small (2.4" - 240x320)

The TouchGFX demo 2014 small (`app/demo/touchgfx_demo2014_240x320`) is a reduces version of demo 2014. It is custom made for the ST STM-32F429 Discovery board. The resolution is QVGS (320x240) and only internal flash is used for storing image date (2 MB).

Figure 7.9: The menu screen (Demo 2014 small)

One part of the demo shows the effect of TouchGFX in combination with the ST Chrom Art acceleration of the STM32F429 MCU. The result is a very low MCU load while performing advanced alpha blended animations.

Figure 7.10: The ST chrom art screen (Demo 2014 small)

7.6 Home Control Demo (7.0" - 800x480)

The Home Control Demo is a 7.0" demo designed to show some of the capabilities of TouchGFX on a large display. Contrary to other demos described on this page, this demo uses the `Model` class, as it was intended, to store state between screens.

Figure 7.11: Entertainment screen (Home Control Demo)

Figure 7.12: Main screen (Home Control Demo)

Chapter 8

Porting Guide

This page details the steps involved to port TouchGFX to a custom hardware platform. Understanding and following the porting procedure detailed in this guide will allow you to run TouchGFX applications on hardware platforms other than the ones immediately supported by the TouchGFX distribution. We recommend that you understand the following advanced topics before proceeding:

- Timing and TFT Controller Integration
  - Rendering Pass
  - Frame Rate
  - Number of Frame Buffers
- Blitting Operations and Hardware Blit Capabilities

The hardware abstraction layer of TouchGFX is split into several components, dealing with various aspects of hardware interaction. If the target platform uses the same hardware as the evaluation board abstraction layers in the TouchGFX distribution, the porting effort is simply to pick and combine the appropriate hardware components already implemented. If the target platform differs in one or more areas, some of the TouchGFX HAL components must be implemented to fit this specific hardware. For the sake of completeness this guide will cover the process of implementing all the components of the TouchGFX hardware abstraction, but chances are you will be able to skip at least some chapters, provided that you choose already supported hardware components for your design.

In general terms, a typical hardware platform will consist of the following:

- An MCU with a TFT controller and optionally DMA capabilities to efficiently transfer graphics data
- External RAM for frame buffers
- External memory-mapped flash for storing graphics data
- A TFT display, with resistive or capacitive touch

Note

If the external flash is not memory-mapped, see Non-memory mapped external flash / Supporting bitmap caching.

If your display is 1BPP monochrome, see Configuring TouchGFX for 1BPP Monochrome

If your MCU does not have a TFT controller, see MCUs without TFT controller

8.1 Recommended Approach

The task of porting TouchGFX will quickly become almost unmanageable if simply attempting to make everything work in one fell swoop. Instead we recommend the following gradual approach, that adds on portions of TouchGFX functionality through a number of steps. As previously mentioned, depending on your platform, you might be able to skip a few or even all these steps.

1. Initial system (without TouchGFX)
   (a) Board Bring-up
   (b) LCD Pin Configuration
   (c) TFT controller initialization
   (d) Simple Display Test
2. Minimalistic TouchGFX System with software rendering from internal flash
   (a) MCU Component
   (b) OS Abstraction Layer
   (c) BoardConfiguration (Wiring Things Together)
   (d) Installing LCD Interrupt Handler
3. Placing Graphics Data in External Flash
4. Touch Driver
5. Add a DMA Component
6. Adding a GPIO implementation

8.2 Initial system

A stable, initial configuration with a working operating system and display is a good place to start before adding TouchGFX code to your project. This chapter deals with the basic system configuration needed before integrating TouchGFX. However, the topic of initializating the MCU clock domains and peripherals is somewhat out of scope, so it is only discussed in general terms here.

8.2.1 Board Bring-up

Hardware initialization usually consists of the following:

- Configure the MCU to use the appropriate clock domain, most likely such that the MCU runs at the highest possible frequency.
- Configure the external memory controller(s) such that the external RAM and external flash are accessible. Since external memory generally is the bottleneck of any graphics system, it is quite important that the bus settings are tuned to high performance.
- Configure the MCU pins connected to the display.
- Configure the TFT controller.

The main reason these tasks are kept outside of TouchGFX components is that many projects already have a mechanism for hardware initialization and maybe even a software framework for dealing with HAL access. Naturally, if the target platform uses an MCU and memories already supported by TouchGFX, you can grab the hardware initialization code from one of the evaluation board packages and use that.

8.2.2 LCD Pin Configuration

The appropriate MCU pins must be configured to function as LCD interface pins. That includes the RGB data pins and control signals (PIXELCLOCK, HSYNC, VSYNC, DE). You will probably also need to configure a backlight control pin. These are normally connected to a PWM-capable output which can initially be set to full duty cycle or used as a simple general purpose digital output to obtain maximum backlight intensity.

8.2.3 TFT controller initialization

Configuration of the TFT controller varies depending on platform hardware because peripheral registers and display timings vary across MCU families and displays. As such, this is a general description of the concepts involved in initializing TFT controllers for TouchGFX application, which can hopefully aid in the configuration of your concrete platform.

We recommend that you take a look at the Timing and TFT Controller Integration diagram to understand the basic concepts of porches vs. active area and their effect on VSYNC,HSYNC signals.

A TFT controller normally needs to know the following:

- *Frame buffer address*. Set this to the start address of the block of external memory you will want to use for frame buffers.
- *Color format*. This should be 16-bit, RGB565. There is usually also a flag to control the color ordering (BGR vs RGB).
- *Interrupts*. These must not occur until TouchGFX is properly initialized, so leave them off initially. The MCU Component will configure these later.
- *Clock source*. Consult the display datasheet/manual for the acceptable pixel clock frequency range. You will normally want the pixel clock to run as fast as the display accepts. This might require the use of clock divider mechanisms of the MCU. A typical value could be around 34MHz.
- *Back porches*. These define the vertical and horizontal delays between VSYNC/HSYNC signals and actual data (the active area). Back porch values must normally be a specific number of pixel clocks for the image to appear at the correct position on the display. Consult the display datasheet for correct values.
- *Front porches*. These delays can be modified to obtain the desired VSYNC frequency. You should aim for front porch settings which yield a VSYNC approx. every 17-20ms.
- *Arbitration priority*. Not usually a setting of the TFT controller itself, but some MCUs allow configuring the access priority in case several bus masters wish to access e.g. the external memory simultaneously. It is important that the TFT controller has a high priority in this case, as starvation will lead to visible frame errors.

8.2.4 Simple Display Test

With the TFT controller running, a simple color test can help verify the correctness of the configuration. Fill the entire frame buffer region with various 16-bit color values using memcpy:

```
uint16_t black = 0x0;
uint16_t white = 0xFFFF;
uint16_t red   = 0xF800;
uint16_t green = 0x7E0;
uint16_t blue  = 0x1D;
```

Note that it might be necessary to also configure or tweak the display itself. This is usually done by setting parameters through an SPI interface to the display. In most cases however the standard configuration pre-programmed by the display manufacturer will work just fine. Consult the display manual for information on how to tune it if necessary.

8.3 Minimalistic TouchGFX System

With a basic TFT configuration running, the next step is to create a basic TouchGFX port and integrate it into your project. TouchGFX itself expects the hardware initialization from the previous steps to be done before TouchGFX is initialized, so keep the configuration of clocks, pins, external memory, and TFT controller from the simple display test.

A TouchGFX port is made up of several components. A number of these already exist for a range of hardware platforms and can be reused if they match yours. To get TouchGFX up and running you will need a working MCU component, an OS abstraction layer implementation, a DMA component and finally some TouchGFX initialization code to tie everything together.

Before starting the actual port, you should create a very simple TouchGFX application, consisting of a single screen showing just a full-screen touchgfx::Box widget of some color. You can base your test application on the Empty↩ Application found under app/template (see Creating a New Application). Verify using the PC Simulator that the test application works as intended (remember to modify the simulator/main.cpp file to reflect the display resolution of your platform).

8.3.1 MCU Component

The MCU component is an MCU specific (or MCU-family specific) subclass of the generic touchgfx::HAL class. It has two responsibilities: controlling interrupts, and interfacing with the frame buffer address register of the TFT controller. To create an MCU component, you simply need to create a HAL subclass and provide implementation for the six pure virtual functions of the base class. Care has been taken to make sure these functions are as simple as possible.

The following MCU components are already available in The TouchGFX Distribution :

- **NXP LPC1788** (`include/platform/hal/nxp/mcu/LPC17xx/NXP1788HAL.hpp`)
- **NXP LPC43xx series** (`include/platform/hal/nxp/mcu/LPC43xx/NXP43XXHAL.hpp`)
- **NXP LPC18xx series** (`include/platform/hal/nxp/mcu/LPC18xx/NXP18xxHAL.hpp`)
- **ST STM32F429/STM32F439** (`include/platform/hal/ST/mcu/stm32f4x9/STM32F4HAL.↩hpp`)
- **Freescale K70** (`include/platform/hal/Freescale/mcu/MK70F12/FreescaleK70HAL.↩hpp`)

The pure virtual functions that need to be implemented are:

Function	Purpose
configureInterrupts()	**Configure the priorities of the DMA interrupt and the LCD/TFT controller interrupt.** This function must ensure that the priorities of the DMA and TFT interrupts are *the same*. Both interrupt vector routines will access TouchGFX internal state information, and if either ISR preempts the other, race conditions can occur. It is therefore quite important that these two interrupts are configured to be processed in a tail-chained/round-robin manner. *Note:* On Cortex-M, some RTOSes require that their API functions are never called from interrupt contexts with a higher priority than those used by the OS itself (usually the SVC interrupt). E.g. if using FreeRTOS on Cortex-M, make sure the priorities set by this function for DMA and TFT interrupts are *functionally* lower (on Cortex-M this means numerically higher) than configMAX_SYSCALL_INTERRUPT_PRIORITY defined in FreeRTOSConfig.h. Otherwise consult your RTOS documentation to determine whether this requirement is present.
enableLCDControllerInterrupt()	This function must configure the TFT controller to produce an interrupt on VSYNC. *Note:* This function will automatically be called by TouchGFX once the framework has been properly initialized and is ready to receive interrupts. Make sure TFT controller interrupts at an earlier time than this.
disableInterrupts()	This function must disable both the DMA and LCD interrupt vectors (in the NVIC for Cortex-M devices). Used by TouchGFX to temporarily guard critical sections.
enableInterrupts()	This function must enable both the DMA and LCD interrupt vectors (in the NVIC for Cortex-M devices). Used by TouchGFX to temporarily guard critical sections.
getTFTFrameBuffer()	This function must return the start address of the frame buffer currently being used by the TFT controller. Typically implemented by simply returning the peripheral register containing the frame buffer start address.
setTFTFrameBuffer()	This function must set the start address of the frame buffer to be used by the TFT controller. Typically implemented by setting the peripheral register containing frame buffer start address.

8.3.2 OS Abstraction Layer

In TouchGFX, the touchgfx::OSWrappers component is a single class that serves as an OS abstraction layer. TouchGFX provides an implementation for FreeRTOS which you can use if that is your operating system, otherwise you need to provide an implementation for the specific operating system you use. The use of OS features is kept to a minimum and it should be possible to implement this component with relative ease. The implementation requires two synchronization points which can be implemented as binary semaphores, mutexes or queues (with a size of 1) depending on your operating system. The synchronization points are:

8.3.2.1 VSYNC signaling

The GUI task is synchronized with the occurence of VSYNC events by the TFT controller. This is done in practice by having the GUI task wait on a semaphore/queue/mutex (through waitForVSync). The TFT interrupt routine, which is fired when a VSYNC is created, will post to this semaphore (through signalVSync) which will cause the GUI task to be scheduled in.

8.3.2.2 Frame Buffer Access

Access to the frame buffer needs to be synchronized because both the GUI task and the processor DMA engine may draw in the frame buffer in parallel.

The functions that need to be implemented in OSWrappers are:

Function	Purpose
initialize()	Create the two binary semaphores/mutexes/queues
signalVSync	Signal that a VSYNC has occured. Should make the vsync queue/mutex available. *Note:* This function is called from an ISR, and should (depending on OS) trigger a scheduling
waitForVSync()	This function blocks until a VSYNC occurs. *Note:* This function must first clear the mutex/queue and then wait for the next one to occur.
takeFrameBufferSemaphore()	Take the frame buffer semaphore. Blocks until semaphore is available.
tryTakeFrameBufferSemaphore()	Attempt to obtain the frame buffer semaphore. If semaphore is not available, do nothing. *Note:* Must return immediately.
giveFrameBufferSemaphore()	Release the frame buffer semaphore
giveFrameBufferSemaphoreFromISR	Release the frame buffer semaphore in a way that is safe in interrupt context. *Note:* Called from ISR.

8.3.3 Installing LCD Interrupt Handler

The LCD interrupt handler drives the timing that triggers event processing and frame endering in TouchGFX. Most TFT controllers will have a single interrupt vector associated, which can be configured to trigger on various timing events within the controller. There are two distinct TFT controller events that need to be handled in TouchGFX context: One is VSYNC, which is used to begin the rendering pass of the next frame. The second one is vertical front porch entry (meaning that the display update procedure has *just* completed). When this occurs, TouchGFX can begin transferring the DMA queue. Since we are not yet using DMA the latter is not important for now, but we might as well configure it anyway. For a diagram of when these two events occur, see Timing and TFT Controller Integration.

The following pseudo code illustrates a typical implementation of the LCD controller interrupt handler:

```
extern "C"
__irq void LCD_IRQHandler( void )
{
  if (int_src == VSYNC) //Determine if interrupt was caused by VSYNC
  {
      // This will signal the GUI task to begin processing a new frame.
      OSWrappers::signalVSync();
      // Configure TFT controller to fire interrupt on porch
      setIntSourcePorch();
  }
  else if (int_src == FRONTPORCH) // Interrupt caused by entering front porch area.
  {
      // This will signal the framework that the DMA queue can be safely started.
      HAL::getInstance()->frontPorchEntered();
      // Configure TFT controller to fire interrupt on VSYNC
      setIntSourceVSYNC();
  }
}
```

8.3.4 BoardConfiguration (Wiring Things Together)

Now that the mandatory components have been created, it is time to get TouchGFX up and running. What we usually do is create a BoardConfiguration.cpp file which contains approximately the following:

```
#include <common/TouchGFXInit.hpp>
#include <touchgfx/hal/BoardConfiguration.hpp>
#include <touchgfx/hal/OSWrappers.hpp>
#include <touchgfx/hal/NoDMA.hpp>
#include <platform/driver/lcd/LCD16bpp.hpp>
#include <platform/driver/touch/NoTouchController.hpp>
#include <path/to/MCUComponent.hpp> //Modify to reflect location of your MCU component (HAL subclass)

NoDMA dma; //Initially we do not want DMA support
NoTouchController tc; //Likewise we do not want touch input (yet).
LCD16bpp display;

namespace touchgfx
{
  void hw_init()
  {
    // Hardware initialization code from step 1.
  }
  void touchgfx_init()
  {
    //Instantiate HAL
    HAL& hal = touchgfx_generic_init<MCUComponentClassName>(dma, display, tc, DISPLAYWIDTH, DISPLAYHEIGHT,
      0, 0);
    hal.setFrameBufferStartAddress((uint16_t*)SDRAM_BASE_ADDR);
  }
}
```

Note

The DISPLAYWIDTH and DISPLAYHEIGHT arguments to touchgfx_generic_init must match the native resolution of the display.

Now we need to initialize hardware and the framework by calling these two functions from somewhere appropriate. Creating an OS task for the GUI and starting the scheduler will complete our basic system with no touch support and no DMA transfers. It could look like the following:

```
#include <touchgfx/hal/HAL.hpp>
#include <touchgfx/hal/BoardConfiguration.hpp>

using namespace touchgfx;

// The OS-specific GUI task should execute this function.
static void GUITask(void* params)
{
  touchgfx::HAL::getInstance()->taskEntry(); /*Starts processing events.
       Never returns.*/
}

int main (void)
{
  hw_init();
  touchgfx_init();

  // Create OS GUI task.
  // Start scheduler.
  // Example for FreeRTOS:
  xTaskCreate( GUITask, (signed char*)"GUITask",
               configGUI_TASK_STK_SIZE,
               NULL,
               configGUI_TASK_PRIORITY,
               NULL);

  vTaskStartScheduler();


  for(;;);

}
```

At this point you should now have a basic port of TouchGFX, capable of executing your test application.

8.4 Placing Graphics Data in External Flash

Up until this point, the generated code for any graphics data/bitmaps (which is essentially just const arrays) have been located in the internal flash of the MCU along with the code and other const data. However, since bitmaps take up a substantial amount of space, you will almost always need to place bitmap data in the external flash instead. As a prerequisite from Board Bring-up, there should already be read access to the external flash chip at runtime. To get started on this task, modify your test application to use an touchgfx::Image widget, add a .bmp or .png image to your assets folder and build the simulator in order to generate the image data. Note that your Image widget must be configured to use the new bitmap. Verify that the simulator correctly displays your Image widget.

8.4.1 Linker Script

The const arrays located in the autogenerated .cpp files under generated/images image data are annotated with a section attribute for easy placement in external flash. Simply amend the linker script for your target compiler to place "section ExtFlashSection" in the external flash memory area. Consult the MCU manual to determine the address space of this memory region. Note that the linker scripts used by the example applications for the supported evaluation boards already do this for IAR, Keil and GCC cross-compilers, so look there for inspiration (or reuse).

8.4.2 Flash Loader

You are going to need a mechanism for programming the external flash with the contents of the ExtFlashSection of the produced binary image. This may or may not be easy depending on your particular toolchain and MCU/flash combination. IAR Embedded Workbench has a flash loader plugin mechanism which allows you to write a custom flash loader for the appropriate MCU and flash. IAR will, when configured to use such a flash loader, automatically invoke the appropriate flash loader a specified memory address range when deploying code (e.g. the built-in loader for internal flash, and a custom loader for the external flash). If your hardware matches one of the TouchGFX evaluation platforms, you can reuse the flash loader from there. The same overall mechanism applies to the Keil compiler, which also provides a collection of flash loader algorithms and the ability to create custom loaders. If compiling with GCC there are several tools available, both generic and platform specific. See Target Hardware for examples.

8.4.3 Separate Projects for Internal and External Flash

Normally our projects are configured to include all code files, including generated data for the external flash, in a single project compiled as a single binary. There are, however, a few occasions where it makes sense to separate the code that goes into the external flash in a standalone project. For example the external flash loader used might require the binary to contain only external flash data, or the amount of external flash data is so large that you do not want to incur the delay of flashing it every time code in the internal flash changes. If deciding to split the project, please note the following. The autogenerated file BitmapDatabase.cpp must be placed in the internal flash project. It contains a table of addresses for the graphics data (i.e. external flash addresses). In order to successfully link the internal flash project in this case, the *external* flash project must be configured to emit a symbol file, listing the addresses of the symbols placed in external flash. This symbol file must then be included in the internal flash project, such that the linker is able to resolve the physical address of the symbols mentioned in the BitmapDatabase table.

At this point you should be able to deploy any application (including the examples and demos), since graphics data is now stored in the external flash. However with no touch input, its usefulness is somewhat limited.

8.5 Touch Driver

Adding a touch driver is done by replacing the touchgfx::NoTouchController we declared in our BoardConfiguration with another object deriving from the base class touchgfx::TouchController that is capable of interfacing with the concrete touch controller mounted on the display. This object needs to provide implementation for two functions: init

which is called automatically during initialization, and sampleTouch. The latter will be called once for every frame (i.e. once per VSYNC).

The `include/platform/driver/touch/ST1232TouchController.hpp` that comes with TouchGFX can be used for most of the Sitronix ST123x series of touch controllers, which are quite common. This driver requires an I^2C implementation for the target microcontroller in order to work.

Note

If polling the touch controller is MCU intensive, it might be a good idea to have sampleTouch only actually performing a sample once every other (or every third) frame. This would normally still be frequent enough for obtaining a responsive UI while reducing MCU load. This can be done through the HAL::setTouchSampleRate function.

8.5.1 Coordinate Transformations

The default behaviour of TouchGFX is that no coordinate transformations take place. That means that the coordinates you report as out parameters in the touchgfx::TouchController::sampleTouch function will be propagated to the widgets of your screen unmodified. If your display has a touch controller IC, which is the case for all capacitive touch displays and some resistive touch displays, the coordinates reported by the touch IC will most likely already be correct coordinates within the pixel dimensions of the display, and these can therefore be reported unmodified by sampleTouch. However, if your display is a 4-wire resistive touch where the MCU performs A/D conversions in order to read coordinate values, you will need a way to transform the (usually 12-bit) ADC values to pixel coordinates. In order to help with this scenario, TouchGFX provides a TouchCalibration matrix, which can translate ADC values to pixel coordinates. In order to use this feature, you must initialize the calibration matrix with a set of three reference x,y coordinates in screen space, and a corresponding set of ADC values for these three coordinates. The following is an example of such an initialization:

```
#include <touchgfx/transformations/TouchCalibration.hpp>

using namespace touchgfx;

static  Point  lcdCalibPoints[3] =  /* Display size is 800x480 */
{
  {  80,  48 }, /* 10%, 10% */
  { 720, 240 }, /* 90%, 50% */
  { 400, 432 }, /* 50%, 90% */
};

//calibration ADC 12 bit values
static  Point  touchCalibPoints[3] =
{
  { 581, 3128 },  /* ADC values read when clicking in 80,48 */
  { 3475, 2025 }, /* ADC values read when clicking in 720,240 */
  { 2001, 978 },  /* ADC values read when clicking in 400,432 */
};

TouchCalibration::setCalibrationMatrix(lcdCalibPoints,
      touchCalibPoints);
// From this point on, sampleTouch can just report raw ADC values, and TouchGFX will automatically convert
// these to screen space coordinates based on the three setpoints.
```

If you wish to apply a more advanced coordinate translation you can of course just do that within your driver and disregard the touch calibration functionality.

8.6 DMA Component

The last task in the porting effort is to make use of DMA to transfer graphics data from external flash (or external RAM cache) to the frame buffer.

Note

If your UI is 1 bit per pixel (monochrome), do not bother implementing a DMA component, as it is not supported by the framework in 1BPP mode. Use NoDMA instead.

Adding DMA support will *substantially* reduce the MCU load imposed by TouchGFX on your target platform. It is therefore highly recommended to support at least BLIT_OP_COPY (basic drawing of opaque bitmaps). In TouchGFX, a DMA component acts as an interface to the DMA peripheral of the microcontroller, and is as such usually tightly coupled with the MCU component. It is responsible for initiating DMA transfers of graphics data, and notifying TouchGFX when these transfers are complete. In terms of code, a DMA component is specifically a subclass of the touchgfx::DMA_Interface class.

It is already present for the following:

- **NXP LPC43xx series** (`include/platform/hal/nxp/mcu/LPC43xx/NXP43XXDMA.hpp`)
- **NXP LPC17xx series** (`include/platform/hal/nxp/mcu/LPC17xx/NXP1788DMA.hpp`)
- **NXP LPC18xx series** (`include/platform/hal/nxp/mcu/LPC18xx/NXP18xxDMA.hpp`)
- **ST STM32F429/STM32F439** (`include/platform/hal/ST/mcu/stm32f4x9/STM32F4DMA.hpp`)
- **Freescale K70** (`include/platform/hal/Freescale/mcu/MK70F12/FreescaleK70DMA.hpp`)

When creating a DMA component, the following pure virtual functions need to be implemented. They are explained in greater detail in the following table.

Function	Purpose
getBlitCaps()	This function returns which blitting capabilities are supported by this DMA component.
setupDataCopy()	Initiates a DMA transfer. Used for all copy-based blit operations (bitmap transfers), where source address is incremented for each pixel. This function is called when encountering the following blit operations: • BLIT_OP_COPY • BLIT_OP_COPY_WITH_ALPHA • BLIT_OP_COPY_ARGB8888 • BLIT_OP_COPY_ARGB8888_WITH_ALPHA • BLIT_OP_COPY_A4 • BLIT_OP_COPY_A8

setupDataFill()	Initiates a DMA transfer. Used for all fill-based blit operations (i.e. touchgfx::Box), where a single color is copied to all pixels (i.e. no source increment). This function is called when encountering the following blit operations: • BLIT_OP_FILL • BLIT_OP_FILL_WITH_ALPHA
signalDMAInterrupt()	This function, which is called by the DMA ISR, must determine whether the interrupt was caused by a DMA transfer being completed. If so, it must call executeCompleted() in the base class.

During the TouchGFX rendering pass, the framework will query the getBlitCaps function to determine whether a given blitting operation can be delegated to the hardware (the DMA object). If that is the case, the touchgfx::DMA_Interface::setupDataCopy or touchgfx::DMA_Interface::setupDataFill will be called automatically when appropriate. Blitting operations that are not supported by the DMA component will be rendered using software fallback instead. Therefore, the DMA component must be able to properly handle the blitting operations returned by the getBlitCaps function.

As mentioned, the most important blit operation to support is BLIT_OP_COPY, which is a basic data copy from bitmap data to the frame buffer. It is used for drawing bitmaps that are completely solid (no alpha blending). In most cases this will cover the majority of the pixel transfers for a given application. Most, if not all DMA engines on 32-bit MCUs will be able to support this operation. The BLIT_OP_FILL operation is used for drawing solid rectangles of a specific color (the touchgfx::Box widget). It too should be implementable on any hardware, but this widget is not normally used very often, so it is of less importance. The remaining blit operations require advanced DMA hardware, capable of applying alpha-blending or doing color format conversions on the fly.

There are two important things to note when implementing a DMA subclass:

1. The DMA must support *two-dimensional* transfers
2. The transfer must be as fast as possible

8.6.1 Two-dimensional Transfers

Because of the occlusion culling mechanism of TouchGFX, the actual transfer of bitmap data to the frame buffer will be split in many partial transfers. This has the implication that most DMA transfers will be two-dimensional as opposed to a simple linear transfer. If only a portion of a bitmap is to be transferred, then for each line an offset (address increment) has to be applied for both the source and destination address, and those two offsets will have a different value since the width of the display is most likely different than the width of the bitmap. While some DMA peripherals support this type of transfer natively by providing registers for both source and destination address increment, it is not too common. If this is not supported, the functionality can usually be achieved by doing a scatter-gather transfer instead, where a linked list is constructed in RAM, with an element for each line.

8.6.2 Optimizing DMA Transfers

The bottleneck of the system in terms of graphics performance is the transfer of pixels from external flash to the frame buffer. Therefore, the performance of the DMA transfers are of paramount importance, and it is usually a good investment to spend time on optimizing the performance of these transfers. Performing 32-bit transfers whenever possible is an easy optimization yielding a substantial improvement, instead of just transferring a single pixel (16 bits) at a time. Most DMAs support setting a transfer unit size of 32 bits. Of course this optimization is only possible to apply for transfers that are actually 32-bit aligned, which usually means that the following must be true:

```
bool use32Bit = (blitOp.nSteps%2)==0 &&              // Pixels pr line is even
                ((unsigned long)blitOp.pSrc&3)==0 && // src-address is 32bit-aligned
                ((unsigned long)blitOp.pDst&3)==0 && // dst-address is 32bit-aligned
                (blitOp.srcLoopStride%2)==0 &&       // src-stride is even
                (blitOp.dstLoopStride%2)==0;         // dst-stride is even
```

This optimization is the basis for the recommendation to place bitmaps on even coordinates and give them an even width.

8.6.3 Example Transfer

The framework itself will perform the address calculations necessary for both source and destination addresses before calling setupDataCopy, so the fields of the BlitOp struct passed as arguments to this function just need to be mapped to the appropriate DMA peripheral registers of the MCU. In this example, the blit op is a request to copy a partial area of a bitmap to a frame buffer with a width of 480 pixels:

Figure 8.1: Example bitmap 160x100 pixels, with a subsection of 75x80 pixels that needs to be transferred by DMA

The BlitOp struct passed to setupDataCopy in this example will contain the following values:

```
pSrc: Address of first pixel to be transferred (commonly external flash)
pDst: Destination address of first pixel in frame buffer (commonly external RAM)
nSteps: 75 (pixels in a line)
nLoops: 80 (number of lines)
srcLoopStride: 160 (width of bitmap)
dstLoopStride: 480 (width of frame buffer)
operation: BLIT_OP_COPY (simple 2D transfer from source to destination)
color: Do not care (N/A for BLIT_OP_COPY)
alpha: 255 (Always completely opaque for BLIT_OP_COPY)
```

The TouchGFX framework must be notified when a transfer is complete, so the frame buffer semaphore can be released. To achieve this, the DMA transfer should be configured to generate an interrupt upon transfer completion.

Note

If using scatter-gather, make sure an interrupt is only generated upon completion of the last line

The interrupt service routine which fires upon completion is discussed in Interrupt Configuration.

8.6.4 Interrupt Configuration

TouchGFX does not begin processing the next element of the DMA queue until it has been informed of the completion of the previous DMA transfer. Therefore, a DMA interrupt handler needs to be installed, which should look something like the following:

```
extern "C"
__irq void DMA_IRQHandler( void )
```

```
{
   touchgfx::HAL::getInstance()->signalDMAInterrupt();
}
```

The notification to TouchGFX goes through the HAL object since it is globally accessible. HAL will simply call signalDMAInterrupt on the generic DMA object, which is a pure virtual function in the base class that needs to implemented for the specific MCU used. This implementation should determine whether the DMA interrupt means that a transfer has been completed and in that case call touchgfx::DMA_Interface::executeCompleted :

```
void MyDMAComponent::signalDMAInterrupt()
{
  if (int_src == myLastTransfer)//Determine whether interrupt was caused by our last transfer being
      complete.
  {
    executeCompleted();
  }
}
```

Note

The execution path of both the DMA and TFT interrupt will end up in semaphore manipulation functions of touchgfx::OSWrappers. These are non-blocking of course, but your operating system might require that the interrupt priorities from these two sources are configured to be lower priority than those used by the operating system itself.

8.7 Adding a GPIO implementation

The TouchGFX framework expects an implementation of the touchgfx::GPIO class to be present as part of the hardware abstraction layer. The purpose of this class is to provide facilities for instrumentation of the timing and performance of TouchGFX. It is a simple class, capable of toggling four GPIO pins of your own choice.

Note

The class is mandatory, but the functions can be left empty / un-implemented.

The GPIO set/clear/toggle functions serve to provide timing output for the following four signals:

Signal	Purpose	Note
VSYNC_FREQ	Readout for the VSYNC signal.	Platform-dependent. Your porting should include calls to GPIO set/clear for this signal in the LCD interrupt handler.
RENDER_TIME	Goes high when TouchGFX starts processing a frame, and low on completion	This signal is toggled automatically by the framework
FRAME_RATE	Toggles whenever a frame buffer swap occurs	This signal is toggled automatically by the framework
MCU_ACTIVE	Toggles when the OS Idle task is entered/exited	OS-dependent. Already implemented for FreeRTOS in the corresponding OSWrappers.cpp file. For other OSes must be implemented manually

If you implement the GPIO class such that four pins of your choice are toggled, you can attach an oscilloscope and get readouts similar to those presented in the Frame Rate section of the Advanced chapter. This is normally very useful for optimizing performance.

See also Performance Measurement

8.8 Non-memory mapped external flash / Supporting bitmap caching

This chapter deals with the special case of having an external flash which cannot be read by simply dereferencing a pointer to somewhere in the external flash memory region. Instead, reading the flash contents must go through a device-specific driver function. This is generally the case with all serial flashes or flashes which require bad block handling (e.g. NAND flashes). As an exception some NXP microcontrollers provide a memory-mapped interface for serial flashes (SPIFI), in which case a serial NOR flash can be used straightforwardly and this chapter can be disregarded.

For performance reasons, TouchGFX expects all graphics data stored in external flash to be directly accessible, without going through a driver layer. Therefore TouchGFX cannot render directly from a non-memory mapped flash. Instead we provide a simple mechanism for caching all graphics data in external RAM during power-up. The rendering mechanism will then read from there instead of the external flash. This, of course, requires that the external RAM is large enough to hold not only frame buffers but also a copy of all graphics data.

The details of using bitmap caching in an application is described in the Bitmap Caching section of the Advanced page. But in order for bitmap caching to work properly on the specific platform you are porting to, you need to supply a driver for copying a block of data from the flash to the cache and to configure TouchGFX with a memory region to use for caching. The former is done by implementing the touchgfx::HAL::blockCopy function in your MCU Component. The function simply needs to be able to copy a number of bytes from an address in the flash to the caching area (typically external RAM). touchgfx::HAL::blockCopy will be called automatically a number of times (one for each bitmap) when caching bitmaps.

Recall from Linker Script that bitmap data is placed in a section called ExtFlashSection. If you want to e.g. place bitmaps on an SD card and use bitmap caching you can do the following:

1. Change linker script to place ExtFlashSection at some unused area, e.g. 0x24000000.
2. Implement HAL::blockCopy in your subclass like this:

```
// This function is called whenever a bitmap is cached. Must copy a number of bytes from the
       (non-memory-mapped) source to the cache.
bool MyHAL::blockCopy(void* RESTRICT dest, const void* RESTRICT src, uint32_t numBytes)
{
    void* dataOffset = src - 0x24000000;
    sdcard_read(dest, dataOffset, numBytes);
    return true;
}
```

The example function above assumes that there exists a driver function called sdcard_read() which is capable of reading from the SD card. You need to supply the correct function for copying data of course.

Note

There is a default implementation of blockCopy in the generic HAL class, which will essentially perform a memcpy() from src to dest. If your bitmap data (src) is in fact located in a memory-mapped region, there is no need to provide a blockCopy specialization in your HAL subclass. Likewise, if bitmap caching is not used in your application(s), there is no need to provide a specialization either.

8.8.1 Configuring the Cache

In order to make use of bitmap caching, TouchGFX must be informed of the memory area to use for caching as well as the size of that memory area. This is done through the touchgfx::touchgfx_generic_init function (which is called from your BoardConfiguration file) where the last two arguments configures the bitmap cache. The first of these is simply a pointer to the beginning of the memory region where the bitmap cache is to be located. This will usually be external RAM. The last argument configures the size of the cache area, and will impact how many bitmaps it is possible to keep in the cache simultaneously. The required size depends somewhat on data alignment on your platform as well as the bitmap format used. As a general rule of thumb, the required size of your bitmap cache will be

```
CacheSize = TOTAL_NUMBER_OF_BITMAPS_IN_APPLICATION * sizeof(uint8_t*) + SIZE_IN_BYTES_OF_CACHED_BITMAPS
```

Where the size of each cached bitmap will be approximately

```
width * height * 2 for RGB565 images with no alpha channel
width * height * 3 for RGB565 images with alpha channel
width * height * 4 for ARGB8888 images
```

When caching a bitmap, the return value will indicate if the operation failed due to lack of memory. If you are certain that there is enough memory available, you need not bother with calculating the size parameter and can instead just pass 0xFFFFFFFF as size argument. This is safe to do since the caching mechanism will not access the remaining memory chunk outside the needed size.

8.9 Optimization

In many cases, some tweaking will substantially increase performance. It is therefore strongly recommended to follow the hints in this chapter.

8.9.1 Optimizing DMA During TFT Controller Access

The single most important performance optimization to perform is to allow DMA transfers to take place while the TFT controller is updating the display. This is done by calling the following function in your TouchGFX initialization code:

```
HAL::getInstance()->lockDMAToFrontPorch(false); //DMA is now no longer tied to the TFT controller.
```

When doing this, the DMA will no longer be locked to the TFT controller and thereby allowing DMA transfers to begin immediately, instead of having to wait until the front porch is entered (TFT controller exits active area). In most cases this will decrease the rendering time by around 25%. This optimization requires that the MCU of your platform is able to correctly perform arbiting of simulataneous RAM access by DMA and TFT. If this is not the case you will immediately see rendering errors on the display when enabling this optimization, in which case it cannot be used on that platform. The board configurations distributed with TouchGFX are configured to enable this optimization on the platforms where it is known to be supported. If you use a custom board configuration, or a board configuration from TouchGFX 4.1.1 or earlier, you must manually enable this optimization.

If using single buffering, this optimization can only be applied in cases where you do not use a regular TFT controller. It is applicable, for instance, when using 8080 or DSI mode displays in single buffering configuration.

Note

This optimization is scheduled to be default enabled in TouchGFX 5.0.

8.9.2 Optimizing VSYNC Timing

The chapter Timing and TFT Controller Integration explains in detail how the TFT controller timing relates to the rendering of frames. As mentioned earlier in the porting guide, having a VSYNC interval of around 16-20 milliseconds is generally a good value, but if you are seeing performance problems on particularly complex screens in your application, it is most likely a good idea to tweak the VSYNC interval.

8.9.3 Touch Sample Method and Frequency

If your application is very dependent on a low MCU load at all times, it can be a good idea to reduce the frequency with which the Touch Controller is being sampled, through the function HAL::setTouchSampleRate. In many cases it will be sufficient to sample touch approximately every three ticks instead of every tick or maybe even less. Most, if not all, capacitive touch controllers have an output pin to signal that touch data if available. By reading this pin, unnecessary polling of the touch controller can be avoided.

8.10 MCUs without TFT controller

TouchGFX can be configured to run on MCUs with no built-in TFT controller. In that case, one would normally attach a display though SPI or similar, and use some kind of protocol to set {x,y} positions and transfer pixel data over the serial connection. In this scenario, however, it is still necessary to use frame buffer(s). TouchGFX does *not* support rendering directly to a TFT display. After rendering to the frame buffer is complete, you can through a hook function transmit the changes to the display.

As explained in Timing and TFT Controller Integration, TouchGFX relies on a TFT controller interrupt signal to periodically activate the GUI task. In the absence of a TFT controller, a hardware timer or similar must be configured to periodically wake the GUI task instead:

```
void HW_Timer_IRQHandler()
{
  OSWrappers::signalVSync();
}
```

The tick frequency of this timer should be around 50Hz.

In order to transfer the contents of the frame buffer to the display, override the function HAL::flushFrameBuffer(const Rect&). This function is called at the end of each frame, with the parameter Rect describing the area of the frame buffer that was changed. This area must then be transferred to the display using the appropriate communication protocol.

When not using a TFT controller, you should enable the optimization mentioned in Optimizing DMA During TFT Controller Access

Chapter 9

Licensing

In the following the licenses for the TouchGFX Grapical Framework are described. This section covers the following parts:

- The core framework as precompiled libraries, TouchGFX driver and Core files for the following targets:
  - Windows/gcc
  - Windows/MSVS
  - Linux/gcc
  - ARM Cortex-M3/IAR
  - ARM Cortex-M4F/IAR
- TouchGFX examples, templates and demos
- Tools for converting images, texts and fonts
- The manual you are reading now

9.1 Precompiled Libraries

The non-evaluation versions license for the precompiled libraries, TouchGFX driver and Core files, for which Draupner Graphics is the copyright owner, is as follows:

```
 Copyright (C) 2014 Draupner Graphics A/S  http://www.touchgfx.com.
All rights reserved.

TouchGFX is protected by international copyright laws and the knowledge of
this source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be redistributed
in any way without the prior permission of Draupner Graphics.

This is licensed software, any use must strictly comply with the signed
license agreement and associated Terms & Conditions.

Standard Terms & Conditions can be seen on www.touchgfx.com

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Draupner Graphics A/S has
no obligation to support this Software. Draupner Graphics A/S is providing
the Software "AS IS", with no express or implied warranties of any kind,
including, but not limited to, any implied warranties of merchantability
or fitness for any particular purpose or warranties against infringement
of any proprietary rights of a third party.
```

```
Draupner Graphics A/S can not be held liable for any consequential, incidental,
or special damages, or any other relief, or for any claim by any third
party, arising from your use of this Software.
```

The evaluation version license for the precompiled libraries, TouchGFX driver and Core files, for which Draupner Graphics is the copyright owner, is as follows:

```
 Copyright (C) 2014 Draupner Graphics A/S  http://www.touchgfx.com.
All rights reserved.

TouchGFX is protected by international copyright laws and the knowledge of
this source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be redistributed
in any way without the prior permission of Draupner Graphics.

This is licensed software for evaluation use, any use must strictly comply
with the evaluation license agreement provided with delivery of the
TouchGFX software.

The evaluation license agreement can be seen on www.touchgfx.com

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Draupner Graphics A/S has
no obligation to support this Software. Draupner Graphics A/S is providing
the Software "AS IS", with no express or implied warranties of any kind,
including, but not limited to, any implied warranties of merchantability
or fitness for any particular purpose or warranties against infringement
of any proprietary rights of a third party.

Draupner Graphics A/S can not be held liable for any consequential, incidental,
or special damages, or any other relief, or for any claim by any third
party, arising from your use of this Software.
```

9.2 Examples, templates and demos

The licenses for TouchGFX examples, templates and demo programs for which Draupner Graphics is the copyright owner, is as follows:

```
 This file is free software and is provided for example purposes. You may
use, copy, and modify within the terms and conditions of the license agreement.

This is licensed software for evaluation use, any use must strictly comply
with the evaluation license agreement provided with delivery of the
TouchGFX software.

The evaluation license agreement can be seen on www.touchgfx.com

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Draupner Graphics A/S has
no obligation to support this Software. Draupner Graphics A/S is providing
the Software "AS IS", with no express or implied warranties of any kind,
including, but not limited to, any implied warranties of merchantability
or fitness for any particular purpose or warranties against infringement
of any proprietary rights of a third party.

Draupner Graphics A/S can not be held liable for any consequential, incidental,
or special damages, or any other relief, or for any claim by any third
party, arising from your use of this Software.
```

9.2.1 Exceptions

In the following the files are listed, for which Draupner Graphics are not the copyright owner (and therefore not covered by the above license.)

9.2.1.1 FreeRTOS

The FreeRTOS source code is licensed under the modified GNU General Public License (GPL) as described in the license: http://www.freertos.org/license.txt

9.2.1.2 libpng

libpng version 1.6.7 - November 14, 2013. (official PNG reference library)

- Copyright (c) 1998-2013 Glenn Randers-Pehrson
- Copyright (c) 1996-1997 Andreas Dilger
- Copyright (c) 1995-1996 Guy Eric Schalnat, Group 42, Inc.

This code is released under the libpng license http://www.libpng.org/pub/png/src/libpng-LIC↩ENSE.txt.

9.2.1.3 Hal Related Source Code

In case of the various header files provided by silicon manufacturers and tool vendors that define processor specific memory addresses and utility macros, the users must ensure license conditions are adhered to for any use other than compilation of the TouchGFX demo and examples applications.

9.2.1.4 Fonts

A TouchGFX license does not in any way provide licenses for commercial use of any TrueType and OpenType fonts included in demos and examples.

MISO is an architectural lettering font completed in 2006 by Mårten Nettelbladt. See http://martennettelbladt.↩se/miso/

9.2.1.5 Penner Easing Equations

Robert Penner's easing equations are Open Source under the BSD License. The license is as follows:

```
Copyright (c) 2001 Robert Penner~

All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:
```

- Redistributions of source code must retain the above copyright notice, this list of conditions ar
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions
- Neither the name of the author nor the names of contributors may be used to endorse or promote pr

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

9.3 Font Converter

The Font Converter is the utility program used for generating source code to include fonts into the target and simulator source code.

Portions of this software are copyright © 2012 The FreeType Project (www.freetype.org). All rights reserved.

Draupner Graphics is the copyright owner for the rest of the Font Converter except for the files listed in the subsection below. The licenses for the Font Converter, for which Draupner Graphics is the copyright owner, is as follows:

```
 Copyright (C) 2014 Draupner Graphics A/S  http://www.touchgfx.com.
All rights reserved.

TouchGFX is protected by international copyright laws and the knowledge of
this source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be redistributed
in any way without the prior permission of Draupner Graphics.

This is licensed software, any use must strictly comply with the signed
license agreement and associated Terms & Conditions.

Standard Terms & Conditions can be seen on www.touchgfx.com

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Draupner Graphics A/S has
no obligation to support this Software. Draupner Graphics A/S is providing
the Software "AS IS", with no express or implied warranties of any kind,
including, but not limited to, any implied warranties of merchantability
or fitness for any particular purpose or warranties against infringement
of any proprietary rights of a third party.

Draupner Graphics A/S can not be held liable for any consequential, incidental,
or special damages, or any other relief, or for any claim by any third
party, arising from your use of this Software.
```

9.3.1 Exceptions

Portions of this software are copyright © 2012 The FreeType Project (www.freetype.org). All rights reserved. FreeType is licensed under the FreeType License (FTL) (http://git.savannah.gnu.org/cgit/freetype/freetype2.git/tree/docs/FTL.TXT).

9.4 Image Converter

The Image Converter is the utility program used for generating source code to include images into the target and simulator source code.

The Image Converter uses the libpng library for reading png images.

Draupner Graphics is the copyright owner of the Image Converter, except for the libpng library and the file Dither.cpp as listed in the subsection below.

The licenses for the rest of the Image Converter, for which Draupner Graphics is the copyright owner, is as follows:

```
Copyright (C) 2014 Draupner Graphics A/S  http://www.touchgfx.com.
All rights reserved.

TouchGFX is protected by international copyright laws and the knowledge of
this source code may not be used to write a similar product. This file may
only be used in accordance with a license and should not be redistributed
in any way without the prior permission of Draupner Graphics.

This is licensed software, any use must strictly comply with the signed
license agreement and associated Terms & Conditions.

Standard Terms & Conditions can be seen on www.touchgfx.com

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Draupner Graphics A/S has
no obligation to support this Software. Draupner Graphics A/S is providing
the Software "AS IS", with no express or implied warranties of any kind,
including, but not limited to, any implied warranties of merchantability
or fitness for any particular purpose or warranties against infringement
of any proprietary rights of a third party.

Draupner Graphics A/S can not be held liable for any consequential, incidental,
or special damages, or any other relief, or for any claim by any third
party, arising from your use of this Software.
```

9.4.1 Exceptions

Part of the Image converter (the Dither class (Dither.cpp)) is licensed under the terms of the GNU Lesser General Public License version 2.1 as published by the Free Software Foundation and appearing in the file LICENSE.LGPL included in the packaging of this file (see http://www.gnu.org/licenses/old-licenses/lgpl-2.1.html).

Draupner Graphics A/S will offer you (free of charge) the complete machine-readable "work that uses the Library", as object code, so that you can modify this file and then relink to produce a modified executable containing the modified file.

If you have any questions regarding the use of this file or want to accept the offer given above, please contact Draupner Graphics A/S at touchgfx-info@draupnergraphics.com.

The Image Converter uses the following library for reading png images
libpng version 1.6.7 - November 14, 2013

- Copyright (c) 1998-2013 Glenn Randers-Pehrson
- Copyright (c) 1996-1997 Andreas Dilger
- Copyright (c) 1995-1996 Guy Eric Schalnat, Group 42, Inc. This code is released under the libpng license http://www.libpng.org/pub/png/src/libpng-LICENSE.txt.

9.5 Text Converter

The Text Converter is the utility program used for generating source code to include texts into the target and simulator source code.

The licenses for the Font Converter, for which Draupner Graphics is the copyright owner, is as follows:

Copyright (C) 2014 Draupner Graphics A/S http://www.touchgfx.com.
All rights reserved.

TouchGFX is protected by international copyright laws and the knowledge of this source code may not be used to write a similar product. This file may only be used in accordance with a license and should not be redistributed in any way without the prior permission of Draupner Graphics.

This is licensed software, any use must strictly comply with the signed license agreement and associated Terms & Conditions.

Standard Terms & Conditions can be seen on www.touchgfx.com

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Draupner Graphics A/S has no obligation to support this Software. Draupner Graphics A/S is providing the Software "AS IS", with no express or implied warranties of any kind, including, but not limited to, any implied warranties of merchantability or fitness for any particular purpose or warranties against infringement of any proprietary rights of a third party.

Draupner Graphics A/S can not be held liable for any consequential, incidental, or special damages, or any other relief, or for any claim by any third party, arising from your use of this Software.

9.6 Manual

The licenses for this Manual, for which Draupner Graphics is the copyright owner, is as follows:

Copyright (C) 2014 Draupner Graphics A/S http://www.touchgfx.com.
All rights reserved.

TouchGFX is protected by international copyright laws and the knowledge of this source code may not be used to write a similar product. This file may only be used in accordance with a license and should not be redistributed in any way without the prior permission of Draupner Graphics.

This is licensed software, any use must strictly comply with the signed license agreement and associated Terms & Conditions.

Standard Terms & Conditions can be seen on www.touchgfx.com

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Draupner Graphics A/S has no obligation to support this Software. Draupner Graphics A/S is providing the Software "AS IS", with no express or implied warranties of any kind, including, but not limited to, any implied warranties of merchantability or fitness for any particular purpose or warranties against infringement of any proprietary rights of a third party.

Draupner Graphics A/S can not be held liable for any consequential, incidental, or special damages, or any other relief, or for any claim by any third party, arising from your use of this Software.

Chapter 10

Advanced

This page contains a number of advanced topics relating to inner workings of TouchGFX. It is not mandatory reading and certainly not intended to be read in its entirety by everyone using TouchGFX. Instead, the chapters here will be referenced from other places of the documentation where relevant, such as from the Porting Guide.

10.1 Bitmap Caching

TouchGFX provides a mechanism for caching bitmap data in volatile memory. The memory area where the cache is placed will usually be the external RAM where the frame buffers are also placed. There are two main scenarios in which bitmap caching is useful:

- On platforms where accessing the external flash is slower than accessing the external RAM, thereby offering performance increase by keeping a cached copy of select bitmaps on particularly taxing screens.
- On platforms where the external flash is not memory-mapped, which means that all bitmaps must be cached. This is the case when using e.g. an SD card, a NAND flash or a serial flash to store bitmap data.

In the first case, only select bitmaps would be cached. In the second case, all bitmaps in the application must be cached, because TouchGFX is not able to render from locations that cannot be accessed in a memory mapped fashion, e.g. through a pointer.

Note

Bitmap caching is not supported for 1BPP monochrome.

10.1.1 Configuring the Bitmap Cache

There are a few things which need to be configured before the bitmap caching mechanism can be used on your platform. This is described in the chapter Non-memory mapped external flash / Supporting bitmap caching of the Porting Guide.

10.1.2 Using the Bitmap Cache

With the bitmap caching mechanism properly configured, it is ready to be used in applications. For caching a single bitmap, simply call the function touchgfx::Bitmap::cache. If this function returns true, the bitmap is now cached, and all subsequent renderings of this bitmap will read from cache memory instead of the flash storage. A good place to call this function would be the setupScreen function of the View which needs to display this bitmap. If the bitmap was already cached, the cache function will return instantly so it is safe to call every time a View is entered.

If all bitmaps must be cached because they are located in a non-memory-mapped flash type, use the function touchgfx::Bitmap::cacheAll. This function might take some time (several seconds) to execute depending on the

amount of bitmaps and the memory transfer speed. It is therefore often desirable to create some sort of boot screen displaying a progress indicator while the caching operation takes place. To implement a progress indicator, you should call the cacheAll function from another task with lower priority than the GUI task (as it will otherwise freeze the UI until the caching operation is complete).

10.2 Changing the Location of Bitmap Data

The default behavior is that all bitmaps are placed in external flash. This is because bitmap data take up a lot of space, and most applications will have more bitmaps than fits in the internal flash. The placement of bitmap data is managed by a linker section placement attribute which is affixed to each auto-generated array of pixel data.

```
LOCATION_EXTFLASH_PRAGMA
KEEP extern const unsigned char _down_btn[] LOCATION_EXTFLASH_ATTRIBUTE = { // 130x56 RGB565 pixels.
```

The LOCATION_ symbols above are compiler specific defines which translates to a way of labelling the data array with a section name called ExtFlashSection. Thus the linker script can be used to control whether bitmap data should be located in internal or external flash.

There is also an option for more fine grained control of the placement of each individual bitmap. If, for instance, one particular bitmap should be placed in internal flash because it is used in a very performance-critical animation, the bitmap file name can be changed to contain ".int.":

```
logotexturemap.int.png
```

This causes the pixel data array for this image to *not* contain a linker placement directive, so that it instead will be placed in the default readonly data segment (typically internal flash):

```
KEEP extern const unsigned char _logotexturemap_int[]  = { // 152x154 RGB565 pixels.
```

10.3 Timing and TFT Controller Integration

For performance reasons, the timing of TouchGFX is strongly connected with the operation of the TFT controller. Therefore in order to understand TouchGFX timing, we must first look into how a TFT controller operates.

The TFT controller on an MCU is responsible for updating the display glass with the pixel values found in the frame buffer. The display glass itself does not have any memory, so the pixel values must be transmitted to the display continuously, regardless of whether or not the frame buffer contents change. So even when displaying a still image, the frame buffer is being transferred to the display.

The display update frequency varies among displays, but it is usually in the 50-60 Hz range. That means, every 16-20 milliseconds the frame buffer contents are being transferred. The transfer usually happen in bursts with a shorter duration than the update frequency, leaving some idle time between each update where the memory holding the frame buffer can be accessed for other means (like, for instance, writing new pixel values in the frame buffer).

The following figure shows the update cycle of a TFT controller on a 320x240 pixel display:

Figure 10.1: Timing Diagram of TFT controller

The porch areas are delays where no data is being sent. Pixel data is transferred one pixel at a time, starting from the top left corner of the display (pixel 0,0) traversing horizontally, with delays before and after each line (horizontal porches) and delays before first line and after last line (vertical porches). For every display update (frame) a VSYNC signal is issued, and for each line in that frame, a HSYNC signal is issued. Whenever the controller is inside the active area, the SDRAM containing the frame buffer is busy. In order to optimize SDRAM access, the controller signals are directly connected to when TouchGFX performs its rendering, as explained in the following.

10.3.1 Rendering Pass

A frame rendering pass begins when the TFT controller enters the active area. The rendering pass consists of the following steps:

1. Call the virtual beginFrame function
2. Call Application::handleTickEvent function, which will tick timer widgets and the active view. This step may cause invalidation of the screen.
3. Sample touch and forward touch events to the application. This step may also cause invalidation.
4. Sample keys and buttons if appropriate and forward these to the application. This step may also cause invalidation.

5. Coalesce invalidated areas and begin actual drawing where needed

6. Wait for drawing to be completed

7. Call the virtual endFrame function

10.3.2 Frame Rate

If the time it takes to render the screen is less than the display update rate, the actual frame rate of the system will match the display update rate. This will usually happen if nothing or only a small portion of the screen needs to be drawn. If, however, the drawing takes longer, the actual frame rate will decrease because the next frame rendering pass will not happen the next time the TFT controller enters the active area, since the previous drawing was still underway. The following figure outlines the timing associated with a typical rendering pass, spanning four frames, where all rendering passes take less time than the display update frequency.

Figure 10.2: Timing diagram showing four frames

What you see above is that the GUI task begins its frame rendering immediately when the TFT controller enters the active area (ie. begins updating the display), labeled a). The GUI task will sample touch and buttons, and call tick on the active view and any widgets that have registered for timer events. In this example, this causes an invalidation of a region of the screen. At b), rendering is completed, and the GUI task can now sleep. At c), the TFT controller begins updating the display (entering the active area for the next frame). Here several things happen: since we have redrawn parts of the screen, we swap frame buffers, such that the TFT controller will use the frame buffer in which we just drew as base for updating the display in this cycle. Additionally, the GUI task wakes again and begins the next frame rendering pass. In this example figure, no screen changes were required in this pass, so the GUI task will quickly sleep (d) and the system idles until next frame. You will also note that no frame buffer swap takes place at next display update (e) because nothing was drawn.

If the screen area that needs to be drawn is large, it is quite possible that the drawing operations take longer than the display update frequency, as illustrated here:

Figure 10.3: Example where redrawing takes longer than VSYNC frequency

Here you will notice the rendering process is still underway when the next TFT controller update begins (a). Since the previous frame was not complete, this event is ignored so no new frame rendering pass begins here. The frame buffer swap will occur at next TFT update after the rendering is completed (b). In this example, the *effective* frame rate (ie. the frequency with which the pixels on the display actually change) is VSYNC/2. If the drawing operation took a very long time, it might even span across three or four updates. This is not a problem if it only happens on occasions such as switching to a different view. But if it happens during an animation that is supposed to look fluent the effective frame rate will be too slow.

Because the frame buffer swapping takes places at the next display update after drawing is complete, it may be necessary to adjust the VSYNC frequency in order to obtain the optimal timing. Consider the following example:

Figure 10.4: Example where VSYNC frequency is suboptimal with regards to drawing

What you see above is that rendering completes immediately *after* the display begins updating (a). At this point the frame buffer swap cannot occur because the TFT controller is in the middle of updating the display. Therefore the swap cannot take place until next update, which means that the pixel update will not be visible until next update (b). This yields an effective frame rate of VSYNC/3.

In such a scenario it would actually be better to slow down the TFT update frequency e.g. by extending the vertical porches. The following figure illustrates the same drawing operation, but with a slightly lower VSYNC frequency:

Figure 10.5: Same scenario with slower VSYNC frequency, yielding better results

The porch delay area has been slightly extended (a). Now the rendering completes immediately *before* a TFT update, thus allowing the frame buffers to be swapped earlier, yielding a better effective frame rate.

Because of this, the optimal VSYNC timing of your system can be hard to determine before the UI has been implemented. A good rule of thumb is to aim for a 16-20ms update frequency initially, and investigate the actual rendering times of your application to fine tune.

10.3.3 Interleaving MCU- and DMA-based Rendering

For the sake of simplicity, all the examples above simply show the rendering time and does not distinguish whether the rendering is done by DMA or MCU. TouchGFX will attempt to use the DMA as much as possible in order for the MCU load to remain low. Depending on your target microcontroller and the composition of your user interface, it is not uncommon for the GUI task itself to also perform rendering, however, for instance when drawing text or alpha blended images if this is not supported by the DMA engine. See Blitting Operations and Hardware Blit Capabilities for details on hardware support. If software rendering is involved, a more detailed timing diagram is as follows:

Figure 10.6: Example where the GUI task performs some rendering operations

In the above figure we see the GUI task wake up (a) and begin processing the frame (sampling touch, handling timer ticks, create a queue of DMA operations). When this is done, the DMA takes over (b). We then see the GUI task performing MCU based rendering at c), and with subsequent interleaved DMA and MCU based drawing for the

remainder of the rendering process.

The synchronization of the interleaving process is built into TouchGFX and controlled by a frame buffer semaphore, which allows the GUI task and the DMA to alternately perform drawing operations in the frame buffer.

10.3.4 Number of Frame Buffers

TouchGFX can be configured to use either a single frame buffer or double buffering. This configuration is done through the HAL::setFrameBufferStartAddress function which is usually called in your board configuration file. The default operation is to use double buffering.

We strongly recommend using double-buffering even though it takes up additional external memory space. The reason is that if using single buffering, *all* drawing operations must be able to complete within a single TFT update cycle. If it takes longer, the frame buffer contents are being changed while at the same time being transferred to the display by the TFT controller. The result of this will be visible artifacts on the screen such as tearing. The TouchGFX rendering algorithm has been specifically designed such that it has a tendency to draw the upper parts of the screen first, regardless of how the UI is composed. This may help in the sense that a short overlap in most cases will not cause tearing, but is not really something to count on. Even so, screens with much alpha blending will be close to impossible to render in a single frame.

The only two occasions where single buffering might be of use are:

- If the resolution is small enough to fit the frame buffer in internal RAM. The rendering time will in this case normally be fast enough to complete in a single update cycle, even for complex screens with alpha blending.
- If the hardware platform simply does not have enough external RAM for two buffers. But in this case you should probably be prepared to accept limitations in UI design.

As mentioned, when using single buffering, it is not allowed to draw in the frame buffer while the TFT controller is in the active area. This is handled by TouchGFX by waiting for a TFT controller signal indicating that the active area has been exited before starting any DMA or MCU based rendering, as illustrated here:

Figure 10.7: Single-buffered rendering scenario

Also in single-buffering mode, the rendering pass begins when TFT controller enters active area. But nothing will actually be drawn until TFT controller is done updating (b), where the DMA queue will begin transferring. The frame rendering completes at c), well before the next display update, so no glitches appear. A second frame begins at d), where more complex graphics were drawn so the rendering pass took longer and did not complete in time (e). This frame would likely cause graphics glitches, and it would be necessary to either speed up the rendering, increasing porch timing or switching to double buffering to alleviate this.

10.4 Animation Storage

In TouchGFX, animation storage is an optional storage placed in external RAM along with the frame buffer(s). If used, it will take up the same amount of memory as a frame buffer (ie. displayWidth ∗ displayHeight ∗ pixelsizeIn↩ Bytes). The purpose of the animation storage is to provide memory for taking a snapshot of the current contents of the frame buffer and use it for certain animations. Currently it is only used when employing SlideTransitions where a snapshot of the "old" view is used in the sliding animation. If no animation storage is allocated, SlideTransition will behave like NoTransition, ie. an instant screen switch without animation.

The configuration of whether to enable animation storage is done through the HAL::setFrameBufferStartAddress function. The default behavior is enabled.

10.5 Blitting Operations and Hardware Blit Capabilities

In the context of TouchGFX, blit capabilities refer to which types of blitting operations the hardware is able to perform. This varies considerably across hardware platforms and mainly depend on how advanced the DMA controller on the MCU is. The hardware abstraction layer must define which blitting operations the hardware is able to carry out natively. Those that are not supported directly by hardware, will automatically be rendered using built-in software fallback routines instead.

Note

It will always mean higher MCU load, and often also slower rendering times to use the software routines.

The blitting operations used in TouchGFX are:

Name	Description	Usage
BLIT_OP_COPY	This blitting operation refers to a basic copy of data from e.g. an image to the frame buffer. It is the most common operation in TouchGFX and is used when rendering an image that does not use per-pixel-alpha, contains no transparent pixels, and the source and destination formats match. Since it is, essentially, a memcpy(), this operation will almost always be supported by the hardware.	An Image widget displaying an RGB565 bitmap
BLIT_OP_FILL	This blitting operation refers to a basic fill of a single color value to a region, e.g. rendering a solid color box to the frame buffer. This operation is also supported on most hardware.	A Box widget
BLIT_OP_COPY_WITH_ALPHA	This operation is similar to BLIT_OP_COPY, but where a specific alpha blend factor is applied to each pixel. Requires an advanced DMA controller in order to support this operation in hardware.	An Image widget displaying a standard RGB565 bitmap but with an alpha value set on the widget.

BLIT_OP_FILL_WITH_ALPHA	This operation is similar to BLIT_OP_FILL, but where a specific alpha blend factor is applied to each pixel. Requires an advanced DMA controller in order to support this operaetion in hardware.	A Box widget with an alpha value set.
BLIT_OP_COPY_WITH_TRANS↩ PARENT_PIXELS	This operation is a basic copy, but where a certain pixel value means transparency (i.e. that pixel should be skipped). This operation has been superseded by the more powerful per-pixel-alpha operations and is therefore not used. There is no reason to implement support for this blit operation.	An Image widget displaying an RGB565 image containing pixels of color touchgfx::LCD16bpp::TR↩ ANSPARENT_COL
BLIT_OP_COPY_ARGB8888	This operation is a data copy where a format conversion is being applied for each pixel. In this case, source data is a 32-bit per pixel value (with an 8 bit alpha channel). This must be downscaled to 16 bits and blended with the background pixel color according to the alpha channel. Requires an advanced DMA with format conversion capabilities to support in hardware.	An Image widget display an ARGB8888 bitmap (image converter output format ARGB8888)
BLIT_OP_COPY_ARGB8888_↩ WITH_ALPHA	As above, but also apply a generic alpha blend to each pixel.	An Image widget displaying an ARGB8888 bitmap and which also has an alpha value set.
BLIT_OP_COPY_A4	This operation is used for rendering data represented as a 4-bit alpha channel only (i.e. no color information). Requires an advanced DMA with format conversion capabilities in order to support this operation in hardware.	Used for rendering text, when glyph data format is configured to be 4bpp in the text database.
BLIT_OP_COPY_A8	As above, with an 8bpp alpha channel.	Used for rendering text, when glyph data format is configured to be 8bpp in the text database.

10.6 Languages and Chars

TouchGFX supports displaying text in many different languages and charsets. There are however some limitations, which means that not all languages are supported out of the box.

10.6.1 Glyph Limitations

Due to the encoding scheme used by TouchGFX, we only support displaying glyphs whose Unicode code point is below 65536 (U+FFFF).

10.6.2 Text Limitations

- Text direction: TouchGFX supports only left-to-right writing direction.

- Complex Text Layout (e.g. context-sensitive shaping, digraphs and presentation forms) are not supported.

10.6.3 List of Supported Languages

TouchGFX products have shipped with the following languages:

- Bosnian
- Bulgarian
- Chinese
- Croatian
- Czech
- Danish
- Dutch
- English
- Estonian
- Finnish
- French
- German
- Hungarian
- Italian
- Japanese
- Latvian
- Lithuanian
- Norwegian
- Polish
- Portuguese
- Russian
- Romanian
- Serbian
- Slovenian
- Slovak
- Spanish
- Swedish
- Turkish
- Ukrainian

10.6.4 Unsupported Languages

The following languages are known to be unsupported because they rely on right-to-left writing direction and/or make extensive use of ligatures and digraphs:

- Arabic
- Hebrew
- Thai

10.7 Performance Measurement

Two methods

10.7.1 GPIO

The touchgfx::GPIO class, compiled for all target applications, contains methods to `set`, `clear`, `toggle` and `get` the state of a GPIO pin. An implementation must be provided for these static members; Even if it is simply empty and does nothing. While the TouchGFX core might not be available to you, our code is instrumented to make calls into this GPIO implementation when it performs certain functions to help developers track down performance and timing issues. The following types are toggled throughout TouchGFX:

1. `VSYNC_FREQ` is typically toggled from the HAL component of an MCU
2. `RENDER_TIME` defines the length of a frame, and is toggled from HAL base class
3. `FRAME_RATE` defines the time it takes to swap framebuffers and is toggled from the HAL base class upon swap requests
4. `MCU_ACTIVE` is toggled from the idle task hook in the touchgfx::OSWrappers class for an OS package as soon as the idle task is scheduled

You can perform measurements with an oscilloscope by toggling the GPIO pins of choice in your implementation of the GPIO class. You can gain inspiration from existing GPIO implementations for various boards found under the `touchgfx/board` folder.

10.7.2 MCU Instrumentation

The class touchgfx::MCUInstrumentation provides an interface for reading out CPU-cycles in an effort to, for instance, calculate general MCU-load caused by an application. The following items are required in order to enable HAL to perform MCU load calculations:

1. Configure OS to set MCU active when Idle Task is switched out during rescheduling.
2. Configure HAL with a concrete instance of `MCUInstrumentation` for a particular processor (For Cortex-M: touchgfx::CortexMMCUInstrumentation located under `platform/core/arm/cortex-m`)

10.7.3 HAL

The following code snippet from a BoardConfiguration shows how to configure HAL to use MCU instrumentation.

```
.
.
.
CortexMMCUInstrumentation mcuInstr;
```

```
void touchgfx_init()
{
  HAL& hal = touchgfx_generic_init<STM32F4HAL>(dma, display, tc, 480, 272, 0, 0);
  hal.setFrameBufferStartAddress((uint16_t*)frameBuf0);

  mcuInstr.init();

  //Set MCU instrumentation and Load calculation
  hal.setMCUInstrumentation(&mcuInstr);
  hal.enableMCULoadCalculation(true);
 }
```

10.7.4 OS

The following snippet shows how `OSWrappers.cpp` (Wrapper for FreeRTOS7.6.0), through a task hook, is able to indicate when the MCU is active and when it is Idle. We use the FreeRTOS method `vTaskSetApplicationTaskTag` to callback function and assign our function pointer `IdleTaskHook` as the task tag parameter.

```
static portBASE_TYPE IdleTaskHook(void* p)
{
  if ((int)p) //idle task sched out
  {
    touchgfx::HAL::getInstance()->setMCUActive(true);
  }
  else //idle task sched in
  {
    touchgfx::HAL::getInstance()->setMCUActive(false);
  }
  return pdTRUE;
}

// FreeRTOS specific handlers
extern "C"
{
  void vApplicationStackOverflowHook( xTaskHandle xTask,
                                      signed portCHAR *pcTaskName )
  {
    while(1);
  }

  void vApplicationMallocFailedHook( xTaskHandle xTask,
                                     signed portCHAR *pcTaskName )
  {
    while(1);
  }

 void vApplicationIdleHook( void )
  {
    vTaskSetApplicationTaskTag( NULL, IdleTaskHook );
    while(1)
    {
    }
  }
```

Enabling the *Idle-hook* is an explicit setting found in `FreeRTOSConfig.h` for a specific application. The following snippets were taken from `app/demo/touchgfx_demo2014_480x272/platform/os/FreeRTOSConfig.h`:

```
#define configUSE_IDLE_HOOK                     1
```

The following define configuration is required to call the idle hook when a task is switched out.

```
#define traceTASK_SWITCHED_OUT() xTaskCallApplicationTaskHook( pxCurrentTCB, (void*)1 )
#define traceTASK_SWITCHED_IN() xTaskCallApplicationTaskHook( pxCurrentTCB, (void*)0 )
```

10.8 Platform Abstraction

The folder `platform/core` contains a vendor-independent hardware abstraction layer for Cortex-M processors (`CMSIS - Cortex Microcontroller Software Interface Standard`). Also in `core` lies source allowing retrieval of elapsed CPU cycles and measurement of CPU load in TouchGFX applications. For more information on how to enable MCU instrumentation, see MCU Instrumentation. `driver` contains generic

driver interfaces used by *board packages* (See Board Packages (touchgfx/board/)) to configure platform peripherals and the `hal` folder contains MCU specific TouchGFX ports - See Porting Guide.

```
+-- platform
|    +-- core
|    |-- driver
|    |      +-- button
|    |      |-- i2c
|    |      \-- touch
|    |
|    \-- hal
|         +-- Freescale
|         |-- nxp
|         |-- simulator
|         \-- ST
```

TouchGFX offers a small range of generic, reusable driver interfaces for configuration of hardware peripherals.

- `button` contains generic button controller interfaces. Concrete implementations are practically always board-specific.
- `i2c` contains generic I2C driver interfaces. Concrete implementations are always MCU specific.
- `touch` contains generic touch controller interfaces. Concrete implementations are generic and can be used with any board that utilises a supported touchcontroller.

10.8.1 Button Controllers

Button Controllers (touchgfx::ButtonController) can be used to add hardware button support to TouchGFX applications. A concrete implementaiton of a Button Controller is typically located under the Board Support `bsp` folder in a board configuration. The folder `button` contains a single interface description touchgfx::ButtonController that exposes methods for initializing and sampling. The `sample()` method touchgfx::ButtonController::sample returns whether or not a keypress was detected, and stores the numerical value for the key that was pressed in an 8-bit reference.

During touchgfx::HAL::tick a configured Button Controller is used to sample for pressed buttons and listeners are notified. In the pseudo example below, a `KeySampler` tests for the status of 5 buttons. Reading the state of an appropriate GPIO port through the `Buttons` class gives us a button state sample that can be used by the `sample` method.

```
#include "KeySampler.hpp"
#include "touchgfx/HAL/Buttons.hpp"

using namespace touchgfx;

void KeySampler::init()
{
  Buttons::init();
}

bool KeySampler::sample(uint8_t& key)
{
   unsigned int buttons = Buttons::sample();

   //Return true if button was pressed, save 1 to key argument
   //Return true if button was pressed, save 2 to key argument
   //Return true if button was pressed, save 3 to key argument
   //Return true if button was pressed, save 4 to key argument
   //Return true if button was pressed, save 5 to key argument

   return false;
}
```

Configuring HAL with a button controller using touchgfx::HAL::setButtonController will cause sampling for pressed buttons every tick.

10.8.2 i2c

The `i2c` folder contains a generic interface for concrete I2C driver implementations in TouchGFX (See touchgfx::I2C). I2C drivers are always MCU specific and currently TouchGFX supports generic I2C drivers for NXPs LPC43XX (touchgfx::I2C_LPC43xx) and LPC1788 (touchgfx::I2C_LPC1788) series. In general, only the `readRegister()` method is used for touch controller purposes.

10.8.3 Touch Controllers

The `touch` folder contains generic interfaces for touch controllers (touchgfx::TouchController) in TouchGFX.

During touchgfx::HAL::tick, the configured touch controller is sampled touchgfx::TouchController::sampleTouch and coordinates are translated into click and drag events internally.

The `Sitronix ST1232 Touch controller` `include/platform/driver/touch/ST1232TouchController.hpp` is a type of touchgfx::I2CTouchController used on a number of boards supported by TouchGFX. It takes an MCU specific I2C driver implementation as argument, as well as the height and width of the display. Specifying an additional correction type will after-process the coordinates read from the touch controller. The ST1232 touch controller provides two methods of correcting/altering the coordinates read from the registers of the touch controller.

- Transformation by type ("Flip X" and "Flip Y")
- User defined transformation

For type transformation, the driver supplies the following types:

```
typedef enum
{
  NO_TRANSFORM = 0,
  FLIP_X_AXIS,
  FLIP_Y_AXIS
} TransformationType;
```

The following snippet initializes an I2C driver for NXPs LPC43XX series and returns coordinates flipped around the X axis to offset native. Users must make sure that appropriate coordinates are read back to TouchGFX HAL. (calls `sampletouch()`).

```
I2C_LPC43xx i2c(0);                                    //Initialize I2c channel 0
ST1232TouchController tc(i2c, 480, 272, FLIP_X_AXIS);   //Initialize touch controller with i2c driver, and
       native display dimensions. Default transformation is NO_TRANSFORM
```

If instead we'd like to perform a custom transformation that does the same thing as FLIP_X_AXIS, the ST1232 touch controller contains a function definition as follows that allow users to define function pointers and manipulate coordinate data before it's returned to TouchGFX HAL. The code also shows how to the custom transformation for FLIP_Y_AXIS.

```
//Function definition for touch correction
typedef void (*TransformFuncPtr)(int32_t&, int32_t&, Finger&);
```

In our board configuration, we'd then define our own transformation function as follows, making appropriate calculations based on screen dimensions (in this case 480 x 270).

```
/* Callback function used for applying a correction strategy to mis-aligned touch coordinates */
static void applyTransformation(int32_t& x, int32_t& y, Finger& f)
{

  // X-axis flipped
```

```
  /****************************************
   * (0,272)                      (480,272)


   * (0,0)                          (480,0)
                                         */
   y = 272 - f.y;

  /*******************************/
  /* Other linear permutations */
  /*******************************/

   // Y-axis flipped
  /****************************************
   * (480,0)                          (0,0)


   * (480,272)                      (0,272)
                                         */
   //x = 480 - f.x
}
```

And on our touchcontroller we'd then set the callback method to our custom method.

```
tc.setTransformCallback(applyTransformation);
```

10.8.4 HAL

The `hal` folder contains MCU specific TouchGFX ports primarily related to LCD- and DMA controller setup. The general structure for each folder under `hal/mcu` follows this general set of rules:

- `driver` contains MCU specific drivers (e.g. I2C drivers) that can be used in conjunction with touch controller drivers. As we saw previously, we could use MCU specific I2C drivers to plug into existing touch controllers like the Sitronix ST1232/1233
- `vendor` contains vendor supplied code for the MCU
- `{MCU_ID}DMA.hpp` contains the DMA implementation for the MCU (See Porting Guide)
- `{MCU_ID}HAL.hpp` contains the HAL implementation for the MCU (See Porting Guide)

```
+-- {mcu vendor}
|     \-- mcu
|         +-- {MCU_ID}
|         |      +-- driver
|         |      |-- vendor\
|         |      |-- {MCU_ID}DMA.hpp
|         |      \-- {MCU_ID}HAL.hpp
```

The MCU packages (HAL, DMA) that are preinstalled with TouchGFX can be seen in the following table, which also lists the evaluation boards that use a specific MCU package.

Package Name	MCUs Supported	Compilers	Evaluation Boards
NXP1788	LPC1788	IAR, GCC	FDI uEZGUI-1788-70WVT
NXP18xx	LPC1857	IAR, GCC	None
NXP43xx	LPC4350,LPC4353,LP↩ C4357	IAR, KEIL, GCC	TouchGFX demo board 4.3", Flashless, 7.0", LPC4357DevKit

STM32F4x9	STM32F429, STM32F439	IAR, GCC	ST STM32429I-EVAL1, STM324x9I-EVAL-5.7, STM32F429I-DISCO
FreescaleK70	Freescale K70	IAR	None

10.9 Target Configurations

Functioning TouchGFX buildconfiguration for specific targets require a number of things outlined in the table below.

Resonsibility	Artifact
Code that initializes OS	target/main.cpp
Hardware initialization code	Boardpackage
Software initialization code	Boardpackage

Remembering what we learned from RT Operating System (touchgfx/os/) and Board Packages (touchgfx/board/) and keeping in mind the requirements above we can now initialize software, hardware and OS in a single, target-independent file, `main.cpp`. The code is identical to the other example applications because the requirement to stack size is the same for all applications.

```
#include <touchgfx/hal/HAL.hpp>
#include <touchgfx/hal/BoardConfiguration.hpp>

using namespace touchgfx;

/* Kernel includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"

#define configGUI_TASK_PRIORITY                 ( tskIDLE_PRIORITY + 3 )
#define configGUI_TASK_STK_SIZE                 ( 950 )


//Define GUI task
static void GUITask(void* params)
{
  touchgfx::HAL::getInstance()->taskEntry();
}


int main (void)
{
  //Initialize TouchGFX and Hardware
  hw_init();
  touchgfx_init();

  //Create task
  xTaskCreate( GUITask, (signed char*)"GUITask",
               configGUI_TASK_STK_SIZE,
               NULL,
               configGUI_TASK_PRIORITY,
               NULL);

  vTaskStartScheduler();

  for(;;);
}
```

IAR and GCC buildconfigurations, for all or some target boards, are already provided for all TouchGFX applications and utilize one of the available boardpackages for any hardware target specified. An IAR workspace would include the files of both board and OS packages for compilation. The image below shows an IAR configuration that embodies the three responsibilities outlined above.

Figure 10.8: IAR Build Configuration

For a more complete overview of how a project is put together, refer to one of the IAR (.eww) workspace files or one of the GCC buildfiles. Take note of how each build configuration makes use of compiler and target specific (Cortex-M3 / Cortex-M4F) portable code.

10.10 Configuring TouchGFX for 1BPP Monochrome

The default behavior of TouchGFX is to represent graphics in 16 bit RGB565 format. It does, however, come with built in support for monochrome, 1 bit per pixel user interfaces. When running in this mode, the frame buffer(s) only take up one sixteenth of the space required for the RGB565 color format. Likewise, all the generated image data also only take up one bit per pixel. Therefore it is quite likely that 1BPP applications can run on devices with no external memories by having the frame buffer in internal RAM and the image data in internal flash. In order to configure TouchGFX for 1BPP operation, the following needs to be done:

10.10.1 1BPP Asset Generation

Both image data and fonts must be generated in a 1BPP format in order to be displayed properly when running in 1BPP mode.

10.10.1.1 Image Data

In order to make the image converter output data in 1BPP format, simply modify the config/gcc/app.mk file in your application to specify this

```
# Settings for image converter output format
opaque_image_format := BW
non_opaque_image_format := BW
```

And likewise for Visual Studio in the config/msvs/Application.props file, if using Visual Studio to generate assets.

This will cause the image converter to output image data in 1BPP. The .png/.bmp image files used as input for the converter must still be in regular 24bpp format. The image converter itself will do the downscaling to monochrome. If the bitmap contains colored pixels (not just black or white), the resulting output will be based on the luminance of the pixel.

10.10.1.2 Compressed Bitmaps

Monochrome images can alternatively be stored in a horizontal run-length encoded format. This will in many cases substantially reduce the amount of flash memory each image takes up, and is therefore worth investigating if flash memory is scarce. If compression is enabled, the imageconverter will automatically calculate the resulting size for each bitmap and only use the run-length encoded format if the size is smaller than the normal unpacked format. Therefore, run-length encoding has a worst case size equal to that of uncompressed bitmaps. The drawback to using compression is that the rendering speed will decrease somewhat for compressed bitmaps because the format must be decoded by TouchGFX before drawing. You must therefore decide this tradeoff for your specific application based on your requirements. In order to enable bitmap compression, use the image format BW_RLE instead of BW:

```
# Settings for image converter output format
opaque_image_format := BW_RLE
non_opaque_image_format := BW_RLE
```

10.10.1.3 Font Data

In order to specify the use of 1BPP fonts, simply modify all typographies used in the Texts.xlsx Excel sheet to use 1 bit per pixel.

10.10.2 Board Configuration

The BoardConfiguration.cpp for the appropriate target hardware (or simulator/main.cpp for PC simulator) file must be changed to use a 1-bit LCD object instead of the standard 16-bit type:

```
//LCD16bpp display;

//Monochrome 1BPP mode
LCD1bpp display;
```

Finally, on target only, the setFrameBufferStartAddress function must be modified to inform the framework that the frame buffer(s) should use a bit depth of 1:

```
hal.setFrameBufferStartAddress((uint16_t*)SDRAM_BASE, 1);
```

Warning

> The use of DMA to transfer graphics data is not supported in 1BPP mode. You can change your DMA object in BoardConfiguration to be of type NoDMA for clarity, but the DMA object is ignored entirely in 1BPP mode.

With the changes above, your application now runs in 1BPP mode. This does not impact the API or features of TouchGFX compared to 16bit mode except in one area: alpha blending is ignored since it does not really make sense in a 1BPP context. There are some instances where colors are expressed in uint16_t (for instance in TextArea or Box widgets). In those cases, a value of zero is interpreted as black, where as any other value is interpreted as white.

See Monochrome Example for an example configured for 1BPP mode.

10.11 Switching Display Orientation

Most TFT displays is built for either landscape or portrait orientation. That is, the display will have a fixed starting point where the first pixel value received from the TFT controller is placed, and a fixed direction in which the following pixels are placed. Some displays are configurable in terms of this but that is not necessarily the case. In order to support both landscape and portrait orientation for user interfaces, TouchGFX has a built-in mechanism for rotating the UI such that e.g. a portrait UI can be realized on a display that natively runs in landscape mode. This rotation is applied at no performance cost so rendering times are not increased if your UI design does not match the native orientation of the display.

Figure 10.9: Example UI in landscape and portrait orientation

In order to achieve this, TouchGFX must be informed of what the *native* orientation of the display itself is. This is done in the BoardConfiguration, where the hardware abstraction layer is informed of the width and height of the display in pixels. If the width is larger than the height, the native orientation is inferred to be landscape.

If the user interface you are implementing has the same display orientation as your actual TFT display has, nothing special needs to be done since the default behavior is to match the orientation of the display itself. If, however, your UI is e.g. portrait and your display is landscape, you need explicitly change orientation at runtime by calling the function

```
//Switch to portrait mode.
HAL::getInstance()->setDisplayOrientation(ORIENTATION_PORTRAIT);
```

If your entire application uses the same orientation throughout, it is easiest to change the display orientation before the GUI task starts so you do not have to worry about it in your views. If your application supports both, please see Dynamic Switching at Runtime (supporting both orientations).

The necessary coordinate transformations take place behind the scenes. The width and height of the root Container of your view, touch input coordinates, and widget coordinates are all expressed in the coordinate space denoted by your chosen display orientation for the user interface. As such, there is really only one thing an application developer needs to be concerned with regarding display orientation, and that relates to bitmaps and is described in the following section.

10.11.1 Graphics Data in a Non-Native User Interface Orientation

If the orientation of your user interface is different from the native orientation of the TFT display, all graphics data must be generated with a specific flag. This will cause the binary pixel data to be represented in a different format on the target, which is what allows TouchGFX to render a rotated display without performance overhead. When configuring your project it is therefore necessary to know whether UI orientation differs from the native orientation. If that is the case, modify the config/gcc/app.mk to specify the following:

```
# Settings for image converter screen orientation (empty string = default value, -rotate90 rotates the
      image 90 degrees)
screen_orientation := -rotate90
```

And likewise for config/msvc/Application.props if using Visual Studio to generate assets:

```
<RotateImage90>true</RotateImage90>
```

To summarize, the rotate90 flag must be applied to your asset generation if one of the following is true

- Your application is in portrait mode and the display is natively in landscape mode, or
- Your application is in landscape mode and the display is natively in portrait mode

10.11.2 Dynamic Switching at Runtime (supporting both orientations)

This subsection only applies in the case where your application must be able to dynamically switch between portrait and landscape mode at runtime, for instance if the device is equipped with an accelerometer to detect rotation change. If both orientations are to be supported, then the bitmaps that are to be displayed in the orientation matching the native display orientation must be generated without rotation, whereas the bitmaps to display in the non-native orientation must be generated with the -rotate90 option. In order to achieve this, the TouchGFX asset generation mechanism has built in support for a per-bitmap setting of rotation instead of the global -rotate flags. Therefore, when having both orientations present, set the screen_orientation variable to an empty string (or -rotate0). We then need to override this setting for the bitmaps in non-native orientation, e.g. portrait bitmaps on a native landscape display. TouchGFX has a naming convention regarding bitmaps which means that the orientation can be overridden from the global setting by including ".90." in the file name, such that for instance a bitmap called

```
icon.90.png
```

will be generated with the -rotate90 switch automatically.

10.11.2.1 Performing the Switch

Because the root container of your view must be changed to the new dimensions when switching orientation, it is necessary to perform a screen transition after switching orientation. This transition can either be to a different view/presenter, or to the same view/presenter as being currently displayed. For instance:

```
#include <gui/common/FrontendApplication.hpp>
#include <touchgfx/hal/HAL.hpp>
void MyView::performOrientationChange(DisplayOrientation orientation)
{
  HAL::getInstance()->setDisplayOrientation(orientation); //set new orientation
  static_cast<FrontendApplication*>(Application::getInstance())->gotoMyScreen(); //switch screen (to the
      same screen we are currently displaying).
}
```

Whether or not to reuse the same screen for both orientations or to create landscape and portrait versions of a screen primarily depends on the amount of differences between the two. Usually it is only widget coordinates and bitmap IDs that differ, and therefore it is easier to use the same view/presenter and simply testing for current orientation in the setupScreen function, like this:

```
void MyView::setupScreen()
{
  bg.setXY(0, 0);
```

```
  if (HAL::getInstance()->getDisplayOrientation() == ORIENTATION_LANDSCAPE)
  {
    bg.setBItmap(BITMAP_BG_ID);
    icon.setBitmap(BITMAP_ICON_ID);
    icon.setXY(200, 20);
  }
  else //Portrait
  {
    bg.setBItmap(BITMAP_BG_90_ID);
    icon.setBitmap(BITMAP_ICON_90_ID);
    icon.setXY(40, 30);
  }

  add(bg);
  add(icon);
}
```

Note that in the example above, the native orientation is landscape which means that the bitmaps used in portrait mode have been suffixed with .90. as mentioned earlier.

Chapter 11

Changelog

```
4.5.0
=====

* Release date: February 2nd, 2016

* New features
  * Support for two new languages, Arabic and Hebrew, with
    right-to-left text rendering. RTL strings can be mixed with LTR
    texts and numbers.
  * Support for 24 bits per pixel frame buffers. Images look more
    detailed, but also consume more memory. See this article:
    https://touchgfx.zendesk.com/hc/en-us/articles/206725849
  * Bitmaps can now be created at runtime using method
    Bitmap::dynamicBitmapCreate. Useful for e.g. displaying .bmp files
    loaded from an SD card. See dynamic_bitmap_example and this
    article: https://touchgfx.zendesk.com/hc/en-us/articles/207460605
  * Frame rate compensation feature which provides smoother animations
    if frame rate occasionally drops. Not enabled by default. See
    article: https://touchgfx.zendesk.com/hc/en-us/articles/206430529
  * Bitmap caching is enhanced to allow removal of bitmaps from the
    cache to make room for caching of other bitmaps. See this article:
    https://touchgfx.zendesk.com/hc/en-us/articles/205953932
  * A new widget, PixelDataWidget, is introduced. This widget makes it
    possible to display raw pixel data obtained at runtime (e.g. video
    samples).
  * The simulator executable on windows now features an icon for
    easier identification in the task bar.
  * ST boards supported by TouchGFX can now have just their internal
    flash programmed from the command using 'make intflash' provided
    that ST-Link Utility Release 3.7+ is installed. For usage and
    troubleshooting, please refer to this article:
    https://touchgfx.zendesk.com/hc/en-us/articles/205264831
  * Unicode::snprintf() has been improved and updated substantially to
    support more of the standard format specifiers like %02d.
  * Unicode::snprintfFloat() added to support floats (in separate
    function because the "%f" va_args approach would force inclusion
    of doubles).
  * Quality of image converter dithering has been improved (floating
    point arithmetics). Also added support for new types of dither
    algorithms, and can take into account hardware with various wiring
    of the low (unused) bits in 16/18 bit displays.
  * touchgfx::ButtonWithLabel now contains a method,
    updateTextPosition(), that can be used to ensure horizontal text
    centering when changing label content (e.g. when changing
    language).
  * touchgfx::TextArea has a new method, setBaselineY(), that allows
    placing texts according to a text baseline instead of upper left
    corner.
  * The internal format of glyph encoding now stores the first pixel
    in the least significant bit instead of the most significant bit.
  * Specification of color values has been switched from uint16_t to
    colortype to support seamless switching between 16 and 24 bit
    colors.
```

```
  * The touchgfx::TextArea class now has a method, setIndentation(),
    that can prevent the glyph of characters from being cut off in the
    rare case where it extends under the previous character (similarly
    for touchgfx::Keyboard class which has a new setTextIndentation()
    method).
  * STM32F7xx and STM32F4x9 ports now support DMA transfers of
    touchgfx::Box.
  * The GPIO::VSYNC_FREQ signal was previously "toggled" exclusively
    on "VSYNC" interrupt (NXP LPC18xx, NXP LPC43xx, Freescale MK70F12,
    ST stm32f4x9). The signal is now high on "VSYNC" interrupt and low
    on "Front-Porch-Entered" interrupt.
  * GCC support for Cortex-M3.

* Bugfixes
  * Fixed rare crash on STM32F7 caused by speculative caching of
    invalid QSPI memory region. Update your BoardConfiguration if
    yours is based on 4.4.x.
  * Fixed occasional display flickering on STM32F746G-DISCO board
    caused by cache access on FMC bank 1.
  * Handling of the character "%"" in touchgfx::TextAreaWithWildcards
    has been improved to prevent inserting %% in some special cases.
  * touchgfx::DragEvent and touchgfx::GestureEvent now use and report
    signed coordinates instead of unsigned. This makes more sense as
    drags/gestures are expressed in coordinates relative to the
    drawable receiving them.
  * snprintf("%x") would generate uppper case hex. Now "%X" generates
    uppercase hex and "%x" generates lower case hex, just like the
    standard snprintf().
  * Fixed randomness for demos when running on Linux.
  * Fixed redrawing when using heavily italicized fonts.
  * Pointer to ModelListener in Model class for all TouchGFX
    applications was not properly initialized (NULL).
  * Fixed support for heavily italicized fonts in touchgfx::TextArea.
  * Subtle error in the Image Converter where column 0 could get
    slightly incorrect pixel colors. As a result the entire image
    could be slightly wrong, probably not noticeable.
  * Minor error in Slider where values were not distributed evenly.

* Deprecated
  * LCD::drawGlyph() has been deprecated. Use LCD::drawString instead.
* Update Procedure
  * Compatible with existing 4.x applications and HAL ports. Please
    refer to this article for details:
    https://touchgfx.zendesk.com/hc/en-us/articles/206819819


4.4.2
=====

* Release date: November 26th, 2015

* Bugfixes:
  * Corrected rare GUI task hangup on STM32F7 targets when compiling
    with IAR 7.x


4.4.1
=====

* Release date: October 27th, 2015

* Bugfixes:
  * Corrected occasional GUI task hangup on STM32F7 targets when
    compiling with Keil 5.x
  * Fixed occasional tearing on STM32 F469 EVAL/Discovery boards when
    using DSI in landscape orientation and single frame buffer mode.
  * Modified IAR flash loader settings for STM32 F469 boards to enable
    programming of internal flash (Note: QuadSPI flash must still be
    programmed from ST-Link Utility as there are no IAR loaders for
    this)
  * GPIO class for perf. measurement for STM32F746G-EVAL boards now
    properly uses the BSP_LED functions. Note that only two signals
    are active on this board per default because LED2 and LED4 use IO
```

```
    Expander, making them unsuited for measuring performance.
  * Removed annoying "Get Alternative File" dialog popups in IAR
    Workbench when debugging Cortex-M7 applications.


4.4.0
=====

* Release date: October 6th, 2015

* New features
  * Added support for the Cortex-M7 core.
  * Introduced concept of "finger size" for touch input. When used,
    TouchGFX will attempt to find touchable widgets in the area
    surrounding the reported x,y coordinates, so users no longer have
    to click precisely on a widget. This feature makes it substantially
    easier to hit small buttons.
    See HAL::setFingerSize().
  * Supports Visual Studio 2015
  * Visual Studio projects for Demos and Examples now include
    Application.props under Resources for quick access. As always a
    rebuild might be required when altering the contents of
    Application.props.
  * Support for Bitmap Fonts in BDF format. If the requested font size
    is not available in the font file, the font converter will write
    the supported font size(s) in the error message. See the example
    monochrome_example for usage.
  * Generating assets now issues better error messages when spaces are
    detected in paths and file names.
  * All ST boards can now be flashed from the command line provided
    that ST-Link Utility Release 3.7 has been installed. Simply use
    'make -f target/ST/<board>/Makefile flash' to build and flash
    your application to the connected board. If timeouts occur during
    flashing, go to Device Manager in Windows and disable "MBED
    microcontroller USB Device" under "Disk drives".
  * New touchgfx-env version 2.5 available with new gcc cross compiler
    version 4.9.3. The older version 4.8.4 could generate invalid code
    for Cortex-M7 cores in rare cases.

* Board support
  * Added support for the STM32F7xx processors
  * Added support for the STM32F746G-DISCO and STM32756G-EVAL boards
  * Added support for the STM32F469 processor with DSI displays
  * Added support for the STM32469I-EVAL and STM32469I-Discovery boards

* Bugfixes
  * TextureMapper and ScaleableImage now draws images correctly when
    using "rotate90".
  * Fixed potential initialization order bug in STM32F4DMA.cpp
  * Fixed bug that limited number of glyphs in a single font to 32768.
    Now supports 65536 glyphs per font as intended.
  * Fixed bug that caused hal.lockDMAToFrontPorch(false) to not have
    any effect in single frame buffer mode.
  * ButtonWithLabel correctly center texts vertically if text contains
    newlines


4.3.0
=====

* Release date: June 8th, 2015

* New features
  * TextureMapper widget added.
    The TextureMapper is a highly optimized image renderer that
    can be used for displaying an image that is scaled and/or rotated
    in two or three dimensions during run time. This can be used for
    doing advanced rotation animations of images.
    See manual or texture_mapper_example for more information.
    LCD has new methods for drawing triangles and corresponding scan lines,
    drawTextureMapTriangle and drawTextureMapScanLine
  * Alpha Channel Dithering
    Images with alpha channel can now get the alpha channel dithered
```

```
    for smoother alpha gradients, see examples or Application Development
    section in manual for details
  * Compression of 1BPP (monochrome) bitmaps
    Added image format option of BW_RLE, which will cause bitmaps to
    be automatically run-length encoded if that takes up less space than
    the regular per-pixel format. Yields substantially smaller bitmap
    footprint in many cases. See advanced chapter in manual for details.
  * Slider widget added.
    See manual or slider_example for more information.
  * Makefiles has been updated to work with make-4.1.
  * Added support for the LPC4088 processor and the Embedded Artists
    LPC4088 Display Module board.
  * Individual bitmaps can now be placed in internal flash instead
    of external by having the bitmap file name include the string ".int."
  * MoveAnimator, FadeAnimator and ZoomAnimationImage now have a
    cancelMoveAnimation/cancelFadeAnimation/cancelZoomAnimation
    method.

* Update procedure
  * Compatible with existing 4.X applications. Just replace the
    touchgfx folder.
  * Check Known Issues in the documentation.

* Info
  * The evaluation version of TouchGFX is now distributed with source
    code for the hardware abstraction layer instead of a precompiled
    library. This makes it possible to port the evaluation version
    to custom hardware instead of it being limited to the supported
    eval boards only. Instead, the evaluation version now has a
    TouchGFX watermark which will appear occasionally.
  * Memory consumption reduced due to improved rendering algorithm.
    Will typically allow GUI task stack to be reduced by around 1400
    bytes compared to version 4.2.0 (depending on actual application).
    Additionally the statically allocated memory is also reduced by
    around 1KB.
  * Maximum number of visible widgets limit of 150 removed.
  * Added two new demos for 640x480 and 480x272 resolutions showcasing
    new features, graphs, internationalization and custom widgets.
  * Drawable.setPosition() now calls setXY(), setWidth() and
    setHeight() for easier subclassing.
  * AbstractPainterRGB565 and AbstractPainterBW are recommended as
    base classes when implementing your own painters.
  * CanvasWidgets now have setAlpha() and getAlpha() methods. Your
    custom Painter classes must implement this, or inherit from the
    AbstractPainterRGB565 class
  * Maximum number of registered timer widgets increased from 16 to
    24.
  * touchgfx-env updated to 2.4. The environment does not beep anymore.
  * Board Support Package for STM324x9I-EVAL is now based on the
    STMCubeF4 drivers.

* Bugfixes
  * Screen::handleGestureEvent now converts x/y to relative coordinates
  * Fixed bug when drawing several objects on the same canvas using
    moveTo() more than once.
  * ZoomAnimationImage movement relative to scaling did not use
    correct easing equation.
  * PainterRGB565 did not blend green alpha correctly.
  * RadioButtonGroup now initializes callbacks to zero.
  * ScalableImage now works with bitmaps with transparancy.
  * AnimatedImage would display the start and end of an animation twice.
  * Default implementation of CanvasWidget::getMinimalRect() returned
    coordinates relative to its parent, not itself.
  * ScrollableContainer erroneously unregistered itself as a timer widget
    at every tick, which made it difficult to use with other timer-based
    operations.

* Performance
  * ScalableImage and ZoomAnimationImage has been optimized for better
    performance.


4.2.0
```

```
=====

* Release date: January 14th, 2015

* Performance
  * Substantially improved rendering performance, which in most cases
    will result in a 25% reduction of time it takes to render a frame.

    NOTE: This optimization does not necessarily work on all targets
    so it must be manually enabled. See the "Optimization" chapter in
    the porting guide for how to enable this optimization for existing
    portings.  It is STRONGLY recommended that the optimization is
    enabled.  This optimization is enabled for all appropriate
    evaluation boards in the 4.2.0 board packages.

* Major new features
  * Added CanvasWidgets for smooth, anti-aliased drawing of geometric
    shapes. Currently Line, Circle and a more generic Shape have been
    implemented. CanvasWidgets can be painted with a solid color (+
    alpha), a bitmap (including alpha) or a custom painter. Read more
    on Canvas Widgets and Painters in the documentation.
  * Added support for the Keil compiler and uVision4 IDE. Please refer
    to the "Supported Hardware" section of the TouchGFX Distribution
    chapter in the documentation for a list of Keil-supported targets.

* New features
  * It is now possible to specify an animation start delay on
    ZoomAnimationImage, MoveAnimator and FadeAnimator.
  * Added Board support for 4.3" TouchGFX Demo board w. LPC4350 (No
    internal flash)
  * RadioButton and RadioButtonGroup widgets added. See
    app/examples/radio_button_example and documentation.
  * LPC43XX and LPC1788 can now fill rectangles using DMA.
  * Visual Studio 2013 is now supported.
  * Preliminary support for Visual Studio 2015 Preview version.
  * Improved performance when generating assets.
  * New canvas_widget_example added to the example directory.
  * The "using namespace touchgfx" present in various header files can
    now be avoided by defining the symbol NO_USING_NAMESPACE_TOUCHGFX
    in your project.

* TouchGFX env
  * The message displayed when starting a shell has been fixed with
    correct path to examples.

* Bugfixes
  * Fixed bug in simulator for 1bpp displays when width and/or height
    was not not a multiple of 8.
  * Fixed bug in ScrollableContainer where CANCEL events where not
    always delegated to correct child, causing e.g. buttons to remain
    pressed when dragging outside SC area.
  * Fixed bug when rendering chromArt fonts with a rotated display.
  * Fixed bug - Keyboard widget setTouchable(false) had no effect.
  * Freescale K70 DMA now checks the appropriate DONE bit in TCD0_CSR.
  * On ST processors fixed bug with rotated texts rendered by ChromArt
    when in non-native display orientation.

* Board support
  * Embedded Artists LPC4357DevKit board package: CPU clocked to
    204Mhz (previously 96Mhz). Now uses SPIFI flash instead of NOR.

* Update procedure
  * Compatible with existing 4.X applications. Just replace the
    touchgfx folder.

* Info
  * Documentation has been updated.


4.1.1
=====

* Release date: October 29th, 2014
```

```
* New features
  * Mixin: MoveAnimator added. The MoveAnimator mixin makes the
    template class T able to animate a movement from its current
    position to a specified end position. See
    app/example/move_fade_example.
  * Mixin: FadeAnimator added. The FadeAnimator mixin makes the
    template class T able to animate an alpha fade from its current
    alpha value to a specified end alpha value. See
    app/example/move_fade_example.
  * ScalableImage and ZoomAnimationImage now support alpha per pixel
    bitmaps and alpha per bitmap
  * ScalableImage and ZoomAnimationImage now support ARGB8888 format
    bitmaps

* Bugfixes
  * Fixed a bug causing the Keyboard widget to render incorrectly in
    rare cases.
  * Fixed a bug causing drag event coordinates to be incorrect for
    widgets when placed in a Container with coords != {0,0} which
    itself was placed in a ScrollableContainer.
  * The Application class now properly keeps track of number of times
    registerTimerWidget vs. unregisterTimerWidget is called for a
    given widget, meaning that if registered several times it now
    requires same number of unregisters before widget no longer
    receives tick events.
  * Some ZoomAnimationImage functions were not virtual as they should
    be.
  * Some widgets were missing certain getter functions.

* Update procedure
  * Compatible with existing 4.X applications. Just replace the
    touchgfx folder.


4.1.0
=====

* Release date: October 17th, 2014

* New features
  * Now supports monochrome 1BPP displays. See manual for details.
  * Support for dynamic screen orientation change (landscape/portrait)
  * Support for scaling images (See ScalableImage and
    ZoomAnimationImage drawables)
* Demo
  * Home Control Demo now support 640x480 mode.
  * Home Control Demo now supports STM324xI-EVAL 5.7" board.

* Board support changes
  * Added support for STM324xI-EVAL 5.7" board (IAR+gcc).
  * Added gcc support for the EmbeddedArtists LPC4357DevKit board.
  * Optimized SPIFI initialization for TouchGFX eval board.

* Bugfixes
  * Adding a persistent Drawable to a ScrollableContainer could cause
    assertion
  * Support for much larger fonts

* Update procedure
  * Compatible with existing 4.X applications. Just replace the
    touchgfx folder.


4.0.0
=====

* Release date: September 26th, 2014

* New features
  * TouchEvent refactoring (API breaking):
    * Drawable::setActive is renamed to Drawable::setTouchable
    * Drawable::isActive is renamed to Drawable::isTouchable
```

```
    * Drawable::hijackTouchEvent is deprecated
    * Drawables are now per default not touchables
    * TouchEvents are now always propagated to all containers children
  * Language specific typography and alignment columns support added
    to text converter. Read more about this feature in the
    documentation.
  * Font rendering has been vastly improved with regards to font
    shapes and kerning.
  * Simulator - assert check on new view/presenter/transition size
    when doing screen transition. Failed assert checks probably due
    to missing definition of view/presenter/transition in
    FrontEndHeap.
  * TextArea and ButtonWithLabel now support text rotated 0, 90, 180
    or 270 degrees.
  * Text centering on ButtonWithLabel has been improved in special
    cases.
  * Hardware Accelerated text rendering (4 and 8bpp) on supported ST
    platforms.
  * Ability to cache all items in the bitmap database in external RAM.
  * Support for Freescales K70 MCU.
  * Translation Sheet: Instances of "\<" and "\>" are converted into
    "<" and ">" respectively.  This enables literal translated strings
    such as "<Not a wildcard>" using "\<Not a wildcard\>".
  * Support for NXP LPC18XX series of MCU's.

* Bugfixes
  * Rendering error of images with odd width and alpha value less than
    255
  * Correct handling of TextArea::getTextHeight in case of non
    initialized textArea
  * TextAreaWithWildcard::getTextWidth now includes the width of the
    wildcard text
  * gcc Makefiles now includes *.BMP and *.PNG from image assets.
  * Do not trim leading and trailing white space from any translations
    in the texts sheet.
  * Font converter did not generate font data properly for 8bpp.
  * ButtonWithIcon::setBitmaps - Suppress IAR warning for intentional
    virtual function override.
  * ButtonWithIcon optimized draw functionality
  * In extremely rare cases text could be written slightly outside the
    text area

* Update procedure
  * Due to the TouchEvent refactoring you have to rename functions
    accordingly. You also need to state in any custom widget or
    containers if they need to receive touch events. If you were using
    hijackTouchEvent to prevent children of getting touch events, you
    now need to make sure that all children is not touchable instead.
  * Main.cpp for simulators need to be updated by replacing the line:
        TypedText::registerTypedTextDatabase(TypedTextDatabase::getInstance(), TypedTextDatabase::getInstanceS
    with:
        Texts::setLanguage(0)
    You can also specify a specific language from your text database
    e.g.
        Texts::setLanguage(GB)
    In that case you also need to:
        #include <texts/TextKeysAndLanguages.hpp>
  * Rebuild entire project.

* Info
  * The TouchGFX Manual has been updated considerably.


3.1.0
=====

* New features
  * Added support for FDI uEZGUI-1788-70WVT eval board (NXP LPC-1788
    Cortex M3).
  * Added support for Mjolner TouchGFX Demo Board Rev. 1.1 eval board
    (NXP LPC-4353 Cortex M4/M0 4.3").

* Bugfixes
```

```
  * Visual Studio build now rebuild BitmapDatabase.h when new images
    are added to the assets/images folder.

* Update procedure
  * Only if using Visual Studio: Update TouchGFXReleasePath in your
    Visual Studio .props file. Simply edit the file in a text editor.
    The path should be extended with "touchgfx\". See the
    template_application for inspiration.
  * Only if using Visual Studio: Update your Visual Studio project
    file (.vcxproj file). Simply edit the file in a text editor.
    Replace all paths on the form
    "$(TouchGFXReleasePath)\framework\config\msvs\touchgfx_prebuild.targets"
    with
    "$(TouchGFXReleasePath)\config\msvs\touchgfx_prebuild.targets".

* Info
 * Hardware Abstraction Layer architecture has been reworked so that
   all common code for various hardware components (MCUs and drivers)
   is now shared across different target boards. This greatly
   simplifies the porting effort for new/custom boards as long as they
   contain one or more hardware components already supported by
   TouchGFX.


3.0.0
=====

* New features
  * Visual Studio 2010/2012 support.
  * Added support for png images with alpha channel.
  * Added support for subfolders in assets/bitmaps folder
  * Added support for ST STM32F4X9I-EVAL eval board.
  * Added support for Robert Penners Easing Equations (see
    touchgfx/EasingEquations.hpp).
  * Image converter: Added sanity check of input image file names,
    must not start with digit and must be alphanumeric.
  * Image converter: Added checking against case insensitively file
    name duplicates in input list.
  * Text converter: Added build stopping sanity checks for bpp and
    font_size values.
  * ScrollableContainer: Now supports setScrollbarPadding,
    setScrollbarWidth, setScrollbarColor, and setScrollbarAlpha.
  * ScrollableContainer: Set default value of ScrollThreshold to 5
    pixels, instead of 1.
  * Added support for alpha blending of fonts
    (TextArea::setAlpha(uint8_t alpha))
  * ImageConvert support two different output formats: RGB565 and ARGB8888
  * ImageConvert - two options added to control output format for
    images with/without an alpha channel
  * Touchgfx environment under MinGW is updated due to linker errors
    for large projects. g++ version is updated from 4.6.2 -> 4.8.1
  * Internal RAM footprint improvements
  * Structural changes of target library and hardware abstraction layers

* Bugfixes
  * Fill operation (Box widget) resulted in a crash on the
    lpc4357_emb_artist board
  * Textconvert & fontconvert: Different typographies may now have
    identical properties.
  * Imageconvert & fontconvert: Better error handling for POSIX
    compliant platforms
  * HALSDL: Do not overflow key data type.
  * LanguageXX.cpp files now end with a newline (removing warnings).
  * TextArea::draw now handles non initialized TypedText correctly.

* Update procedure
  * The folders assets/bitmaps and generated/bitmaps must be renamed
    to assets/images and generated/images.
  * Upgrade TouchGFX environment to version 2.0
  * Update any application Makefile to adhere with the Makefile
    specified in the updated template_application
  * Rebuild entire project
  * Convert bmp images that contains the former transparent color to
```

```
    png images that uses alpha channels. This can be done
    automatically using a free tool called imagemagick. More info and
    hints can be acquired by writing touchgfx-support@mjolner.com
  * Custom HAL implementations must be updated to conform with the new
    structure

* Info
  * The "magic" transparent color that was previously used for
    transparent color in the bmp format is no longer supported.
    Instead use png images with alpha channel.


2.2.0
=====

* New features
  * Added support for portrait mode with landscape displays at zero
    performance/resource cost.
  * Added kerning support.
  * Added Keyboard example (with IAR project for the Energy Micro
    DK3750 eval board)
  * Changed interface for blitCopy method in LCD.
  * Removed SyncBackBuffer method from HAL.
  * Removed clearLCD method from LCD.
  * Removed fillGradientRect method from LCD.
  * ScrollableContainer supports setScrollbarsVisible(bool visible).

2.1.0
=====

* First release of TouchGFX as a commercially available framework
```

Chapter 12

Known Issues

The Known Issues list has migrated to our support website. Please refer to

https://touchgfx.zendesk.com/hc/en-us/articles/207507415

for an updated list of known issues.

Chapter 13

Globals

Member touchgfx::touchgfx_generic_init (DMA_Interface &dma, LCD &display, TouchController &tc, int16_t width, int16_t height, uint16_t *bitmapCache, uint32_t bitmapCacheSize, uint32_t numberOfDynamicBitmaps=0)

Chapter 14

Namespace Index

14.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

touchgfx
The global touchgfx namespace. All TouchGFX framework classes and global functions are placed in this namespace 171

Chapter 15

Hierarchical Index

15.1 Class Hierarchy

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

AbstractPainter . . . 189
- AbstractPainterBW . . . 191
  - PainterBW . . . 448
- AbstractPainterRGB565 . . . 192
  - PainterRGB565 . . . 449
  - PainterRGB565Bitmap . . . 451
- AbstractPainterRGB888 . . . 194
  - PainterRGB888 . . . 453
  - PainterRGB888Bitmap . . . 455

AbstractPartition . . . 196
- Partition< ListOfTypes, NUMBER_OF_ELEMENTS > . . . 460

Bitmap . . . 219
Bitmap::BitmapData . . . 227
BlitOp . . . 227
ButtonController . . . 233
Buttons . . . 234
Bitmap::CacheTableEntry . . . 241
Keyboard::CallbackArea . . . 246
Canvas . . . 246
Cell . . . 256
Color . . . 268
colortype . . . 270
CWRUtil . . . 280
DisplayTransformation . . . 285
DMA_Interface . . . 287
- NoDMA . . . 441

DMA_Queue . . . 292
- LockFreeDMA_Queue . . . 428

Drawable . . . 298
- Container . . . 273
  - Keyboard . . . 387
  - ListLayout . . . 425
  - ScrollableContainer . . . 512
  - Slider . . . 523
  - ZoomAnimationImage . . . 593
- Widget . . . 591

AbstractButton . . . 187
Button . . . 231
ButtonWithIcon . . . 235
ButtonWithLabel . . . 238
ToggleButton . . . 565
RadioButton . . . 479
TouchArea . . . 567
Box . . . 228
CanvasWidget . . . 249
AbstractShape . . . 202
Shape< POINTS > . . . 520
Circle . . . 257
Line . . . 419
CanvasWidgetRenderer . . . 253
Image . . . 379
AnimatedImage . . . 209
PixelDataWidget . . . 462
ScalableImage . . . 499
SnapshotWidget . . . 534
TextArea . . . 538
TextAreaWithWildcardBase . . . 548
TextAreaWithOneWildcard . . . 544
TextAreaWithTwoWildcards . . . 546
TextureMapper . . . 551
DrawingSurface . . . 309
Bitmap::DynamicBitmapData . . . 309
EasingEquations . . . 310
Edge . . . 326
Event . . . 328
ClickEvent . . . 263
DragEvent . . . 294
GestureEvent . . . 342
Font . . . 331
ConstFont . . . 271
InternalFlashFont . . . 381
FontManager . . . 337
FontProvider . . . 338
GenericCallback< T1, T2 > . . . 339
Callback< dest_type, T1, T2 > . . . 242
GenericCallback< const T &, const touchgfx::ClickEvent & > . . . 339
GenericCallback< const touchgfx::AbstractButton & > . . . 339
GenericCallback< const touchgfx::AbstractButton &, T2 > . . . 339
Callback< touchgfx::RadioButtonGroup, const touchgfx::AbstractButton & > . . . 242
GenericCallback< const touchgfx::AnimatedImage & > . . . 339
GenericCallback< const touchgfx::DragEvent & > . . . 339
GenericCallback< const touchgfx::FadeAnimator< T > & > . . . 339
GenericCallback< const touchgfx::MoveAnimator< T > & > . . . 339
GenericCallback< const touchgfx::Slider &, int > . . . 339
GenericCallback< const touchgfx::ZoomAnimationImage & > . . . 339
GenericCallback< T1 > . . . 339
Callback< dest_type, T1, void > . . . 243
GenericCallback< T1, void > . . . 340
GenericCallback< touchgfx::Drawable &, T2 > . . . 339
Callback< touchgfx::SlideTransition, touchgfx::Drawable & > . . . 242
GenericCallback< touchgfx::Unicode::UnicodeChar > . . . 339

GenericCallback< void > . . . 341
GenericCallback<> . . . 339
Callback< dest_type, void, void > . . . 245
Gestures . . . 344
GlyphNode . . . 347
GPIO . . . 348
Gradients . . . 350
HAL . . . 351
HALSDL . . . 367
I2C . . . 374
I2C_LPC1788 . . . 375
I2C_LPC43xx . . . 377
Scanline::iterator . . . 383
JSMOCHelper . . . 384
KerningNode . . . 386
Keyboard::Key . . . 387
Keyboard::KeyMapping . . . 392
Keyboard::KeyMappingList . . . 392
Keyboard::Layout . . . 393
LCD . . . 393
LCD16bpp . . . 402
LCD1bpp . . . 408
LCD24bpp . . . 413
LED . . . 418
Matrix4x4 . . . 430
MCUInstrumentation . . . 434
CortexMMCUInstrumentation . . . 278
MVPHeap . . . 440
OSWrappers . . . 444
Outline . . . 445
Pair< T1, T2 > . . . 458
Pair< int16_t, int16_t > . . . 458
Point . . . 464
Point3D . . . 466
Presenter . . . 468
CWRUtil::Q10 . . . 469
CWRUtil::Q15 . . . 471
CWRUtil::Q5 . . . 472
Quadruple . . . 476
Point4 . . . 466
Vector4 . . . 589
RadioButtonGroup< CAPACITY > . . . 483
Rasterizer . . . 487
Rect . . . 490
Renderer . . . 493
RenderingBuffer . . . 495
RGBA8 . . . 498
Scanline . . . 503
Screen . . . 506
View< T > . . . 590
AbstractShape::ShapePoint< T > . . . 523
LCD::StringVisuals . . . 536
TextProvider . . . 549
Texts . . . 550
TextureSurface . . . 565
TouchCalibration . . . 569

TouchController 571
I2CTouchController 378
NoTouchController 442
SDLTouchController 519
Transition 572
NoTransition 443
SlideTransition< templateDirection > 530
TypedText 573
TypedText::TypedTextData 576
UIEventListener 576
Application 213
MVPApplication 438
Unicode 578
Vector< T, capacity > 585
Vector< touchgfx::Drawable *, MAX_TIMER_WIDGETS > 585
Vector< touchgfx::Rect, 8 > 585
T
ClickListener< T > 267
Draggable< T > 297
Snapper< T > 532
FadeAnimator< T > 329
MoveAnimator< T > 436
PreRenderable< T > 467

Chapter 16

Class Index

16.1 Class List

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

AbstractButton
This class defines an abstract interface for button-like elements 187
AbstractPainter
An abstract class for creating painter classes for drawing canvas widgets 189
AbstractPainterBW
A Painter that will paint using a color on a LCD1bpp display 191
AbstractPainterRGB565
A Painter that will paint using a color and an alpha value 192
AbstractPainterRGB888
A Painter that will paint using a color and an alpha value 194
AbstractPartition
This type defines an abstract interface to a storage partition for allocating memory slots of equal size 196
AbstractShape
Simple widget capable of drawing a abstractShape 202
AnimatedImage
A widget capable of basic animation using a range of bitmaps 209
Application
Main interface for manipulating screen contents 213
Bitmap
This class provides a proxy object for a bitmap image 219
Bitmap::BitmapData
Data of a bitmap 227
BlitOp
BlitOp instances carry the required information for performing operations on the LCD (frame buffer) using DMA 227
Box
Simple widget capable of showing a rectangle of a specific color and an optional alpha 228
Button
A button with two states 231
ButtonController
Interface for sampling external key events 233
Buttons
A buttons 234
ButtonWithIcon
A Button specialization that also displays an icon on top of the button bitmap 235
ButtonWithLabel
A Button specialization that also displays a text on top of the button bitmap 238

Bitmap::CacheTableEntry
Cache bookkeeping . . . 241
Callback< dest_type, T1, T2 >
A Callback is basically a wrapper of a pointer-to-member-function . . . 242
Callback< dest_type, T1, void >
A Callback is basically a wrapper of a pointer-to-member-function . . . 243
Callback< dest_type, void, void >
A Callback is basically a wrapper of a pointer-to-member-function . . . 245
Keyboard::CallbackArea
Mapping from rectangle to a callback method to execute . . . 246
Canvas
Class for easy rendering using CanvasWidgetRenderer . . . 246
CanvasWidget
Class for drawing complex polygons on the LCD using CanvasWidgetRenderer . . . 249
CanvasWidgetRenderer
Class for supporting drawing of figures . . . 253
Cell
A pixel cell . . . 256
Circle
Simple widget capable of drawing a circle . . . 257
ClickEvent
A click event . . . 263
ClickListener< T >
Mix-in class that extends a class with a click action event . . . 267
Color
Contains functionality for color conversion . . . 268
colortype
Type for representing a color . . . 270
ConstFont
A ConstFont is a Font implementation that has its contents defined at compile-time and usually placed in read-only memory . . . 271
Container
A Container is a Drawable that can have child nodes . . . 273
CortexMMCUInstrumentation
Interface for instrumenting Cortex-M processors to measure MCU load via measured CPU cycles 278
CWRUtil
Helper classes and functions for CanvasWidget . . . 280
DisplayTransformation
Defines transformations from display space to frame buffer space . . . 285
DMA_Interface
DMA_Interface provides basic functionality and structure for processing "blit" operations using DMA . . . 287
DMA_Queue
This class provides an interface for a FIFO (circular) list used by DMA_Interface and descendants for storing BlitOp's . . . 292
DragEvent
A drag event . . . 294
Draggable< T >
Mix-in class that extends a class to become draggable . . . 297
Drawable
Abstract definition of something that can be drawn . . . 298
DrawingSurface
The destination of a draw operation. Contains a pointer to where to draw and the stride of the drawing surface . . . 309
Bitmap::DynamicBitmapData
Data of a dynamic bitmap . . . 309

EasingEquations
Defines the "Penner easing functions", which are a de facto standard computing aesthetically pleasing motion animations . . . 310
Edge
An edge contains information about one edge, between two points, of a triangle, as well as information about how to interpolate values when moving in the vertical direction . . . 326
Event
Simple base class for events . . . 328
FadeAnimator< T >
A FadeAnimator makes the template class T able to animate an alpha fade . . . 329
Font
The font base class . . . 331
FontManager
This class is the entry point for looking up a font based on a font id . . . 337
FontProvider
A generic pure virtual definition of a FontProvider . . . 338
GenericCallback< T1, T2 >
GenericCallback is the base class for callbacks . . . 339
GenericCallback< T1, void >
GenericCallback is the base class for callbacks . . . 340
GenericCallback< void >
GenericCallback is the base class for callbacks . . . 341
GestureEvent
A gesture event . . . 342
Gestures
This class implements the detection of gestures . . . 344
GlyphNode
Struct providing information about a glyph . . . 347
GPIO
Interface class for manipulating GPIOs in order to do performance measurements on target . . . 348
Gradients
Gradients contains all the data to interpolate u,v texture coordinates and z coordinates across a planar surface . . . 350
HAL
Hardware Abstraction Layer . . . 351
HALSDL
HAL implementation for the TouchGFX simulator . . . 367
I2C
Platform independent interface for I2C drivers . . . 374
I2C_LPC1788 . . . 375
I2C_LPC43xx . . . 377
I2CTouchController
Specific I2C-enabled type of Touch Controller . . . 378
Image
Simple widget capable of showing a bitmap . . . 379
InternalFlashFont
An InternalFlashFont has both glyph table and glyph data placed in a flash which supports random access read (i.e. not a NAND flash) . . . 381
Scanline::iterator
An iterator to help go through all the elements that make up a Scanline . . . 383
JSMOCHelper
Helper class providing caching of certain information while the JSMOC algorithm runs during draw operations . . . 384
KerningNode
Structure providing information about a kerning for a given char pair . . . 386
Keyboard::Key
Mapping from rectangle to key id . . . 387

Keyboard
The keyboard provides text input for touch devices . . . 387
Keyboard::KeyMapping
Mapping from key id to Unicode character . . . 392
Keyboard::KeyMappingList
List of KeyMappings to use . . . 392
Keyboard::Layout
Definition of the keyboard layout. The keyboard can handle changing layouts, so different keyboard modes can be implemented by changing layouts and key mappings . . . 393
LCD
This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles . . . 393
LCD16bpp
This class contains the various low-level drawing routines for drawing bitmaps . . . 402
LCD1bpp
This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles . . . 408
LCD24bpp
This class contains the various low-level drawing routines for drawing bitmaps . . . 413
LED
A led . . . 418
Line
Simple CanvasWidget capable of drawing a line . . . 419
ListLayout
This class provides a layout mechanism for arranging Drawable instances adjacently in the specified Direction . . . 425
LockFreeDMA_Queue
This implements a simple lock-free FIFO queue (single producer, single consumer) . . . 428
Matrix4x4
This class represents row major 4x4 homogeneous matrices . . . 430
MCUInstrumentation
Interface for instrumenting processors to measure MCU load via measured CPU cycles . . . 434
MoveAnimator< T >
A MoveAnimator makes the template class T able to animate a movement . . . 436
MVPApplication
A specialization of the TouchGFX Application class . . . 438
MVPHeap
Generic heap class for MVP applications . . . 440
NoDMA
This is an "empty" DMA subclass that does nothing except assert if accidentally used . . . 441
NoTouchController . . . 442
NoTransition
The most simple Transition without any visual effects . . . 443
OSWrappers
This class specifies OS wrappers for dealing with the frame buffer semaphore and the VSYNC signal . . . 444
Outline
An internal class that implements the main rasterization algorithm . . . 445
PainterBW
A Painter that will paint using a color on a LCD1bpp display . . . 448
PainterRGB565
A Painter that will paint using a color and an alpha value . . . 449
PainterRGB565Bitmap
A Painter that will paint using a bitmap . . . 451
PainterRGB888
A Painter that will paint using a color and an alpha value . . . 453
PainterRGB888Bitmap
A Painter that will paint using a bitmap . . . 455

Pair< T1, T2 >
A simple struct for holding pairs of data . . . 458
Partition< ListOfTypes, NUMBER_OF_ELEMENTS >
This type provides a concrete Partition of memory-slots capable of holding any of the specified list of types . . . 460
PixelDataWidget
A widget for displaying a buffer of pixel data . . . 462
Point
A simple struct containing coordinates . . . 464
Point3D
A 3D point . . . 466
Point4
This class represents a homogeneous 3D point . . . 466
PreRenderable< T >
This mixin can be used on any Drawable . . . 467
Presenter
The Presenter base class that all application-specific presenters should derive from . . . 468
CWRUtil::Q10
Defines a number with 10 bits reserved for fraction . . . 469
CWRUtil::Q15
Defines a number with 15 bits reserved for fraction . . . 471
CWRUtil::Q5
Defines a number with 5 bits reserved for fraction . . . 472
Quadruple
Base class for homogeneous vectors and points . . . 476
RadioButton
Radio button with two states . . . 479
RadioButtonGroup< CAPACITY >
Class for handling a collection of RadioButtons . . . 483
Rasterizer
Polygon Rasterizer that is used to render filled polygons with high-quality Anti- Aliasing . . . 487
Rect
Class representing a Rectangle with a few convenient methods . . . 490
Renderer
This class template is used basically for rendering scan lines . . . 493
RenderingBuffer
Rendering buffer wrapper . . . 495
RGBA8
Deprecated, use colortype instead . . . 498
ScalableImage
Widget for representing a scaled version of a bitmap . . . 499
Scanline
This class is used to transfer data from class Outline (or a similar one) to the rendering buffer . 503
Screen
A Screen represents a full-screen drawable area. Applications create specific screens by subclassing this class . . . 506
ScrollableContainer
A ScrollableContainer is a container that allows its contents to be scrolled . . . 512
SDLTouchController
TouchController for the simulator . . . 519
Shape< POINTS >
Simple widget capable of drawing a shape . . . 520
AbstractShape::ShapePoint< T >
Defines an alias representing the array of points making up the abstract shape . . . 523
Slider
A slider is a graphical element with which the user may set a value by moving an indicator or by clicking the slider . . . 523

SlideTransition< templateDirection >
A Transition that slides from one screen to the next 530
Snapper< T >
A mix-in that will make class T draggable and able to snap to a position 532
SnapshotWidget
A widget that is able to make a snapshot of the area the SnapshotWidget covers 534
LCD::StringVisuals
The visual elements when writing a string 536
TextArea
This widget is capable of showing a text area on the screen 538
TextAreaWithOneWildcard
TextArea with one wildcard 544
TextAreaWithTwoWildcards
TextArea with two wildcards 546
TextAreaWithWildcardBase
Base class for TextAreas displaying texts with one or more wildcards 548
TextProvider
The TextProvider is used in drawing basic and wildcard strings 549
Texts
Class for setting language and getting texts 550
TextureMapper
Widget capable of drawing a transformed image 551
TextureSurface
A texture source. Contains a pointer to the data and the width and height of the texture. The alpha channel is used in 565 rendering with alpha. The stride is the width used when moving to the next line of the texture 565
ToggleButton
A ToggleButton is a Button specialization that swaps the two bitmaps when clicked 565
TouchArea
Invisible widget used to capture touch events 567
TouchCalibration
Calibrates a touch coordinate 569
TouchController
Basic Touch Controller interface 571
Transition
Base class for Transitions 572
TypedText
TypedText represents text (as in characters) and typography (as in font and alignment) 573
TypedText::TypedTextData
The data structure for typed texts 576
UIEventListener
This class declares a handler interface for user interface events 576
Unicode
This class provides simple helper functions for working with 16-bit strings 578
Vector< T, capacity >
A very simple container class using pre-allocated memory 585
Vector4
This class represents a homogeneous 3D vector 589
View< T >
This is a generic touchgfx::Screen specialization for normal applications 590
Widget
A Widget is a Drawable leaf (i.e. not a container) 591
ZoomAnimationImage
Class for optimizing and wrapping move and zoom operations on ScalableImages 593

Chapter 17

Namespace Documentation

17.1 touchgfx Namespace Reference

The global touchgfx namespace. All TouchGFX framework classes and global functions are placed in this namespace.

Classes

- class AbstractButton

  This class defines an abstract interface for button-like elements.
- class AbstractPainter

  An abstract class for creating painter classes for drawing canvas widgets.
- class AbstractPainterBW

  A Painter that will paint using a color on a LCD1bpp display.
- class AbstractPainterRGB565

  A Painter that will paint using a color and an alpha value.
- class AbstractPainterRGB888

  A Painter that will paint using a color and an alpha value.
- class AbstractPartition

  This type defines an abstract interface to a storage partition for allocating memory slots of equal size.
- class AbstractShape

  Simple widget capable of drawing a abstractShape.
- class AnimatedImage

  A widget capable of basic animation using a range of bitmaps.
- class Application

  The Application class is the main interface for manipulating screen contents.
- class Bitmap

  This class provides a proxy object for a bitmap image.
- struct BlitOp

  BlitOp instances carry the required information for performing operations on the LCD (frame buffer) using DMA.
- class Box

  Simple widget capable of showing a rectangle of a specific color and an optional alpha.
- class Button

  A button with two states.
- class ButtonController

  Interface for sampling external key events.
- class Buttons

A buttons.

- class ButtonWithIcon

  A Button specialization that also displays an icon on top of the button bitmap.
- class ButtonWithLabel

  A Button specialization that also displays a text on top of the button bitmap.
- struct Callback

  A Callback is basically a wrapper of a pointer-to-member-function.
- struct Callback< dest_type, T1, void >

  A Callback is basically a wrapper of a pointer-to-member-function.
- struct Callback< dest_type, void, void >

  A Callback is basically a wrapper of a pointer-to-member-function.
- class Canvas

  Class for easy rendering using CanvasWidgetRenderer.
- class CanvasWidget

  Class for drawing complex polygons on the LCD using CanvasWidgetRenderer.
- class CanvasWidgetRenderer

  Class for supporting drawing of figures.
- struct Cell

  A pixel cell.
- class Circle

  Simple widget capable of drawing a circle.
- class ClickEvent

  A click event.
- class ClickListener

  Mix-in class that extends a class with a click action event.
- class Color

  Contains functionality for color conversion.
- struct colortype

  Type for representing a color.
- class ConstFont

  A ConstFont is a Font implementation that has its contents defined at compile-time and usually placed in read-only memory.
- class Container

  A Container is a Drawable that can have child nodes.
- class CortexMMCUInstrumentation

  Interface for instrumenting Cortex-M processors to measure MCU load via measured CPU cycles.
- struct CWRUtil

  Helper classes and functions for CanvasWidget.
- class DisplayTransformation

  Defines transformations from display space to frame buffer space.
- class DMA_Interface

  DMA_Interface provides basic functionality and structure for processing "blit" operations using DMA.
- class DMA_Queue

  This class provides an interface for a FIFO (circular) list used by DMA_Interface and descendants for storing BlitOp's.
- class DragEvent

  A drag event.
- class Draggable

  Mix-in class that extends a class to become draggable.
- class Drawable

  The Drawable class is an abstract definition of something that can be drawn.
- struct DrawingSurface

The destination of a draw operation. Contains a pointer to where to draw and the stride of the drawing surface.

- class EasingEquations

  Defines the "Penner easing functions", which are a de facto standard computing aesthetically pleasing motion animations.
- struct Edge

  An edge contains information about one edge, between two points, of a triangle, as well as information about how to interpolate values when moving in the vertical direction.
- class Event

  Simple base class for events.
- class FadeAnimator

  A FadeAnimator makes the template class T able to animate an alpha fade.
- class Font

  The font base class.
- class FontManager

  This class is the entry point for looking up a font based on a font id.
- class FontProvider

  A generic pure virtual definition of a FontProvider.
- class GenericCallback

  GenericCallback is the base class for callbacks.
- class GenericCallback< T1, void >

  GenericCallback is the base class for callbacks.
- class GenericCallback< void >

  GenericCallback is the base class for callbacks.
- class GestureEvent

  A gesture event.
- class Gestures

  This class implements the detection of gestures.
- struct GlyphNode

  struct providing information about a glyph.
- class GPIO

  Interface class for manipulating GPIOs in order to do performance measurements on target.
- struct Gradients

  Gradients contains all the data to interpolate u,v texture coordinates and z coordinates across a planar surface.
- class HAL

  Hardware Abstraction Layer.
- class HALSDL

  HAL implementation for the TouchGFX simulator.
- class I2C

  Platform independent interface for I2C drivers.
- class I2C_LPC1788
- class I2C_LPC43xx
- class I2CTouchController

  Specific I2C-enabled type of Touch Controller.
- class Image

  Simple widget capable of showing a bitmap.
- class InternalFlashFont

  An InternalFlashFont has both glyph table and glyph data placed in a flash which supports random access read (i.e. not a NAND flash).
- class JSMOCHelper

  Helper class providing caching of certain information while the JSMOC algorithm runs during draw operations.
- struct KerningNode

Structure providing information about a kerning for a given char pair.

- class Keyboard

  The keyboard provides text input for touch devices.
- class LCD

  This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles.
- class LCD16bpp

  This class contains the various low-level drawing routines for drawing bitmaps.
- class LCD1bpp

  This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles.
- class LCD24bpp

  This class contains the various low-level drawing routines for drawing bitmaps.
- class LED

  A led.
- class Line

  Simple CanvasWidget capable of drawing a line.
- class ListLayout

  This class provides a layout mechanism for arranging Drawable instances adjacently in the specified Direction.
- class LockFreeDMA_Queue

  This implements a simple lock-free FIFO queue (single producer, single consumer).
- class Matrix4x4

  This class represents row major 4x4 homogeneous matrices.
- class MCUInstrumentation

  Interface for instrumenting processors to measure MCU load via measured CPU cycles.
- class MoveAnimator

  A MoveAnimator makes the template class T able to animate a movement.
- class MVPApplication

  A specialization of the TouchGFX Application class.
- class MVPHeap

  Generic heap class for MVP applications.
- class NoDMA

  This is an "empty" DMA subclass that does nothing except assert if accidentally used.
- class NoTouchController
- class NoTransition

  The most simple Transition without any visual effects.
- class OSWrappers

  This class specifies OS wrappers for dealing with the frame buffer semaphore and the VSYNC signal.
- class Outline

  An internal class that implements the main rasterization algorithm.
- class PainterBW

  A Painter that will paint using a color on a LCD1bpp display.
- class PainterRGB565

  A Painter that will paint using a color and an alpha value.
- class PainterRGB565Bitmap

  A Painter that will paint using a bitmap.
- class PainterRGB888

  A Painter that will paint using a color and an alpha value.
- class PainterRGB888Bitmap

  A Painter that will paint using a bitmap.
- struct Pair

  A simple struct for holding pairs of data.
- class Partition

This type provides a concrete Partition of memory-slots capable of holding any of the specified list of types.

- class PixelDataWidget

  A widget for displaying a buffer of pixel data.
- struct Point

  A simple struct containing coordinates.
- struct Point3D

  A 3D point.
- class Point4

  This class represents a homogeneous 3D point.
- class PreRenderable

  This mixin can be used on any Drawable.
- class Presenter

  The Presenter base class that all application-specific presenters should derive from.
- class Quadruple

  Base class for homogeneous vectors and points.
- class RadioButton

  Radio button with two states.
- class RadioButtonGroup

  Class for handling a collection of RadioButtons.
- class Rasterizer

  Polygon Rasterizer that is used to render filled polygons with high-quality Anti- Aliasing.
- class Rect

  Class representing a Rectangle with a few convenient methods.
- class Renderer

  This class template is used basically for rendering scan lines.
- class RenderingBuffer

  Rendering buffer wrapper.
- struct RGBA8

  Deprecated, use colortype instead.
- class ScalableImage

  Widget for representing a scaled version of a bitmap.
- class Scanline

  This class is used to transfer data from class Outline (or a similar one) to the rendering buffer.
- class Screen

  A Screen represents a full-screen drawable area. Applications create specific screens by subclassing this class.
- class ScrollableContainer

  A ScrollableContainer is a container that allows its contents to be scrolled.
- class SDLTouchController

  TouchController for the simulator.
- class Shape

  Simple widget capable of drawing a shape.
- class Slider

  A slider is a graphical element with which the user may set a value by moving an indicator or by clicking the slider.
- class SlideTransition

  A Transition that slides from one screen to the next.
- class Snapper

  A mix-in that will make class T draggable and able to snap to a position.
- class SnapshotWidget

  A widget that is able to make a snapshot of the area the SnapshotWidget covers.
- class TextArea

  This widget is capable of showing a text area on the screen.

- class TextAreaWithOneWildcard

  TextArea with one wildcard.
- class TextAreaWithTwoWildcards

  TextArea with two wildcards.
- class TextAreaWithWildcardBase

  Base class for TextAreas displaying texts with one or more wildcards.
- class TextProvider

  The TextProvider is used in drawing basic and wildcard strings.
- class Texts

  Class for setting language and getting texts.
- class TextureMapper

  The TextureMapper class is a widget capable of drawing a transformed image.
- struct TextureSurface

  A texture source. Contains a pointer to the data and the width and height of the texture. The alpha channel is used in 565 rendering with alpha. The stride is the width used when moving to the next line of the texture.
- class ToggleButton

  A ToggleButton is a Button specialization that swaps the two bitmaps when clicked.
- class TouchArea

  Invisible widget used to capture touch events.
- class TouchCalibration

  Calibrates a touch coordinate.
- class TouchController

  Basic Touch Controller interface.
- class Transition

  The Transition class is the base class for Transitions.
- class TypedText

  TypedText represents text (as in characters) and typography (as in font and alignment).
- class UIEventListener

  This class declares a handler interface for user interface events.
- class Unicode

  This class provides simple helper functions for working with 16-bit strings.
- class Vector

  A very simple container class using pre-allocated memory.
- class Vector4

  This class represents a homogeneous 3D vector.
- class View

  This is a generic touchgfx::Screen specialization for normal applications.
- class Widget

  A Widget is a Drawable leaf (i.e. not a container).
- class ZoomAnimationImage

  Class for optimizing and wrapping move and zoom operations on ScalableImages.

Typedefs

- typedef uint16_t BitmapId

  This type shall be used by the application to define unique IDs for all bitmaps in the system. The application shall define bitmap IDs in the range [0, number of bitmaps - 1].
- typedef int16_t(∗ EasingEquation)(uint16_t, int16_t, int16_t, uint16_t)

  This function pointer typedef matches the signature for all easing equations.
- typedef uint16_t FontId

  Defines an alias representing identifier for the font.

- typedef uint8_t Alignment

  Defines an alignment type.
- typedef uint8_t TextDirection

  Defines a the direction to write text.
- typedef int32_t fixed28_4

  A fixed point value using 4 bits for the decimal part and 28 bits for the integral part.
- typedef int32_t fixed16_16

  A fixed point value using 16 bits for the decimal part and 16 bits for the integral part.
- typedef uint16_t TypedTextId

  Text IDs as generated by the text converter are simple uint16_t typedefs.
- typedef uint16_t LanguageId

  Language IDs generated by the text converter are uint16_t typedef'ed.

Enumerations

- enum GlyphFlags { GLYPH_DATA_FORMAT_A4 = 0x01, GLYPH_DATA_WIDTH_BIT8 = 0x08, GLYPH_DATA_HEIGHT_BIT8 = 0x10, GLYPH_DATA_TOP_BIT8 = 0x20, GLYPH_DATA_TOP_BIT9 = 0x40, GLYPH_DATA_ADVANCE_BIT8 = 0x80 }

  Glyph flag definitions.
- enum BlitOperations { BLIT_OP_COPY = 1 << 0, BLIT_OP_FILL = 1 << 1, BLIT_OP_COPY_WITH_ALPHA = 1 << 2, BLIT_OP_FILL_WITH_ALPHA = 1 << 3, BLIT_OP_COPY_WITH_TRANSPARENT_PIXELS = 1 << 4, BLIT_OP_COPY_ARGB8888 = 1 << 5, BLIT_OP_COPY_ARGB8888_WITH_ALPHA = 1 << 6, BLIT_OP_COPY_A4 = 1 << 7, BLIT_OP_COPY_A8 = 1 << 8 }

  The BlitOp operations.
- enum Direction { NORTH, SOUTH, EAST, WEST }

  Defines a 2D direction type.
- enum FrameBuffer { FB_PRIMARY, FB_SECONDARY, FB_TERTIARY }

  Defines a FrameBuffer type.
- enum Gradient { GRADIENT_HORIZONTAL, GRADIENT_VERTICAL }

  Defines a gradient type.
- enum DisplayRotation { rotate0, rotate90 }

  Defines a rotation of the display.
- enum DisplayOrientation { ORIENTATION_LANDSCAPE, ORIENTATION_PORTRAIT }

  Defines the orientation of the display.
- enum TextRotation { TEXT_ROTATE_0, TEXT_ROTATE_90, TEXT_ROTATE_180, TEXT_ROTATE_270 }

  Defines a rotation of text.
- enum RenderingVariant { **NEARESTNEIGHBOR_565**, **NEARESTNEIGHBOR_565_ALPHA**, **NEARESTNEIGHBOR_888**, **NEARESTNEIGHBOR_8888**, **NEARESTNEIGHBOR_8888_ALPHA**, **BILINEAR_565**, **BILINEAR_565_ALPHA**, **BILINEAR_888**, **BILINEAR_8888**, **BILINEAR_8888_ALPHA** }

  Describes a combination of rendering algorithm, image format, and alpha information.

Functions

- template<class HALType >
  HAL & touchgfx_generic_init (DMA_Interface &dma, LCD &display, TouchController &tc, int16_t width, int16_t height, uint16_t *bitmapCache, uint32_t bitmapCacheSize, uint32_t numberOfDynamicBitmaps=0)

  TouchGFX generic initialize.
- static void prepareTransition (Screen **currentScreen, Presenter **currentPresenter, Transition **currentTrans)

  Prepare screen transition. Private helper function for makeTransition. Do not use.

- static void finalizeTransition (Screen ∗newScreen, Presenter ∗newPresenter, Transition ∗newTransition)

  Finalize screen transition. Private helper function for makeTransition. Do not use.
- template<class ScreenType , class PresenterType , class TransType , class ModelType >
  PresenterType ∗ makeTransition (Screen ∗∗currentScreen, Presenter ∗∗currentPresenter, MVPHeap &heap, Transition ∗∗currentTrans, ModelType ∗model)

  Function for effectuating a screen transition (i.e. makes the requested new presenter/view pair active). Once this function has returned, the new screen has been transitioned to. Due to the memory allocation strategy of using the same memory area for all screens, the old view/presenter will no longer exist when this function returns.
- void hw_init ()

  Function to perform generic hardware initialization of the board.
- void touchgfx_init ()

  Function to perform touchgfx initialization.
- Matrix4x4 operator∗ (const Matrix4x4 &multiplicand, const Matrix4x4 &multiplier)

  Multiplication operator.
- Point4 operator∗ (const Matrix4x4 &multiplicand, const Point4 &multiplier)

  Multiplication operator.
- float fixed28_4ToFloat (fixed28_4 value)

  Fixed 28 4 to float.
- fixed28_4 floatToFixed28_4 (float value)

  Float to fixed 28 4.
- fixed16_16 floatToFixed16_16 (float value)

  Float to fixed 16.
- fixed28_4 fixed28_4Mul (fixed28_4 a, fixed28_4 b)

  Fixed 28 4 mul.
- int32_t ceil28_4 (fixed28_4 value)

  Ceiling 28 4.
- void floorDivMod (int32_t numerator, int32_t denominator, int32_t &floor, int32_t &mod)

  Floor div modifier.
- void memset (void ∗data, uint8_t c, uint32_t size)

  Simple implementation of the standard memset function.
- RenderingVariant lookupNearestNeighborRenderVariant (const Bitmap &bitmap)

  Returns the associated nearest neighbor render variant based on the bitmap format.
- RenderingVariant lookupBilinearRenderVariant (const Bitmap &bitmap)

  Returns the associated bilinear render variant based on the bitmap format.
- template<class T >
  T abs (T d)

  Simple implementation of the standard abs function.

Variables

- const BitmapId BITMAP_INVALID = 0xFFFFU

  Define the bitmapId if an invalid bitmap.
- static const Alignment LEFT = 0

  Text is left aligned.
- static const Alignment CENTER = 1

  Text is centered horizontally.
- static const Alignment RIGHT = 2

  Text is right aligned.
- static const TextDirection TEXT_DIRECTION_LTR = 0

  Text is written Left-To-Right, e.g. English.
- static const TextDirection TEXT_DIRECTION_RTL = 1

Text is written Right-To-Left, e.g. Hebrew.

- static const float PI = 3.14159265358979323846f

  PI.

- const TypedTextId TYPED_TEXT_INVALID = 0xFFFFU

  The ID of an invalid text.

17.1.1 Typedef Documentation

17.1.1.1 uint8_t Alignment

Defines an alignment type.

17.1.1.2 int16_t(∗ EasingEquation)(uint16_t, int16_t, int16_t, uint16_t)

This function pointer typedef matches the signature for all easing equations. Thereby EasingEquation is a convenient shorthand for a pointer to any easing equation.

17.1.1.3 uint8_t TextDirection

Defines a the direction to write text.

17.1.2 Enumeration Type Documentation

17.1.2.1 enum BlitOperations

The BlitOp operations.

Enumerator

BLIT_OP_COPY Copy the source to the destination.

BLIT_OP_FILL Fill the destination with color.

BLIT_OP_COPY_WITH_ALPHA Copy the source to the destination using the given alpha.

BLIT_OP_FILL_WITH_ALPHA Fill the destination with color using the given alpha.

BLIT_OP_COPY_WITH_TRANSPARENT_PIXELS Deprecated, ignored. (Copy the source to the destination, but not the transparent pixels)

BLIT_OP_COPY_ARGB8888 Copy the source to the destination, performing per-pixel alpha blending.

BLIT_OP_COPY_ARGB8888_WITH_ALPHA Copy the source to the destination, performing per-pixel alpha blending and blending the result with an image-wide alpha.

BLIT_OP_COPY_A4 Copy 4-bit source text to destination, performing per-pixel alpha blending.

BLIT_OP_COPY_A8 Copy 8-bit source text to destination, performing per-pixel alpha blending.

17.1.2.2 enum enum Direction

Defines a 2D direction type.

Enumerator

NORTH An enum constant representing the north option.

SOUTH An enum constant representing the south option.

EAST An enum constant representing the east option.

WEST An enum constant representing the west option.

17.1.2.3 enum enum DisplayOrientation

Defines the orientation of the display.

Enumerator

ORIENTATION_LANDSCAPE The display has more pixels from left to right than from top to bottom.

ORIENTATION_PORTRAIT The display has more pixels from top to bottom than from right to left.

17.1.2.4 enum enum DisplayRotation

Defines a rotation of the display.

See also

DisplayOrientation

Enumerator

rotate0 The display is oriented like the frame buffer.

rotate90 The display is rotated 90 degrees compared to the frame buffer layout.

17.1.2.5 enum enum FrameBuffer

Defines a FrameBuffer type.

Enumerator

FB_PRIMARY First frame buffer.

FB_SECONDARY Second frame buffer.

FB_TERTIARY Third frame buffer.

17.1.2.6 enum GlyphFlags

Glyph flag definitions.

Enumerator

GLYPH_DATA_FORMAT_A4 Indicates whether or not glyph data was generated in A4 data format for an ST platform.

GLYPH_DATA_WIDTH_BIT8 The 9th bit of "width".

GLYPH_DATA_HEIGHT_BIT8 The 9th bit of "height".

GLYPH_DATA_TOP_BIT8 The 9th bit of "top".

GLYPH_DATA_TOP_BIT9 The sign bit of "top".

GLYPH_DATA_ADVANCE_BIT8 The 9th bit of "advance".

17.1.2.7 enum enum Gradient

Defines a gradient type.

Enumerator

GRADIENT_HORIZONTAL Horizontal gradient.

GRADIENT_VERTICAL Vertical gradient.

17.1.2.8 enum RenderingVariant

Describes a combination of rendering algorithm, image format, and alpha information.

17.1.2.9 enum enum TextRotation

Defines a rotation of text. Each enumeration option specifies the number of degrees the text is turned clockwise.

Enumerator

TEXT_ROTATE_0 Text is written from left to right.

TEXT_ROTATE_90 Text is written from top to bottom.

TEXT_ROTATE_180 Text is written from right to left (upside down)

TEXT_ROTATE_270 Text is written bottom to top.

17.1.3 Function Documentation

17.1.3.1 template< class T > T abs (T *d*)

Simple implementation of the standard abs function.

Template Parameters

T	The class type on which to perform the abs.

Parameters

d	The entity on which to perform the abs.

Returns

The absolute (non-negative) value of d.

17.1.3.2 int32_t ceil28_4 (fixed28_4 *value*) [inline]

Parameters

value	The value.

Returns

The ceil result.

17.1.3.3 static inline void finalizeTransition (Screen * *newScreen,* Presenter * *newPresenter,* Transition * *newTransition*) [inline],[static]

Parameters

in	*newScreen*	If non-null, the new screen.
in	*newPresenter*	If non-null, the new presenter.

in	*newTransition*	If non-null, the new transition.

17.1.3.4 fixed28_4 fixed28_4Mul (fixed28_4 *a*, fixed28_4 *b*) [inline]

Parameters

a	The fixed28_4 to process.
b	The fixed28_4 to process.

Returns

the result.

17.1.3.5 float fixed28_4ToFloat (fixed28_4 *value*) [inline]

Parameters

value	The value.

Returns

The value as float.

17.1.3.6 fixed16_16 floatToFixed16_16 (float *value*) [inline]

Parameters

value	The value.

Returns

The value as fixed16_16.

17.1.3.7 fixed28_4 floatToFixed28_4 (float *value*) [inline]

Parameters

value	The value.

Returns

The value as fixed28_4.

17.1.3.8 void floorDivMod (int32_t *numerator*, int32_t *denominator*, int32_t & *floor*, int32_t & *mod*) [inline]

Parameters

	numerator	The numerator.
	denominator	The denominator.
in,out	*floor*	The floor.
in,out	*mod*	The modifier.

17.1.3.9 void hw_init ( )

Function to perform generic hardware initialization of the board. This function prototype is only provided as a convention.

17.1.3.10 RenderingVariant lookupBilinearRenderVariant (const Bitmap & *bitmap*)

Returns the associated bilinear render variant based on the bitmap format.

Parameters

bitmap	The bitmap.

Returns

A RenderingVariant based on the bitmap format.

17.1.3.11 RenderingVariant lookupNearestNeighborRenderVariant (const Bitmap & *bitmap*)

Returns the associated nearest neighbor render variant based on the bitmap format.

Parameters

bitmap	The bitmap.

Returns

A RenderingVariant based on the bitmap format.

17.1.3.12 template< class ScreenType, class PresenterType, class TransType, class ModelType > PresenterType * makeTransition (Screen ** *currentScreen*, Presenter ** *currentPresenter*, MVPHeap & *heap*, Transition ** *currentTrans*, ModelType * *model*)

Will properly clean up old screen (tearDownScreen, Presenter::deactivate) and call setupScreen/activate on new view/presenter pair. Will also make sure the view, presenter and model are correctly bound to each other.

Template Parameters

ScreenType	Class type for the View.
PresenterType	Class type for the Presenter.
TransType	Class type for the Transition.
ModelType	Class type for the Model.

Parameters

in	*currentScreen*	Pointer to pointer to the current view.

in	*currentPresenter*	Pointer to pointer to the current presenter.
in	*heap*	Reference to the heap containing the memory storage in which to allocate.
in	*currentTrans*	Pointer to pointer to the current transition.
in	*model*	Pointer to model.

Returns

Pointer to the new Presenter of the requested type. Incidentally it will be the same value as the old presenter due to memory reuse.

17.1.3.13 void memset (void * *data*, uint8_t *c*, uint32_t *size*)

Provides utility functions.

```
Simple implementation of the standard memset function.
```

Parameters

out	*data*	Address of data to set.
	c	Value to set.
	size	Number of bytes to set.

17.1.3.14 Matrix4x4 operator* (const Matrix4x4 & *multiplicand*, const Matrix4x4 & *multiplier*)

Multiplication operator.

Parameters

multiplicand	The first value to multiply.
multiplier	The second value to multiply.

Returns

The result of the operation.

17.1.3.15 Point4 operator* (const Matrix4x4 & *multiplicand*, const Point4 & *multiplier*)

Multiplication operator.

Parameters

multiplicand	The first value to multiply.
multiplier	The second value to multiply.

Returns

The result of the operation.

17.1.3.16 static inline void prepareTransition (Screen ** *currentScreen*, Presenter ** *currentPresenter*, Transition ** *currentTrans*) [inline],[static]

Parameters

in	*currentScreen*	If non-null, the current screen.
in	*currentPresenter*	If non-null, the current presenter.
in	*currentTrans*	If non-null, the current transaction.

17.1.3.17 template< class HALType > HAL & touchgfx_generic_init (DMA_Interface & *dma*, LCD & *display*, TouchController & *tc*, int16_t *width*, int16_t *height*, uint16_t * *bitmapCache*, uint32_t *bitmapCacheSize*, uint32_t *numberOfDynamicBitmaps* = 0)

Functions

```
TouchGFX generic initialize.
```

Template Parameters

HALType	The class type of the HAL subclass used for this port.

Parameters

in	*dma*	Reference to the DMA implementation object to use. Can be of type NoDMA to disable the use of DMA for rendering.
in	*display*	Reference to the LCD renderer implementation (subclass of LCD). Could be either LCD16bpp for RGB565 UIs, or LCD1bpp for monochrome UIs or LCD24bpp for 24bit displays using RGB888 UIs.
in	*tc*	Reference to the touch controller driver (or NoTouchController to disable touch input).
	width	The *native* display width of the actual display, in pixels. This value is irrespective of whether the concrete UI should be portrait or landscape mode. It must match what the display itself is configured as.
	height	The *native* display height of the actual display, in pixels. This value is irrespective of whether the concrete UI should be portrait or landscape mode. It must match what the display itself is configured as.
in	*bitmapCache*	Optional pointer to starting address of a memory region in which to place the bitmap cache. Usually in external RAM. Pass 0 if bitmap caching is not used.
	bitmapCacheSize	Size of bitmap cache in bytes. Pass 0 if bitmap cache is not used.
	numberOfDynamicBitmaps	Number of dynamic bitmaps.

Returns

A reference to the allocated (and initialized) HAL object.

17.1.3.18 void touchgfx_init ( )

Function to perform touchgfx initialization. This function prototype is only provided as a convention.

17.1.4 Variable Documentation

17.1.4.1 const TypedTextId TYPED_TEXT_INVALID = 0xFFFFU

This type shall be used by the application to define unique IDs for all typed texts in the system. The application shall define typed text IDs in the range [0,number of typed texts - 1].

Chapter 18

Class Documentation

18.1 AbstractButton Class Reference

This class defines an abstract interface for button-like elements.

```
#include <touchgfx/widgets/AbstractButton.hpp>
```

Public Member Functions

- AbstractButton ()

  Constructor.
- virtual void handleClickEvent (const ClickEvent &event)

  Updates the current state of the button.
- void setAction (GenericCallback< const AbstractButton & > &callback)

  Associates an action to be performed when the AbstractButton is clicked.
- virtual bool getPressedState () const

  Function to determine whether this AbstractButton is currently pressed.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- GenericCallback< const AbstractButton & > * action

  The callback to be executed when this AbstractButton is clicked.
- bool pressed

  Is the button pressed or released? True if pressed.

Additional Inherited Members

18.1.1 Detailed Description

This class defines an abstract interface for button-like elements. A button is a clickable element that has two states - pressed or released - and executes an action when the pressed->released transition is made.

See also

Widget

18.1.2 Constructor & Destructor Documentation

18.1.2.1 AbstractButton ( ) [inline]

Constructs an AbstractButton instance in released state without an associated action.

18.1.3 Member Function Documentation

18.1.3.1 bool getPressedState ( ) const [inline], [virtual]

Function to determine whether this AbstractButton is currently pressed.

Returns

true if button is pressed, false otherwise.

18.1.3.2 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_ABSTRACTBUTTON.

Reimplemented from Widget.

Reimplemented in RadioButton, ButtonWithLabel, ButtonWithIcon, Button, TouchArea, and ToggleButton.

18.1.3.3 void handleClickEvent (const ClickEvent & *event*) [virtual]

Updates the current state of the button - pressed or released - and invalidates it.

If a transition from the pressed to the released state was made, the associated action is executed and then the Widget is invalidated.

Parameters

event	Information about the click.

See also

Drawable::handleClickEvent()

Reimplemented from Drawable.

Reimplemented in ToggleButton, RadioButton, and TouchArea.

18.1.3.4 void setAction (GenericCallback< const AbstractButton & > & *callback*) [inline]

Associates an action to be performed when the AbstractButton is clicked.

Parameters

callback	The callback to be executed. The callback will be given a reference to the AbstractButton.

See also

GenericCallback

18.2 AbstractPainter Class Reference

An abstract class for creating painter classes for drawing canvas widgets.

`#include <touchgfx/widgets/canvas/AbstractPainter.hpp>`

Public Member Functions

- AbstractPainter ()

  Default constructor.
- virtual ~AbstractPainter ()

  Destructor.
- void setOffset (uint16_t offsetX, uint16_t offsetY)

  Sets the offset of the area being drawn.
- void setWidgetAlpha (uint8_t alpha)

  Sets widget alpha.
- virtual void render (uint8_t *ptr, int x, int y, unsigned count, const uint8_t *covers)=0

  Paint a designated part of the RenderingBuffer.

Protected Attributes

- int16_t areaOffsetX

  The offset x coordinate of the area being drawn.
- int16_t areaOffsetY

  The offset y coordinate of the area being drawn.
- uint8_t widgetAlpha

  The alpha of the widget using the painter.

18.2.1 Detailed Description

An abstract class for creating painter classes for drawing canvas widgets.

18.2.2 Constructor & Destructor Documentation

18.2.2.1 AbstractPainter ( )

Default constructor.

18.2.2.2 ~AbstractPainter ( ) [virtual]

Destructor.

18.2.3 Member Function Documentation

18.2.3.1 void render (uint8_t * *ptr,* int *x,* int *y,* unsigned *count,* const uint8_t * *covers*) [pure virtual]

Paint a designated part of the RenderingBuffer with respect to the amount of coverage of each pixel given by the parameter covers.

Parameters

in	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implemented in AbstractPainterRGB565, AbstractPainterRGB888, and AbstractPainterBW.

18.2.3.2 void setOffset (uint16_t *offsetX*, uint16_t *offsetY*)

Sets the offset of the area being drawn. This allows render() to calculate the x, y relative to the widget, and not just relative to the invalidated area.

Parameters

offsetX	The offset x coordinate of the invalidated area relative to the widget.
offsetY	The offset y coordinate of the invalidated area relative to the widget.

18.2.3.3 void setWidgetAlpha (uint8_t *alpha*)

Sets the widget alpha to allow an entire canvas widget to easily be faded without changing the painter of the widget.

Parameters

alpha	The alpha.

18.3 AbstractPainterBW Class Reference

A Painter that will paint using a color on a LCD1bpp display.

```
#include <touchgfx/widgets/canvas/AbstractPainterBW.hpp>
```

Public Member Functions

- virtual void render (uint8_t *ptr, int x, int y, unsigned count, const uint8_t *covers)

  Render part of a scan line.

Protected Member Functions

- virtual bool renderInit ()

  Initialize rendering of a single scan line of pixels for the render.
- virtual bool renderNext (uint8_t &color)=0

  Get the color of the next pixel in the scan line.

Protected Attributes

- uint16_t currentX

  Current x coordinate relative to the widget.
- uint16_t currentY

  Current y coordinate relative to the widget.

18.3.1 Detailed Description

AbstractPainterBW is used for drawing one 1bpp displays. The color is either on or off No transparency is supported.

See also

AbstractPainter

18.3.2 Member Function Documentation

18.3.2.1 void render (uint8_t * *ptr*, int *x*, int *y*, unsigned *count*, const uint8_t * *covers*) [virtual]

Fill a designated part of the frame buffer with the given color with respect to the amount of coverage of each pixel given by the parameter covers.

Parameters

in	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

18.3.2.2 bool renderInit ( ) [inline],[protected],[virtual]

Initialize rendering of a single scan line of pixels for the render.

Returns

true if it succeeds, false if it fails.

18.3.2.3 bool renderNext (uint8_t & *color*) [protected],[pure virtual]

Get the color of the next pixel in the scan line.

Parameters

out	*color*	Color of the pixel, 0 or 1.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterBW.

18.4 AbstractPainterRGB565 Class Reference

A Painter that will paint using a color and an alpha value.

```
#include <touchgfx/widgets/canvas/AbstractPainterRGB565.hpp>
```

Public Member Functions

- virtual void render (uint8_t *ptr, int x, int y, unsigned count, const uint8_t *covers)

  Render part of a scan line.

Protected Member Functions

- virtual bool renderInit ()

  Initialize rendering of a single scan line of pixels for the render.
- virtual bool renderNext (uint8_t &red, uint8_t &green, uint8_t &blue, uint8_t &alpha)=0

  Get the color of the next pixel in the scan line.
- virtual void renderPixel (uint16_t *p, uint8_t red, uint8_t green, uint8_t blue)

  Renders the pixel.

Protected Attributes

- int currentX

  Current x coordinate relative to the widget.
- int currentY

  Current y coordinate relative to the widget.

18.4.1 Detailed Description

The AbstractPainterRGB565 class allows a shape to be filled with a given color and alpha value. This allows transparent, anti-aliased elements to be drawn.

See also

AbstractPainter

18.4.2 Member Function Documentation

18.4.2.1 void render (uint8_t * *ptr*, int *x*, int *y*, unsigned *count*, const uint8_t * *covers*) [virtual]

Fill a designated part of the frame buffer with the given color with respect to the amount of coverage of each pixel given by the parameter covers.

Parameters

in	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

18.4.2.2 bool renderInit ( ) [inline],[protected],[virtual]

Initialize rendering of a single scan line of pixels for the render.

Returns

true if it succeeds, false if it fails.

Reimplemented in PainterRGB565Bitmap.

18.4.2.3 bool renderNext (uint8_t & *red*, uint8_t & *green*, uint8_t & *blue*, uint8_t & *alpha*) [protected],[pure virtual]

Get the color of the next pixel in the scan line.

Parameters

out	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterRGB565Bitmap, and PainterRGB565.

18.4.2.4 void renderPixel (uint16_t * *p*, uint8_t *red*, uint8_t *green*, uint8_t *blue*) [protected], [virtual]

Renders the pixel into the frame buffer. The colors are reduced from 8,8,8 to 5,6, 5.

Parameters

in	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.

18.5 AbstractPainterRGB888 Class Reference

A Painter that will paint using a color and an alpha value.

`#include <touchgfx/widgets/canvas/AbstractPainterRGB888.hpp>`

Public Member Functions

- virtual void render (uint8_t *ptr, int x, int y, unsigned count, const uint8_t *covers)

  Render part of a scan line.

Protected Member Functions

- virtual bool renderInit ()

  Initialize rendering of a single scan line of pixels for the render.
- virtual bool renderNext (uint8_t &red, uint8_t &green, uint8_t &blue, uint8_t &alpha)=0

  Get the color of the next pixel in the scan line.
- virtual void renderPixel (uint16_t *p, uint8_t red, uint8_t green, uint8_t blue)

  Renders the pixel.

Protected Attributes

- int currentX

  Current x coordinate relative to the widget.
- int currentY

  Current y coordinate relative to the widget.

18.5.1 Detailed Description

The AbstractPainterRGB888 class allows a shape to be filled with a given color and alpha value. This allows transparent, anti-aliased elements to be drawn.

See also

AbstractPainter

18.5.2 Member Function Documentation

18.5.2.1 void render (uint8_t * *ptr,* int *x,* int *y,* unsigned *count,* const uint8_t * *covers*) [virtual]

Fill a designated part of the frame buffer with the given color with respect to the amount of coverage of each pixel given by the parameter covers.

Parameters

in	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

18.5.2.2 bool renderInit ( ) [inline],[protected],[virtual]

Initialize rendering of a single scan line of pixels for the render.

Returns

true if it succeeds, false if it fails.

Reimplemented in PainterRGB888Bitmap.

18.5.2.3 bool renderNext (uint8_t & *red,* uint8_t & *green,* uint8_t & *blue,* uint8_t & *alpha*) [protected],[pure virtual]

Get the color of the next pixel in the scan line.

Parameters

out	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterRGB888Bitmap, and PainterRGB888.

18.5.2.4 void renderPixel (uint16_t * *p,* uint8_t *red,* uint8_t *green,* uint8_t *blue*) [protected],[virtual]

Renders the pixel into the frame buffer. The colors are reduced from 8,8,8 to 5,6, 5.

Parameters

in	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.

18.6 AbstractPartition Class Reference

This type defines an abstract interface to a storage partition for allocating memory slots of equal size.

`#include <touchgfx/common/AbstractPartition.hpp>`

Public Member Functions

- virtual ~AbstractPartition ()

  Virtual destructor.
- virtual void * allocate (uint16_t size)

  Gets the address of the next available storage slot.
- virtual void * allocateAt (uint16_t index, uint16_t size)

  Gets the address of the specified index.
- virtual uint16_t getAllocationCount () const

  Gets allocation count.
- virtual uint16_t indexOf (const void *address)

  Determines index of previously allocated location.
- virtual void clear ()

  Prepares the Partition for new allocations.
- virtual uint16_t capacity () const =0

  Gets the capacity, i.e. the maximum allocation count.
- template<typename T >
  void * allocate ()

  Gets the address of the next available storage slot.
- template<typename T >
  void * allocateAt (uint16_t index)

  Gets the address of the specified storage slot.
- template<typename T >
  T & at (const uint16_t index)

  Gets the object at the specified index.
- template<typename T >
  const T & at (const uint16_t index) const

  const version of at().
- template<class T >
  Pair< T *, uint16_t > find (const void *pT)

  Determines if the specified object could have been previously allocated in the partition.
- void dec ()

  Decreases number of allocations.
- virtual uint32_t element_size ()=0

  Access to concrete element-size. Used internally.

Protected Member Functions

- virtual void * element (uint16_t index)=0

  Access to stored element. Used internally.
- virtual const void * element (uint16_t index) const =0

  Access to stored element, const version.
- AbstractPartition ()

  Default constructor.

18.6.1 Detailed Description

This type defines an abstract interface to a storage partition for allocating memory slots of equal size. The "partition" is not aware of the actual types stored in the partition memory, hence it provides no mechanism for deleting C++ objects when clear()'ed.

18.6.2 Constructor & Destructor Documentation

18.6.2.1 ~AbstractPartition ( ) [virtual]

Virtual destructor.

18.6.2.2 AbstractPartition ( ) [protected]

Default constructor.

18.6.3 Member Function Documentation

18.6.3.1 void * allocate (uint16_t *size*) [virtual]

Gets the address of the next available storage slot. The slot size is compared with the specified size.

Note

Asserts if 'size' is too large, or the storage is depleted.

Parameters

size	The size.

Returns

The address of an empty storage slot which contains minimum 'size' bytes.

18.6.3.2 template< typename T > void * allocate ( ) [inline]

Gets the address of the next available storage slot. The slot size is determined from the size of type T.

Note

Asserts if T is too large, or the storage is depleted.

Template Parameters

T	Generic type parameter.

Returns

The address of an empty storage slot.

18.6.3.3 void * allocateAt (uint16_t *index*, uint16_t *size*) [virtual]

Gets the address of the specified index.

Note

Asserts if 'size' is too large.

Parameters

index	Zero-based index of the.
size	The size.

Returns

The address of the appropriate storage slot which contains minimum 'size' bytes.

18.6.3.4 template< typename T > void * allocateAt (uint16_t *index*) [inline]

Gets the address of the specified storage slot. The slot size is determined from the size of type T.

Note

Asserts if T is too large.

Template Parameters

T	Generic type parameter.

Parameters

index	Zero-based index of the.

Returns

The address of the appropriate storage slot.

18.6.3.5 template< typename T > T & at (const uint16_t *index*) [inline]

Gets the object at the specified index.

Template Parameters

T	Generic type parameter.

Parameters

index	The index into the Partition storage where the returned object is located.

Returns

A typed reference to the object at the specified index.

18.6.3.6 template< typename T > const T & at (const uint16_t *index*) const [inline]

const version of at().

Template Parameters

T	Generic type parameter.

Parameters

index	Zero-based index of the.

Returns

A T&

18.6.3.7 uint16_t capacity ( ) const [pure virtual]

Gets the capacity, i.e. the maximum allocation count.

Returns

The maximum allocation count.

Implemented in Partition< ListOfTypes, NUMBER_OF_ELEMENTS >.

18.6.3.8 void clear ( ) [virtual]

Prepares the Partition for new allocations. Any objects present in the Partition shall not be used after invoking this method.

18.6.3.9 void dec ( ) [inline]

Decreases number of allocations.

18.6.3.10 void * element (uint16_t *index*) [protected],[pure virtual]

Access to stored element. Used internally.

Parameters

index	Zero-based index of the.

Returns

null if it fails, else a void*.

Implemented in Partition< ListOfTypes, NUMBER_OF_ELEMENTS >.

18.6.3.11 const void * element (uint16_t *index*) const [protected], [pure virtual]

Access to stored element, const version.

Parameters

index	Zero-based index of the.

Returns

null if it fails, else a void∗.

Implemented in Partition< ListOfTypes, NUMBER_OF_ELEMENTS >.

18.6.3.12 uint32_t element_size ( ) [pure virtual]

Access to concrete element-size. Used internally.

Returns

An uint32_t.

Implemented in Partition< ListOfTypes, NUMBER_OF_ELEMENTS >.

18.6.3.13 template< class T > Pair< T ∗, uint16_t > find (const void ∗ *pT*) [inline]

Determines if the specified object could have been previously allocated in the partition. Since the Partition concept is loosely typed this method shall be used with care. The method does not guarantee that the found object at the returned index is a valid object. It only tests whether or not the object is within the bounds of the current partition allocations.

Template Parameters

T	Generic type parameter.

Parameters

pT	Pointer to the object to lookup.

Returns

If the object seems to be allocated in the Partition, a Pair object containing a typed pointer to the object and an index into the Partition storage is returned. Otherwise, a Pair< 0, 0 > is returned.

18.6.3.14 uint16_t getAllocationCount ( ) const [virtual]

Gets allocation count.

Returns

The currently allocated storage slots.

18.6.3.15 uint16_t indexOf (const void ∗ *address*) [virtual]

Determines index of previously allocated location. Since the Partition concept is loosely typed this method shall be used with care. The method does not guarantee that the found object at the returned index is a valid object. It only tests whether or not the object is within the bounds of the current partition allocations.

Parameters

address	The location address to lookup.

Returns

An uint16_t.

18.7 AbstractShape Class Reference

Simple widget capable of drawing a abstractShape.

```
#include <touchgfx/widgets/canvas/AbstractShape.hpp>
```

Classes

- struct ShapePoint

  Defines an alias representing the array of points making up the abstract shape.

Public Member Functions

- AbstractShape ()

  Constructs a new AbstractShape.
- virtual ~AbstractShape ()

  Virtual Destructor.
- virtual int getNumPoints () const =0

  Gets number points used to make up the shape.
- virtual void setCorner (int i, CWRUtil::Q5 x, CWRUtil::Q5 y)=0

  Sets a corner of the shape.
- virtual CWRUtil::Q5 getCornerX (int i) const =0

  Gets the x coordinate of a corner.
- virtual CWRUtil::Q5 getCornerY (int i) const =0

  Gets the y coordinate of a corner.
- virtual void setCache (int i, CWRUtil::Q5 x, CWRUtil::Q5 y)=0

  Sets the cached coordinates of a given corner.
- virtual CWRUtil::Q5 getCacheX (int i) const =0

  Gets cached x coordinate of a corner.
- virtual CWRUtil::Q5 getCacheY (int i) const =0

  Gets cached y coordinate of a corner.
- template<class T >
  void setShape (ShapePoint< T > *points)

  Sets a shape the struct Points.
- template<class T >
  void setOrigin (T x, T y)

  Sets the position of (0,0).
- template<class T >
  void moveOrigin (T x, T y)

  Moves the start point for this AbstractShape.
- template<class T >
  void getOrigin (T &dx, T &dy) const

  Gets the start coordinates for the line.
- void setAngle (int angle)

Sets the angle to turn the abstractShape.

- void updateAngle (int angle)

  Sets the angle to turn the abstractShape.
- int getAngle () const

  Gets the current angle of the abstractShape.
- template<class T >
  void setScale (T xScale, T yScale)

  Scale the AbstractShape.
- template<class T >
  void setScale (T scale)

  Scale the AbstractShape.
- virtual bool drawCanvasWidget (const Rect &invalidatedArea) const

  Draws the AbstractShape.

Protected Member Functions

- void updateAbstractShapeCache ()

  Updates the abstractShape cache.
- virtual Rect getMinimalRect () const

  Gets minimal rectangle containing the abstractShape.

Additional Inherited Members

18.7.1 Detailed Description

Simple widget capable of drawing a abstractShape. The abstractShape can be scaled and rotated around 0,0. Note that the y axis goes down, so a abstractShape that goes up must be given negative coordinates.

See also

CanvasWidget

tparam T The type of the points used for the abstractShape. Must be int or float.

18.7.2 Constructor & Destructor Documentation

18.7.2.1 AbstractShape ( )

Constructs a new AbstractShape.

18.7.2.2 ~AbstractShape ( ) [virtual]

Virtual Destructor.

18.7.3 Member Function Documentation

18.7.3.1 bool drawCanvasWidget (const Rect & *invalidatedArea*) const [virtual]

Draws the AbstractShape. This class supports partial drawing, so only the area described by the rectangle will be drawn.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Returns

true if it succeeds, false if it fails.

Implements CanvasWidget.

18.7.3.2 int getAngle ( ) const

Gets the current angle of the abstractShape.

Returns

The angle of the AbstractShape.

18.7.3.3 CWRUtil::Q5 getCacheX (int *i*) const [pure virtual]

Gets cached x coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The cached x coordinate.

Implemented in Shape< POINTS >.

18.7.3.4 CWRUtil::Q5 getCacheY (int *i*) const [pure virtual]

Gets cached y coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The cached y coordinate.

Implemented in Shape< POINTS >.

18.7.3.5 CWRUtil::Q5 getCornerX (int *i*) const [pure virtual]

Gets the x coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The corner x coordinate.

Implemented in Shape< POINTS >.

18.7.3.6 CWRUtil::Q5 getCornerY (int *i*) const [pure virtual]

Gets the y coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The corner y coordinate.

Implemented in Shape< POINTS >.

18.7.3.7 Rect getMinimalRect ( ) const [protected], [virtual]

Gets minimal rectangle containing the abstractShape. Used for invalidating only the required part of the screen.

Returns

The minimal rectangle.

Reimplemented from CanvasWidget.

18.7.3.8 int getNumPoints ( ) const [pure virtual]

Gets number points used to make up the shape.

Returns

The number points.

Implemented in Shape< POINTS >.

18.7.3.9 template< class T > void getOrigin (T & *dx*, T & *dy*) const [inline]

Gets the start coordinates for the line.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*dx*	The x coordinate.
out	*dy*	The y coordinate.

18.7.3.10 template< class T > void moveOrigin (T *x*, T *y*) [inline]

Moves the start point for this AbstractShape. The rectangle that surrounds the old and new area covered by the shape will be invalidated.

Note

The area containing the AbstractShape is invalidated before and after the change.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the shapes position (0,0).
y	The y coordinate of the shapes position (0,0).

See also

setOrigin()

18.7.3.11 void setAngle (int *angle*)

Sets the angle to turn the abstractShape. 0 degrees is straight up and 90 degrees is 3 o'clock.

Note

The area containing the AbstractShape is not invalidated.

Parameters

angle	The angle to turn the abstractShape to relative to 0 (straight up), not relative to the previous angle.

See also

updateAngle()

18.7.3.12 void setCache (int *i*, CWRUtil::Q5 *x*, CWRUtil::Q5 *y*) [pure virtual]

Sets the cached coordinates of a given corner. The coordinates in the cache are the coordinates from the corners after rotating and scaling the coordinate.

Parameters

i	Zero-based index of the corner.
x	The x coordinate.
y	The y coordinate.

Implemented in Shape< POINTS >.

18.7.3.13 void setCorner (int *i*, CWRUtil::Q5 *x*, CWRUtil::Q5 *y*) [pure virtual]

Sets a corner of the shape in Q5 format.

Parameters

i	Zero-based index of the corner.
x	The x coordinate in Q5 format.
y	The y coordinate in Q5 format.

Implemented in Shape< POINTS >.

18.7.3.14 template< class T > void setOrigin (T *x*, T *y*) [inline]

Sets the position of (0,0) used when the abstractShape was created. This means that all coordinates initially used when created the shape are moved relative to these given offsets. Calling setOrigin() again, will not add to the previous settings of setOrigin() but will replace the old values for origin.

Note

The area containing the AbstractShape is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the shapes position (0,0).
y	The y coordinate of the shapes position (0,0).

See also

moveOrigin()

18.7.3.15 template< class T > void setScale (T *xScale*, T *yScale*) [inline]

Scale the AbstractShape the given amount in the x direction and the y direction.

Note

The area containing the AbstractShape is not invalidated.

Template Parameters

T	Generic type parameter.

Parameters

xScale	The scale in the x direction.
yScale	The scale in the y direction.

See also

getScale(), updateScale()

18.7.3.16 template< class T > void setScale (T *scale*) [inline]

Scale the AbstractShape the given amount in the x direction and the y direction.

Note

The area containing the AbstractShape is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

scale	The scale in the x direction.

See also

getScale()

18.7.3.17 template< class T > void setShape (ShapePoint< T > * *points*) [inline]

Sets a shape the struct Points.

Note

The area containing the shape is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

in	*points*	The points that make up the shape.

18.7.3.18 void updateAbstractShapeCache ( ) [protected]

Updates the abstractShape cache. The cache is used to be able to quickly redraw the AbstractShape without calculating the points that make up the abstractShape (with regards to scaling and rotation).

18.7.3.19 void updateAngle (int *angle*)

Sets the angle to turn the abstractShape. 0 degrees is straight up and 90 degrees is 3 o'clock.

Note

The area containing the AbstractShape is invalidated before and after the change.

Parameters

angle	The angle to turn the abstractShape.

See also

setAngle()

18.8 AnimatedImage Class Reference

A widget capable of basic animation using a range of bitmaps.

```
#include <touchgfx/widgets/AnimatedImage.hpp>
```

Public Member Functions

- AnimatedImage (const BitmapId &start, const BitmapId &end, const uint8_t &updateInterval=1)

  Constructs an AnimatedImage.
- AnimatedImage (const uint8_t &updateInterval=1)

  Constructor.
- virtual void startAnimation (const bool &rev, const bool &reset=false, const bool &loop=false)

  Starts the animation.
- virtual void stopAnimation ()

  Stops and resets the animation.
- virtual void pauseAnimation ()

  Toggles the running state of an animation.
- virtual void handleTickEvent ()

  Called periodically by the framework if the Drawable instance has subscribed to timer ticks.
- void setDoneAction (GenericCallback< const AnimatedImage & > &callback)

  Associates an action to be performed when the animation of the AnimatedImage is done.
- bool isRunning ()

  Gets the running state of the AnimatedImage.
- bool isReverse ()

  Query if this object is running in reverse.
- void setBitmaps (BitmapId start, BitmapId end)

  Sets the bitmaps that are used by the animation.
- void setUpdateTicksInterval (uint8_t updateInterval)

  Sets the update interval.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- GenericCallback< const AnimatedImage & > * animationDoneAction

  Pointer to the callback being executed when animation is done.
- BitmapId startId

  Id of first bitmap in animation.
- BitmapId endId

  Id of second bitmap in animation.S.
- uint8_t updateTicksInterval

  Number of ticks required between each animation update (image change).
- uint8_t ticksSinceUpdate

  Number of ticks since last animation update.
- bool reverse

  If true, run in reverse direction (last to first).
- bool loopAnimation

  If true, continuously loop animation.
- bool running

  If true, animation is running.

Additional Inherited Members

18.8.1 Detailed Description

The AnimatedImage is capable of running the animation from start to end or in reverse order, end to start. It is capable doing a single animation or looping the animation until stopped or paused. See animation_example far a demonstration of how to use of this widget.

18.8.2 Constructor & Destructor Documentation

18.8.2.1 AnimatedImage (const BitmapId & *start*, const BitmapId & *end*, const uint8_t & *updateInterval* = 1) [inline]

The start and the end specifies the range of bitmaps to be used for animation. The update interval defines how often the animation should be updated. The animation will iterate over the bitmaps that lies between the IDs of start and end, both included.

Parameters

start	Defines the start of the range of images in the animation.
end	Defines the end of the range of images in the animation.
updateInterval	Defines the number of ticks between each animation step. Higher value results in a slower animation.

18.8.2.2 AnimatedImage (const uint8_t & *updateInterval* = 1) [inline]

Constructs an AnimatedImage without initializing bitmaps.

Note

The bitmaps to display must be configured through set setBitmaps function before this widget displays anything.

Parameters

updateInterval	Defines the number of ticks between each animation step. Higher value results in a slower animation.

18.8.3 Member Function Documentation

18.8.3.1 uint16_t getType ( ) const [inline],[virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_ANIMATEDIMAGE.

Reimplemented from Image.

18.8.3.2 virtual void handleTickEvent ( ) [virtual]

Called periodically by the framework if the Drawable instance has subscribed to timer ticks.

See also

Application::registerTimerWidget

Reimplemented from Drawable.

18.8.3.3 bool isReverse ( ) [inline]

Query if this object is running in reverse.

Returns

true if the animation is performed in reverse order.

18.8.3.4 bool isRunning ( ) [inline]

Gets the running state of the AnimatedImage.

Returns

true if the animation is currently running, false otherwise.

18.8.3.5 void pauseAnimation ( ) [virtual]

Toggles the running state of an animation. Pauses the animation if the animation is running. Continues the animation if previously paused.

18.8.3.6 void setBitmaps (BitmapId *start*, BitmapId *end*)

Sets the bitmaps that are used by the animation.

The animation will iterate over the bitmaps that lies between the IDs of start and end, both included.

Parameters

start	Defines the start of the range of images in the animation.
end	Defines the end of the range of images in the animation.

18.8.3.7 void setDoneAction (GenericCallback< const AnimatedImage & > & *callback*) [inline]

Associates an action to be performed when the animation of the AnimatedImage is done.

Parameters

callback	The callback is executed when done. The callback is given the animated image.

18.8.3.8 void setUpdateTicksInterval (uint8_t *updateInterval*)

Sets the update interval.

Parameters

updateInterval	Defines the number of ticks between each animation step. Higher value results in a slower animation.

18.8.3.9 void startAnimation (const bool & *rev*, const bool & *reset* = `false`, const bool & *loop* = `false`) `[virtual]`

Starts the animation.

Parameters

rev	Defines if the animation should be performed in reverse order.
reset	Defines if the animation should reset and start from the first (or last if reverse order) bitmap.
loop	Defines if the animation should loop or do a single animation.

18.8.3.10 void stopAnimation ( ) `[virtual]`

Stops and resets the animation.

18.9 Application Class Reference

The Application class is the main interface for manipulating screen contents.

```
#include <touchgfx/Application.hpp>
```

Public Member Functions

- Screen * getCurrentScreen ()

  Gets the current screen.
- virtual void switchScreen (Screen *newScreen)

  Switch to another Screen.
- virtual void appSwitchScreen (uint8_t screenId)

  An application specific function for switching screen.
- virtual void draw ()

  Initiate a draw operation of the entire screen.
- virtual void draw (Rect &rect)

  Initiate a draw operation of the specified region of the screen.
- virtual void handleClickEvent (const ClickEvent &evt)

  Handle a click event.
- virtual void handleDragEvent (const DragEvent &evt)

  Handle drag events.
- virtual void handleGestureEvent (const GestureEvent &evt)

  Handle gestures.
- virtual void handleTickEvent ()

  Handle tick.
- virtual void handleKeyEvent (uint8_t c)

  Handle an incoming character received by the HAL layer.
- virtual void handlePendingScreenTransition ()

  Evaluates the pending Callback instances.
- virtual void cacheDrawOperations (bool enableCache)

  This functions allows for deferring draw operations to a later time.

- void registerTimerWidget (Drawable *w)

  Adds a widget to the list of widgets receiving ticks.
- void clearAllTimerWidgets ()

  Clears all currently registered timer widgets.
- void unregisterTimerWidget (const Drawable *w)

  Removes a widget from the list of widgets receiving ticks.

Static Public Member Functions

- static Application * getInstance ()

  Gets the single instance application.

Static Public Attributes

- static const uint8_t MAX_TIMER_WIDGETS = 24

  Maximum number of widgets receiving ticks.
- static const uint16_t TICK_INTERVAL_MS = 10

  Tick interval in milliseconds. How accurate this is depends on platform.

Protected Member Functions

- void invalidateArea (Rect area)

  Invalidates this area.
- Application ()

  Procected constructor.

Protected Attributes

- Vector< Drawable *, MAX_TIMER_WIDGETS > timerWidgets

  List of widgets that receive timer ticks.
- uint8_t timerWidgetCounter [MAX_TIMER_WIDGETS]

  A counter for each potentially registered timer widget. Increase when registering for timer events, decrease when unregistering.
- Vector< Rect, 8 > cachedDirtyAreas

  When draw caching is enabled, these rects keeps track of the dirty screen area.
- Vector< Rect, 8 > lastRects

  The dirty areas from last frame that needs to be redrawn because we have swapped frame buffers.
- bool drawCacheEnabled

  True when draw caching is active.
- bool transitionHandled

  True if the transition is done and Screen::afterTransition has been called.

Static Protected Attributes

- static Screen * currentScreen

  Pointer to currently displayed Screen.
- static Transition * currentTransition

  Pointer to current transition.
- static Application * instance

18.9.1 Detailed Description

The Application class is the main interface for manipulating screen contents. It holds a pointer to the currently displayed Screen, and delegates draw requests and events to that Screen. Additionally it contains some global application settings.

A user-defined application subclass can be defined to override standard functionality.

See also

UIEventListener

18.9.2 Constructor & Destructor Documentation

18.9.2.1 Application ( ) [protected]

Procected constructor.

18.9.3 Member Function Documentation

18.9.3.1 void appSwitchScreen (uint8_t *screenId*) [inline],[virtual]

An application specific function for switching screen. Overloading this can provide a means to switch screen from places that does not have access to a pointer to the new screen. Base implementation is empty.

Parameters

screenId	An id that maps to the desired screen.

18.9.3.2 void cacheDrawOperations (bool *enableCache*) [virtual]

This functions allows for deferring draw operations to a later time. If active, calls to draw will simply note that the specified area is dirty, but not perform any actual drawing. When disabling the draw cache, the dirty area will be flushed (drawn) immediately.

Parameters

enableCache	if true, all future draw operations will be cached. If false draw caching is disabled, and the current cache (if not empty) is drawn immediately.

18.9.3.3 void clearAllTimerWidgets ( )

Clears all currently registered timer widgets.

18.9.3.4 void draw ( ) [virtual]

Initiate a draw operation of the entire screen. Standard implementation is to delegate draw request to the current Screen.

18.9.3.5 void draw (Rect & *rect*) [virtual]

Initiate a draw operation of the specified region of the screen. Standard implementation is to delegate draw request to the current Screen.

Note

Unlike Widget::draw this is safe to call from user code as it will properly traverse widgets in z-order.
The coordinates given must be absolute coordinates.

Parameters

in	*rect*	The area to draw.

18.9.3.6 Screen * getCurrentScreen ( ) [inline]

Gets the current screen.

Returns

The current screen.

18.9.3.7 static Application * getInstance ( ) [static]

Gets the single instance application.

Returns

The instance of this application.

18.9.3.8 void handleClickEvent (const ClickEvent & *evt*) [virtual]

Handle a click event. Standard implementation is to delegate the event to the current screen. Called by the framework when a click is detected by some platform specific means.

Parameters

evt	The ClickEvent.

Reimplemented from UIEventListener.

18.9.3.9 void handleDragEvent (const DragEvent & *evt*) [virtual]

Handle drag events. Called by the framework when a drag is detected by some platform specific means. Standard implementation is to delegate drag event to current screen.

Parameters

evt	The drag event, expressed in absolute coordinates.

Reimplemented from UIEventListener.

18.9.3.10 void handleGestureEvent (const GestureEvent & *evt*) [virtual]

Handle gestures. Called by the framework when a gesture is detected by some platform specific means. Standard implementation is to delegate drag event to current screen.

Parameters

evt	The gesture event.

Reimplemented from UIEventListener.

18.9.3.11 void handleKeyEvent (uint8_t *c*) [virtual]

Handle an incoming character received by the HAL layer. Standard implementation delegates to current screen (which, in turn, does nothing).

Parameters

c	The incomming character to handle.

Reimplemented from UIEventListener.

18.9.3.12 void handlePendingScreenTransition ( ) [virtual]

Evaluates the pending Callback instances. If a callback is valid, it is executed and a Screen transition is executed. This base implementation is empty and does nothing.

Reimplemented from UIEventListener.

Reimplemented in MVPApplication.

18.9.3.13 void handleTickEvent ( ) [virtual]

Handle tick. Standard implementation is to delegate tick to the widgets that have registered to receive one. Called by some platform specific means.

Reimplemented from UIEventListener.

18.9.3.14 void invalidateArea (Rect *area*) [protected]

Invalidates this area.

Parameters

area	The area to invalidate.

18.9.3.15 void registerTimerWidget (Drawable * *w*)

Adds a widget to the list of widgets receiving ticks every TICK_INTERVAL_MS milliseconds.

Note

The framework keeps track of the number of times a specific widget is registered.

Parameters

in	*w*	The widget to add.

See also

unregisterTimerWidget

18.9.3.16 void switchScreen (Screen * *newScreen*) [virtual]

Switch to another Screen. Will call tearDownScreen on current Screen before switching, and subsequently call setupScreen and draw automatically for the new Screen.

Parameters

in	*newScreen*	A pointer to the new screen.

18.9.3.17 void unregisterTimerWidget (const Drawable * *w*)

Removes a widget from the list of widgets receiving ticks every TICK_INTERVAL_MS milliseconds.

Note

If widget has been registered multiple times, an equal number of calls to unregister are required to stop widget from receiving tick events.

Parameters

in	*w*	The widget to remove.

18.9.4 Member Data Documentation

18.9.4.1 Application* instance [static],[protected]

Pointer to the instance of the Application-derived subclass.

Note

Must be set by subclass constructor!

18.9.4.2 const uint8_t MAX_TIMER_WIDGETS = 24 [static]

Remarks

Memory impact: x * (sizeof(Drawable*)+1)

18.10 Bitmap Class Reference

This class provides a proxy object for a bitmap image.

```
#include <touchgfx/Bitmap.hpp>
```

Classes

- struct BitmapData

  Data of a bitmap.
- struct CacheTableEntry

  Cache bookkeeping.
- struct DynamicBitmapData

  Data of a dynamic bitmap.

Public Types

- enum BitmapFormat { RGB565, RGB888, ARGB8888, BW, BW_RLE }

  Data of a bitmap can be stored in the following formats.

Public Member Functions

- Bitmap (const BitmapId id=BITMAP_INVALID)

  Creates and binds a Bitmap instance to the corresponding entry in the BitmapData array.
- BitmapId getId () const

  Gets the id of this Bitmap.
- const uint8_t * getData () const

  Gets a pointer to the Bitmap data.
- const uint8_t * getAlphaData () const

  Gets a pointer to the alpha data, if present in the bitmap.
- BitmapFormat getFormat () const

  Gets the format of how the bitmap is stored.
- uint16_t getWidth () const

  Gets the width of the Bitmap in pixels.
- uint16_t getHeight () const

  Gets the height of the Bitmap in pixels.
- Rect getRect () const

  Gets the rectangle describing the dimensions of the Bitmap.
- bool isAlphaPerPixel () const

  Query if this object has an alpha channel.
- Rect getSolidRect () const

  Gets the largest solid rectangle in the bitmap.
- bool hasTransparentPixels () const

  Query if this object has transparent pixels.
- bool operator== (const Bitmap &other) const

  Equality operator.
- bool operator!= (const Bitmap &other) const

  Inequality operator.

Static Public Member Functions

- static void registerBitmapDatabase (const BitmapData *data, const uint16_t n, uint16_t *cachep=0, uint32_t csize=0, uint32_t numberOfDynamicBitmaps=0)

  Registers an array of bitmaps.
- static bool cache (BitmapId id)

  Cache this bitmap into RAM.
- static bool cacheReplaceBitmap (BitmapId out, BitmapId in)

  Replace a bitmap in RAM with another Bitmap.
- static bool cacheRemoveBitmap (BitmapId id)

  Remove this bitmap from the RAM cache.
- static uint8_t * cacheGetAddress (BitmapId id)

  Get address of cache buffer for this bitmap.
- static bool cacheIsCached (BitmapId id)

  Check if the Bitmap is cached.
- static bool cacheAll ()

  Cache all bitmaps from the Bitmap Database into RAM.
- static void clearCache ()

  Clears the cached bitmaps from RAM.
- static BitmapId dynamicBitmapCreate (const uint16_t width, const uint16_t height, BitmapFormat format)

  Create a dynamic bitmap.
- static bool dynamicBitmapDelete (BitmapId id)

Delete a dynamic bitmap.

- static uint8_t * dynamicBitmapGetAddress (BitmapId id)

  Get the address of the dynamic bitmap data.
- static bool dynamicBitmapSetSolidRect (BitmapId id, const Rect &solidRect)

  Set the solid rectangle of a dynamic bitmap.
- static void setCache (uint16_t *cachep, uint32_t csize, uint32_t numberOfDynamicBitmaps=0)

  Register a memory region in which bitmap data can be cached.

18.10.1 Detailed Description

This class provides a proxy object for a bitmap image stored in the application specific bitmap database. The proxy provides access to the raw bitmap data as well as metadata.

18.10.2 Member Enumeration Documentation

18.10.2.1 enum BitmapFormat

Data of a bitmap can be stored in the following formats.

Enumerator

RGB565 16-bit, 5 bits for red, 6 bits for green, 5 bits for blue, no alpha channel

RGB888 24-bit, 8 bits for each of red, green and blue, no alpha channel

ARGB8888 32-bit, 8 bits for each of red, green, blue and alpha channel

BW 1-bit, black / white, no alpha channel

BW_RLE 1-bit, black / white, no alpha channel compressed with horizontal RLE

18.10.3 Constructor & Destructor Documentation

18.10.3.1 Bitmap (const BitmapId *id* = BITMAP_INVALID) [inline],[explicit]

Creates and binds a Bitmap instance to the corresponding entry in the BitmapData array.

Parameters

id	The unique bitmap identifier.

18.10.4 Member Function Documentation

18.10.4.1 static bool cache (BitmapId *id*) [static]

Cache this bitmap into unused RAM in the bitmap cache.

Note

A memory region large enough to hold this bitmap must be configured and a large enough part of it must be available. Caching of a bitmap may involve a defragmentation of the bitmap cache. See TouchGFX documentation for details on caching.

Parameters

id	The id of the bitmap to cache.

Returns

true if caching went well, false otherwise.

See also

registerBitmapDatabase

18.10.4.2 static bool cacheAll ( ) [static]

Cache all bitmaps from the Bitmap Database into RAM.

Note

A memory region large enough to hold all bitmaps must be configured. See TouchGFX documentation for details on caching.

Returns

True if all bitmaps where cached.

See also

cache.

18.10.4.3 static uint8_t * cacheGetAddress (BitmapId *id*) [static]

Get address of cache buffer for this bitmap. Note: The address is only valid until next Bitmap::cache() call.

Parameters

id	The id of the bitmap in cache.

Returns

Address if bitmap was found, zero otherwise.

18.10.4.4 static bool cacheIsCached (BitmapId *id*) [static]

Check if the Bitmap is cached.

Parameters

id	The id of the bitmap.

Returns

true if bitmap is cached.

18.10.4.5 static bool cacheRemoveBitmap (BitmapId *id*) [static]

Remove this bitmap from the RAM cache.

Note

The bitmap will be removed from the RAM cache. Unless the bitmap is otherwise stored in (slow) RAM it can not be drawn anymore and must be cached again before use. The RAM freed can be used for caching of another bitmap. See TouchGFX documentation for details on caching.

Parameters

id	The id of the bitmap to cache.

Returns

true if bitmap was found and removed, false otherwise.

See also

registerBitmapDatabase

18.10.4.6 static bool cacheReplaceBitmap (BitmapId *out*, BitmapId *in*) [static]

Replace a bitmap in RAM with another Bitmap. The Bitmaps must have same size.

Parameters

out	The id of the bitmap to remove from the cache.
in	The id of the bitmap to cache.

Returns

true if the replacement went well, false otherwise.

18.10.4.7 static void clearCache ( ) [static]

Clears the cached bitmaps from RAM.

18.10.4.8 static BitmapId dynamicBitmapCreate (const uint16_t *width*, const uint16_t *height*, BitmapFormat *format*) [static]

Create a dynamic bitmap.

Parameters

width	Width of the bitmap.
height	Height of the bitmap.
format	Bitmap format of the bitmap.

Returns

BitmapId of the new bitmap or BITMAP_INVALID if memory full.

18.10.4.9 static bool dynamicBitmapDelete (BitmapId *id*) [static]

Parameters

id	The BitmapId of the dynamic bitmap.

Returns

true if it succeeds, false if it fails.

18.10.4.10 static uint8_t * dynamicBitmapGetAddress (BitmapId *id*) [static]

Parameters

id	The BitmapId of the dynamic bitmap.

Returns

null if it fails, else an uint8_t*.

18.10.4.11 static bool dynamicBitmapSetSolidRect (BitmapId *id*, const Rect & *solidRect*) [static]

Set the solid rectangle of a dynamic bitmap. Only relevant for RGBA8888 bitmaps. The solid part of the bitmap is drawn faster than the transparent parts.

Parameters

id	The identifier.
solidRect	The solid rectangle.

Returns

true if it succeeds, false if it fails.

18.10.4.12 const uint8_t * getAlphaData ( ) const

Gets a pointer to the alpha data, if present in the bitmap.

Note

If this bitmap is cached, it will return the cached version of alpha data for this bitmap.

Returns

A pointer to the raw alpha channel data. If no alpha channel exist 0 is returned.

18.10.4.13 const uint8_t * getData ( ) const

Gets a pointer to the Bitmap data.

Note

If this bitmap is cached, it will return the cached version of bitmap data.

Returns

A pointer to the raw bitmap data.

18.10.4.14 BitmapFormat getFormat ( ) const

Gets the format of how the bitmap is stored.

Returns

The format of how the bitmap data is stored.

18.10.4.15 uint16_t getHeight ( ) const

Gets the height of the Bitmap in pixels.

Returns

The bitmap height in pixels.

18.10.4.16 BitmapId getId ( ) const [inline]

Gets the id of this Bitmap.

Returns

The id of this Bitmap.

18.10.4.17 Rect getRect ( ) const [inline]

Gets the rectangle describing the dimensions of the Bitmap.

Returns

a Rect describing the dimensions of this bitmap.

18.10.4.18 Rect getSolidRect ( ) const

Gets the largest solid, i.e. not transparent, rectangle in the bitmap.

Returns

The maximum solid rectangle of the bitmap.

18.10.4.19 uint16_t getWidth ( ) const

Gets the width of the Bitmap in pixels.

Returns

The bitmap width in pixels.

18.10.4.20 bool hasTransparentPixels ( ) const

Returns

True if this bitmap has transparent pixels.

18.10.4.21 bool isAlphaPerPixel ( ) const [inline]

Query if this object has an alpha channel.

Returns

True if the bitmap contains an alpha channel (an alpha value for each pixel)

18.10.4.22 bool operator!= (const Bitmap & *other*) const [inline]

Inequality operator.

Parameters

other	The bitmap to compare with.

Returns

True if this bitmap has a different id than the other bitmap.

18.10.4.23 bool operator== (const Bitmap & *other*) const [inline]

Equality operator.

Parameters

other	The bitmap to compare with.

Returns

True if this bitmap has the same id as the other bitmap.

18.10.4.24 static void registerBitmapDatabase (const BitmapData * *data*, const uint16_t *n*, uint16_t * *cachep* = 0, uint32_t *csize* = 0, uint32_t *numberOfDynamicBitmaps* = 0) [static]

Registers an array of bitmaps. All Bitmap instances are bound to this database.

Parameters

	data	A reference to the BitmapData storage array.
	n	The number of bitmaps in the array.
in,out	*cachep*	(Optional) Pointer to memory region in which bitmap data can be cached.
	csize	Size of cache memory region in bytes (0 if unused)
	numberOf↩ DynamicBitmaps	Number of dynamic bitmaps to be allowed in the cache.

18.10.4.25 static void setCache (uint16_t * *cachep*, uint32_t *csize*, uint32_t *numberOfDynamicBitmaps* = 0) [static]

Parameters

in,out	*cachep*	Pointer to memory region in which bitmap data can be cached.

	csize	Size of cache memory region in bytes.
	numberOf↩ DynamicBitmaps	Number of dynamic bitmaps to be allowed in the cache.

18.11 Bitmap::BitmapData Struct Reference

Data of a bitmap.

```
#include <touchgfx/Bitmap.hpp>
```

Public Attributes

- const uint8_t ∗const data

  The data of this bitmap.
- const uint8_t ∗const alphaData

  The data of the alpha channel (contains 0 if no alpha channel exist)
- const uint16_t width

  The width of the bitmap.
- const uint16_t height

  The height of the bitmap.
- const uint16_t solidRect_x

  The x coordinate of the maximum solid rectangle of the bitmap.
- const uint16_t solidRect_y

  The y coordinate of the maximum solid rectangle of the bitmap.
- const uint16_t solidRect_width

  The width of the maximum solid rectangle of the bitmap.
- const uint16_t solidRect_height: 13

  The height of the maximum solid rectangle of the bitmap.
- const uint8_t format: 3

  Determine the format of the data.

18.11.1 Detailed Description

Data of a bitmap.

18.12 BlitOp Struct Reference

BlitOp instances carry the required information for performing operations on the LCD (frame buffer) using DMA.

```
#include <touchgfx/hal/BlitOp.hpp>
```

Public Attributes

- uint32_t operation

  The operation to perform.
- const uint16_t ∗ pSrc

  Pointer to the source.
- uint16_t ∗ pDst

  Pointer to the destination.

- uint16_t nSteps

  The number of pixels in a line.
- uint16_t nLoops

  The number of lines.
- uint16_t srcLoopStride

  The number of bytes to stride the source after every loop.
- uint16_t dstLoopStride

  The number of bytes to stride the destination after every loop.
- colortype color

  Color to fill.
- uint8_t alpha

  The alpha to use.

18.12.1 Detailed Description

BlitOp instances carry the required information for performing operations on the LCD (frame buffer) using DMA.

18.12.2 Member Data Documentation

18.12.2.1 uint32_t operation

See also

BlitOperations

18.13 Box Class Reference

Simple widget capable of showing a rectangle of a specific color and an optional alpha.

```
#include <touchgfx/widgets/Box.hpp>
```

Public Member Functions

- Box ()

  Constructor.
- Box (uint16_t width, uint16_t height, colortype color, uint8_t alpha=255)

  Constructor.
- virtual Rect getSolidRect () const

  Pure virtual function for obtaining the largest possible rectangle that is guaranteed to be solid (non-transparent).
- void setColor (colortype color)

  Sets the color of the rectangle.
- colortype getColor () const

  Gets the current color of the Box.
- void setAlpha (uint8_t alpha)

  Sets the alpha value for this Box.
- uint8_t getAlpha () const

  Returns the current alpha value.
- virtual void draw (const Rect &area) const

  Draws the box.
- void forceReportAsSolid (bool solid)

  Override solid area for the Box.

- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- uint8_t alpha

  The alpha value used for this Box.
- colortype color

  The fill color for this Box.
- bool reportAsSolid

Additional Inherited Members

18.13.1 Detailed Description

Simple widget capable of showing a rectangle of a specific color and an optional alpha.

See also

Widget

18.13.2 Constructor & Destructor Documentation

18.13.2.1 Box ( ) [inline]

Constructs a new Box with a default alpha value of 255 (solid)

18.13.2.2 Box (uint16_t *width*, uint16_t *height*, colortype *color*, uint8_t *alpha* = 255) [inline]

Construct a Box.

Parameters

width	The width of the box.
height	The height of the box.
color	The color of the box.
alpha	The alpha of the box. Default is 255 (solid).

18.13.3 Member Function Documentation

18.13.3.1 void draw (const Rect & *area*) const [virtual]

Draws the Box.

Parameters

area	The rectangle to draw, with coordinates relative to the containing layer.

Implements Drawable.

18.13.3.2 void forceReportAsSolid (bool *solid*) [inline]

If this is set, getSolidRect() will report the widget as completely solid even if is (semi-)transparent.

Note

Very rarely used in practice.

Parameters

solid	true if this Box should report as solid, even when not.

18.13.3.3 uint8_t getAlpha ( ) const [inline]

Returns

Gets the current alpha value of the Box.

18.13.3.4 colortype getColor ( ) const [inline]

Gets the current color of the Box.

Returns

The current color.

18.13.3.5 virtual Rect getSolidRect ( ) const [virtual]

Pure virtual function for obtaining the largest possible rectangle that is guaranteed to be solid (non-transparent). Used by JSMOC to prune the draw graph.

Note

The rectangle returned must be relative to (0, 0), meaning that to indicate a completely solid widget, Rect(0, 0, getWidth(), getHeight()) must be returned.

Returns

The solid rect.

Implements Drawable.

18.13.3.6 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_BOX.

Reimplemented from Widget.

18.13.3.7 void setAlpha (uint8_t *alpha*) [inline]

Sets the alpha value for this Box.

Parameters

alpha	The alpha value. 255 = completely solid.

18.13.3.8 void setColor (colortype *color*) [inline]

Sets the color of the rectangle.

Parameters

color	The color of the box.

18.13.4 Member Data Documentation

18.13.4.1 bool reportAsSolid [protected]

See also

forceReportAsSolid.

18.14 Button Class Reference

A button with two states.

```
#include <touchgfx/widgets/Button.hpp>
```

Public Member Functions

- Button ()

  Default constructor.
- virtual ~Button ()

  Destructor.
- virtual void draw (const Rect &invalidatedArea) const

  Draws the given invalidated area.
- virtual void setBitmaps (const Bitmap &bmpReleased, const Bitmap &bmpPressed)

  Sets the bitmaps used by this button.
- virtual Rect getSolidRect () const

  Gets solid rectangle.
- void setAlpha (uint8_t alpha)

  Sets the alpha value for the image.
- uint8_t getAlpha () const

  Gets the current alpha value.
- Bitmap getCurrentlyDisplayedBitmap () const

  Gets currently displayed bitmap.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- Bitmap up

  The image to display when button is released.
- Bitmap down

The image to display when button is pressed.

- uint8_t alpha

  The current alpha value. 255 denotes solid, 0 denotes completely transparent.

Additional Inherited Members

18.14.1 Detailed Description

A button consists of two images, one for its normal state and one when it is pressed down.

See also

AbstractButton

18.14.2 Constructor & Destructor Documentation

18.14.2.1 Button ( ) [inline]

Default constructor.

18.14.2.2 ~Button ( ) [inline],[virtual]

Destructor.

18.14.3 Member Function Documentation

18.14.3.1 void draw (const Rect & *invalidatedArea*) const [virtual]

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Implements Drawable.

Reimplemented in ButtonWithLabel, and ButtonWithIcon.

18.14.3.2 uint8_t getAlpha ( ) const [inline]

Gets the current alpha value.

Returns

The current alpha value.

18.14.3.3 Bitmap getCurrentlyDisplayedBitmap ( ) const [inline]

Function to obtain the currently displayed bitmap, which depends on the button's pressed state.

Returns

The bitmap currently displayed.

18.14.3.4 Rect getSolidRect ( ) const [virtual]

Gets solid rectangle.

Returns

largest possible solid rect. Delegated to the largest solid rect of the button bitmap(s).

Implements Drawable.

Reimplemented in ButtonWithLabel.

18.14.3.5 uint16_t getType ( ) const [inline],[virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_BUTTON.

Reimplemented from AbstractButton.

Reimplemented in ButtonWithLabel, ButtonWithIcon, and ToggleButton.

18.14.3.6 void setAlpha (uint8_t *alpha*) [inline]

Sets the alpha value for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

18.14.3.7 void setBitmaps (const Bitmap & *bmpReleased*, const Bitmap & *bmpPressed*) [virtual]

Sets the bitmaps used by this button.

Parameters

bmpReleased	Bitmap to use when button is released.
bmpPressed	Bitmap to use when button is pressed.

Reimplemented in ToggleButton.

18.15 ButtonController Class Reference

Interface for sampling external key events.

```
#include <platform/driver/button/ButtonController.hpp>
```

Public Member Functions

- virtual ~ButtonController ()

  Destructor.
- virtual void init ()=0

  Initializes button controller.
- virtual bool sample (uint8_t &key)=0

Sample external key events.

- virtual void reset ()

  Resets button controller.

18.15.1 Detailed Description

Interface for sampling external key events.

18.15.2 Constructor & Destructor Documentation

18.15.2.1 ~ButtonController ( ) [inline],[virtual]

Destructor.

18.15.3 Member Function Documentation

18.15.3.1 void init ( ) [pure virtual]

Initializes button controller.

18.15.3.2 void reset ( ) [inline],[virtual]

Resets button controller. Does nothing in the default implementation.

18.15.3.3 bool sample (uint8_t & *key*) [pure virtual]

Sample external key events.

Parameters

in,out	*key*	Output parameter that will be set to the key value if a keypress was detected.

Returns

True if a keypress was detected and the "key" parameter is set to a value.

18.16 Buttons Class Reference

A buttons.

`#include <touchgfx/hal/Buttons.hpp>`

Static Public Member Functions

- static void init ()

  Perform configuration of IO pins.
- static unsigned int sample ()

  Sample button states.

18.16.1 Member Function Documentation

18.16.1.1 static void init ( ) [static]

Perform configuration of IO pins.

18.16.1.2 static unsigned int sample ( ) [static]

Sample button states.

Returns

the sampled state of the buttons.

18.17 ButtonWithIcon Class Reference

A Button specialization that also displays an icon on top of the button bitmap.

```
#include <touchgfx/widgets/ButtonWithIcon.hpp>
```

Public Member Functions

- virtual void setBitmaps (const Bitmap &newBackgroundReleased, const Bitmap &newBackgroundPressed, const Bitmap &newIconReleased, const Bitmap &newIconPressed)

  Sets the bitmaps used by this button.
- void setIconX (int16_t x)

  Sets the x coordinate of the icon bitmap.
- void setIconY (int16_t y)

  Sets the y coordinate of the icon bitmap.
- void setIconXY (int16_t x, int16_t y)

  Sets the x and y coordinates of the icon bitmap.
- Bitmap getCurrentlyDisplayedIcon () const

  Function to obtain the currently displayed icon.
- int16_t getIconX () const

  Gets the x coordinate of the icon bitmap.
- int16_t getIconY () const

  Gets the y coordinate of the icon bitmap.
- virtual void draw (const Rect &invalidatedArea) const

  Draws the given invalidated area.
- virtual uint16_t getType () const

  For GUI testing only. Returns type of this drawable.

Protected Attributes

- Bitmap iconReleased

  Icon to display when button is not pressed.
- Bitmap iconPressed

  Icon to display when button is pressed.
- int16_t iconX

  x coordinate offset for icon.
- int16_t iconY

  y coordinate offset for icon.

Additional Inherited Members

18.17.1 Detailed Description

A Button specialization that also displays an icon on top of the button bitmap.

See also

Button

18.17.2 Member Function Documentation

18.17.2.1 virtual void draw (const Rect & *invalidatedArea*) const [virtual]

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Reimplemented from Button.

18.17.2.2 Bitmap getCurrentlyDisplayedIcon ( ) const [inline]

Function to obtain the currently displayed icon, which depends on the button's pressed state.

Returns

The icon currently displayed.

18.17.2.3 int16_t getIconX ( ) const [inline]

Gets the x coordinate of the icon bitmap.

Returns

The x coordinate of the icon bitmap.

18.17.2.4 int16_t getIconY ( ) const [inline]

Gets the y coordinate of the icon bitmap.

Returns

The y coordinate of the icon bitmap.

18.17.2.5 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_BUTTONWITHICON.

Reimplemented from Button.

18.17.2.6 void setBitmaps (const Bitmap & *newBackgroundReleased,* const Bitmap & *newBackgroundPressed,* const Bitmap & *newIconReleased,* const Bitmap & *newIconPressed*) [virtual]

Sets the bitmaps used by this button.

Parameters

new↩ Background↩ Released	Bitmap to use when button is released.
new↩ Background↩ Pressed	Bitmap to use when button is pressed.
newIcon↩ Released	The bitmap for the icon in the released/unpressed button state.
newIconPressed	The bitmap for the icon in the pressed button state.

18.17.2.7 void setIconX (int16_t *x*) [inline]

Sets the x coordinate of the icon bitmap.

Note

Changing this does not automatically yield a redraw.
The value will be overwritten by calling.

Parameters

x	The new x value, relative to the background bitmap. A negative value is allowed.

18.17.2.8 void setIconXY (int16_t *x*, int16_t *y*) [inline]

Sets the x and y coordinates of the icon bitmap.

Note

Changing this does not automatically yield a redraw.

Parameters

x	The new x value, relative to the background bitmap. A negative value is allowed.
y	The new y value, relative to the background bitmap. A negative value is allowed.

18.17.2.9 void setIconY (int16_t *y*) [inline]

Sets the y coordinate of the icon bitmap.

Note

Changing this does not automatically yield a redraw.

Parameters

y	The new y value, relative to the background bitmap. A negative value is allowed.

18.18 ButtonWithLabel Class Reference

A Button specialization that also displays a text on top of the button bitmap.

```
#include <touchgfx/widgets/ButtonWithLabel.hpp>
```

Public Member Functions

- ButtonWithLabel ()

  Default constructor.
- void setLabelText (TypedText t)

  Sets the text to display on the button.
- TypedText getLabelText () const

  Gets the text used for the label.
- void setLabelColor (colortype col, bool performInvalidate=false)

  Sets label color.
- void setLabelColorPressed (colortype col, bool performInvalidate=false)

  Sets label color when the button is pressed.
- void setLabelRotation (TextRotation rotation)

  Sets the rotation of the text on the label.
- TextRotation getLabelRotation ()

  Gets the current rotation of the text on the label.
- void updateTextPosition ()

  Positions the label text as horizontally centered.
- virtual Rect getSolidRect () const

  Gets solid rectangle.
- virtual void draw (const Rect &area) const

  Draws the given invalidated area.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- TypedText typedText

  The TypedText used for the button label.
- colortype color

  The color used for the label when not pressed.
- colortype colorPressed

  The color used for the label when pressed.
- TextRotation rotation

  The rotation used for the label.
- uint8_t textHeightIncludingSpacing

  Total height of the label (text height + spacing).

Additional Inherited Members

18.18.1 Detailed Description

A Button specialization that also displays a text on top of the button bitmap.

See also

Button

18.18.2 Constructor & Destructor Documentation

18.18.2.1 ButtonWithLabel ( )

Default constructor.

18.18.3 Member Function Documentation

18.18.3.1 virtual void draw (const Rect & *invalidatedArea*) const [virtual]

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Reimplemented from Button.

18.18.3.2 TextRotation getLabelRotation ( ) [inline]

Gets the current rotation of the text on the label.

Returns

The current rotation of the text.

18.18.3.3 TypedText getLabelText ( ) const [inline]

Gets the text used for the label.

Returns

The text used for the label.

18.18.3.4 virtual Rect getSolidRect ( ) const [inline], [virtual]

Gets solid rectangle.

Returns

largest possible solid rect. Delegated to the largest solid rect of the button bitmap(s).

Reimplemented from Button.

18.18.3.5 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_BUTTONWITHLABEL.

Reimplemented from Button.

18.18.3.6 void setLabelColor (colortype *col*, bool *performInvalidate* = false) [inline]

Sets label color.

Parameters

col	The color with which the text label should be drawn.
perform↩ Invalidate	Optional parameter. If true, performs an instant invalidation of the button.

18.18.3.7 void setLabelColorPressed (colortype *col*, bool *performInvalidate* = `false`) `[inline]`

Sets label color when the button is pressed.

Parameters

col	The color with which the text label should be drawn when the button is pressed.
perform↩ Invalidate	Optional parameter. If true, performs an instant invalidation of the button.

18.18.3.8 void setLabelRotation (TextRotation *rotation*) `[inline]`

Sets the rotation of the text on the label. Please note that this will not rotate the bitmap of the label, only the text.

Parameters

rotation	the rotation of the text. Default is TEXT_ROTATE_0.

18.18.3.9 void setLabelText (TypedText *t*) `[inline]`

Sets the text to display on the button. Texts with wildcards are not supported.

Parameters

t	The text to display.

18.18.3.10 void updateTextPosition ( ) `[inline]`

If the text changes due to a language change you may need to reposition the label text to stay horizontally centered.

Note

The method does not invalidate the button. This must be done manually.

18.19 Bitmap::CacheTableEntry Struct Reference

Cache bookkeeping.

`#include <touchgfx/Bitmap.hpp>`

Public Attributes

- uint8_t * data

  Pointer to location of image data for this bitmap in the cache. 0 if bitmap not cached.

18.19.1 Detailed Description

Cache bookkeeping.

18.20 Callback< dest_type, T1, T2 > Struct Template Reference

A Callback is basically a wrapper of a pointer-to-member-function.

`#include <touchgfx/Callback.hpp>`

Public Member Functions

- Callback ()

  Default constructor.
- Callback (dest_type *pobject, void(dest_type::*pmemfun_2)(T1, T2))

  Initializes a Callback with an object and a pointer to the member function in that object to call.
- virtual void execute (T1 t1, T2 t2)

  Calls the member function.
- virtual bool isValid () const

  Function to check whether the Callback has been initialized with values.

18.20.1 Detailed Description

template<class dest_type, typename T1 = void, typename T2 = void>struct touchgfx::Callback< dest_type, T1, T2 >

A Callback is basically a wrapper of a pointer-to-member-function.

It is used for registering callbacks between widgets. For instance, a Button can be configured to call a member function when it is clicked.

The class is templated in order to provide the class type of the object in which the member function resides, and the argument types of the function to call.

The Callback class exists in three versions, for supporting member functions with 0, 1 or 2 arguments. The compiler will infer which type to use automatically.

Example button_example shows how to use callbacks.

Note

The member function to call must return void. The function can have zero, one or two arguments of any type.

Template Parameters

dest_type	The type of the class in which the member function resides.
T1	The type of the first argument in the member function, or void if none.
T2	The type of the second argument in the member function, or void if none.

18.20.2 Constructor & Destructor Documentation

18.20.2.1 Callback ( ) [inline]

Initializes an empty callback.

18.20.2.2 Callback (dest_type * *pobject*, void(dest_type::*)(T1, T2) *pmemfun_2*) [inline]

Initializes a Callback with an object and a pointer to the member function in that object to call.

Parameters

in	*pobject*	Pointer to the object on which the function should be called.
in	*pmemfun_2*	Address of member function. This is the version where function takes two arguments.

18.20.3 Member Function Documentation

18.20.3.1 void execute (T1 *t1*, T2 *t2*) [inline], [virtual]

Calls the member function. Do not call execute unless isValid() returns true (ie. a pointer to the object and the function has been set).

Parameters

t1	This value will be passed as the first argument in the function call.
t2	This value will be passed as the second argument in the function call.

Implements GenericCallback< T1, T2 >.

18.20.3.2 bool isValid ( ) const [inline], [virtual]

Function to check whether the Callback has been initialized with values.

Returns

true If the callback is valid (i.e. safe to call execute).

Implements GenericCallback< T1, T2 >.

18.21 Callback< dest_type, T1, void > Struct Template Reference

A Callback is basically a wrapper of a pointer-to-member-function.

```
#include <touchgfx/Callback.hpp>
```

Public Member Functions

- Callback ()

  Default constructor.
- Callback (dest_type *pobject, void(dest_type::*pmemfun_1)(T1))

  Initializes a Callback with an object and a pointer to the member function in that object to call.
- virtual void execute (T1 t1)

  Calls the member function.
- virtual bool isValid () const

  Query if this object is valid.

18.21.1 Detailed Description

template<class dest_type, typename T1>struct touchgfx::Callback< dest_type, T1, void >

A Callback is basically a wrapper of a pointer-to-member-function.

It is used for registering callbacks between widgets. For instance, a Button can be configured to call a member function when it is clicked.

The class is templated in order to provide the class type of the object in which the member function resides, and the argument types of the function to call.

The Callback class exists in three versions, for supporting member functions with 0, 1 or 2 arguments. The compiler will infer which type to use automatically.

Example button_example shows how to use callbacks.

Note

The member function to call must return void. The function can have zero, one or two arguments of any type.

Template Parameters

dest_type	The type of the class in which the member function resides.
T1	The type of the first argument in the member function, or void if none.

18.21.2 Constructor & Destructor Documentation

18.21.2.1 Callback ( ) [inline]

Initializes an empty callback.

18.21.2.2 Callback (dest_type * *pobject*, void(dest_type::*)(T1) *pmemfun_1*) [inline]

Initializes a Callback with an object and a pointer to the member function in that object to call.

Parameters

in	*pobject*	Pointer to the object on which the function should be called.
in	*pmemfun_1*	Address of member function. This is the version where function takes one argument.

18.21.3 Member Function Documentation

18.21.3.1 void execute (T1 *t1*) [inline], [virtual]

Calls the member function. Do not call execute unless isValid() returns true (ie. a pointer to the object and the function has been set).

Parameters

t1	This value will be passed as the first argument in the function call.

See also

isValid()

18.21.3.2 bool isValid ( ) const [inline], [virtual]

Query if this object is valid.

Returns

true if valid, false if not.

Implements GenericCallback< T1 >.

18.22 Callback< dest_type, void, void > Struct Template Reference

A Callback is basically a wrapper of a pointer-to-member-function.

```
#include <touchgfx/Callback.hpp>
```

Public Member Functions

- Callback ()

  Default constructor.
- Callback (dest_type *pobject, void(dest_type::*pmemfun_0)())

  Initializes a Callback with an object and a pointer to the member function in that object to call.
- virtual void execute ()

  Calls the member function.
- virtual bool isValid () const

  Function to check whether the Callback has been initialized with values.

18.22.1 Detailed Description

template<class dest_type>struct touchgfx::Callback< dest_type, void, void >

A Callback is basically a wrapper of a pointer-to-member-function.

It is used for registering callbacks between widgets. For instance, a Button can be configured to call a member function when it is clicked.

The class is templated in order to provide the class type of the object in which the member function resides, and the argument types of the function to call.

The Callback class exists in three versions, for supporting member functions with 0, 1 or 2 arguments. The compiler will infer which type to use automatically.

Example button_example shows how to use callbacks.

Note

The member function to call must return void. The function can have zero, one or two arguments of any type.

Template Parameters

dest_type	The type of the class in which the member function resides.

18.22.2 Constructor & Destructor Documentation

18.22.2.1 Callback ( ) [inline]

Initializes an empty callback.

18.22.2.2 Callback (dest_type * *pobject*, void(dest_type::*)() *pmemfun_0*) [inline]

Initializes a Callback with an object and a pointer to the member function in that object to call.

Parameters

in	*pobject*	Pointer to the object on which the function should be called.
in	*pmemfun_0*	Address of member function. This is the version where function takes zero arguments.

18.22.3 Member Function Documentation

18.22.3.1 void execute ( ) [inline], [virtual]

Calls the member function. Do not call execute unless isValid() returns true (ie. a pointer to the object and the function has been set).

18.22.3.2 bool isValid ( ) const [inline], [virtual]

Function to check whether the Callback has been initialized with values.

Returns

true If the callback is valid (i.e. safe to call execute).

Implements GenericCallback<>.

18.23 Keyboard::CallbackArea Struct Reference

Mapping from rectangle to a callback method to execute.

`#include <touchgfx/widgets/Keyboard.hpp>`

Public Attributes

- Rect keyArea

  The area occupied by a key.
- GenericCallback * callback

  The callback to execute, when the area is "pressed". The callback should be a Callback<YourClass> member in the class using the keyboard.
- BitmapId highlightBitmapId

  A bitmap to show when the area is "pressed".

18.24 Canvas Class Reference

Class for easy rendering using CanvasWidgetRenderer.

`#include <touchgfx/widgets/canvas/Canvas.hpp>`

Public Member Functions

- Canvas (const CanvasWidget *_widget, const Rect &invalidatedArea)

  Canvas Constructor.
- virtual ~Canvas ()

  Destructor.
- void moveTo (CWRUtil::Q5 x, CWRUtil::Q5 y)

Move the current pen position.

- void lineTo (CWRUtil::Q5 x, CWRUtil::Q5 y)

  Draw line from current pen position.

- template<class T >
  void moveTo (T x, T y)

  Move the current pen position.

- template<class T >
  void lineTo (T x, T y)

  Draw line from current pen position.

- bool render ()

  Render the drawn shape.

18.24.1 Detailed Description

The Canvas class will make implementation of a new CanvasWidget very easy. The few simple primitives allows moving a "pen" and drawing the outline of a shape which can then be rendered.

The Canvas class has been optimized to eliminate drawing unnecessary lines above and below the currently invalidated rectangle. This was chosen because CanvasWidgetRenderer works with horizontal scan lines, and eliminating unnecessary lines on the left and right does result in as good optimizations, and in some cases (as e.g. Circle) work against the desired (and expected) optimization.

18.24.2 Constructor & Destructor Documentation

18.24.2.1 Canvas (const CanvasWidget * *_widget,* const Rect & *invalidatedArea*)

Canvas Constructor. Locks the frame buffer and prepares for drawing only in the allowed area which has been invalidated. The color depth of the LCD is taken into account.

Parameters

_widget	a pointer to the CanvasWidget using this Canvas. Used for getting the canvas dimensions.
invalidatedArea	the are which should be updated.

18.24.2.2 ~Canvas ( ) [virtual]

Destructor. Takes care of unlocking the frame buffer.

18.24.3 Member Function Documentation

18.24.3.1 void lineTo (CWRUtil::Q5 *x,* CWRUtil::Q5 *y*)

Mark the line from the current (x, y) to the new (x, y) as part of the shape being drawn. As for moveTo, moveTo and lineTo commands completely outside the drawing are are discarded.

Parameters

x	The x coordinate for the pen position in Q5 format.
y	The y coordinate for the pen position in Q5 format.

See also

CWRUtil::Q5
moveTo

18.24.3.2 template< class T > void lineTo (T *x*, T *y*) [inline]

Mark the line from the current (x, y) to the new (x, y) as part of the shape being drawn. As for moveTo, moveTo and lineTo commands completely outside the drawing are are discarded.

Template Parameters

T	either int or float.

Parameters

x	The x coordinate for the pen position.
y	The y coordinate for the pen position.

18.24.3.3 void moveTo (CWRUtil::Q5 *x*, CWRUtil::Q5 *y*)

Move the current pen position to (x, y). If the pen is outside (above or below) the drawing area, nothing is done, but the coordinates are saved in case the next operation is lineTo a coordinate which is inside (or on the opposite side of) the drawing area.

Parameters

x	The x coordinate for the pen position in Q5 format.
y	The y coordinate for the pen position in Q5 format.

See also

CWRUtil::Q5
lineTo

18.24.3.4 template< class T > void moveTo (T *x*, T *y*) [inline]

Move the current pen position to (x, y). If the pen is outside (above or below) the drawing area, nothing is done, but the coordinates are saved in case the next operation is lineTo a coordinate which is inside (or on the opposite side of) the drawing area.

Template Parameters

T	Either int or float.

Parameters

x	The x coordinate for the pen position.
y	The y coordinate for the pen position.

18.24.3.5 bool render ( )

Render the graphical shape drawn (using moveTo() and lineTo()) using the widgets Painter. The shape is automatically closed, i.e. a lineTo() is automatically inserted connecting the current pen position with the initial pen position given in the previous moveTo() command.

Returns

true if the widget was rendered, false if insufficient memory was available to render the widget.

18.25 CanvasWidget Class Reference

Class for drawing complex polygons on the LCD using CanvasWidgetRenderer.

```
#include <touchgfx/widgets/canvas/CanvasWidget.hpp>
```

Public Member Functions

- CanvasWidget ()

  Constructor.
- virtual ~CanvasWidget ()

  Destructor.
- void setPainter (AbstractPainter &painter)

  Sets the painter for the CanvasWidget.
- AbstractPainter & getPainter () const

  Gets the current painter for the CanvasWidget.
- virtual void setAlpha (uint8_t alpha)

  Sets the alpha channel for the CanvasWidget.
- virtual uint8_t getAlpha () const

  Gets the current alpha value.
- virtual void draw (const Rect &invalidatedArea) const

  Draws the given invalidated area.
- virtual void invalidate () const

  Invalidates the area covered by this CanvasWidget.
- virtual Rect getMinimalRect () const

  Gets minimal rectangle containing the shape drawn by this widget.
- virtual Rect getSolidRect () const

  Gets the largest solid (non-transparent) rectangle.
- virtual bool drawCanvasWidget (const Rect &invalidatedArea) const =0

  Draw canvas widget for the given invalidated area.

Additional Inherited Members

18.25.1 Detailed Description

Class for drawing complex polygons on the LCD using CanvasWidgetRenderer.

See also

Widget

18.25.2 Constructor & Destructor Documentation

18.25.2.1 CanvasWidget ( )

Constructor.

18.25.2.2 ~CanvasWidget ( ) [virtual]

Destructor. Declared virtual for sub classing purposes.

18.25.3 Member Function Documentation

18.25.3.1 void draw (const Rect & *invalidatedArea*) const [virtual]

Draws the given invalidated area. If the underlying CanvasWidgetRenderer fail to render the widget (due to memory limitations), the invalidated area is cut into smaller slices which are then drawn separately. If drawing a single raster line fails, that line is skipped (left blank/transparent) and drawing continues on the next raster line.

If drawing has failed at least once, the number of successfully drawn lines is remembered for the next time. If a future draw would need to draw more lines, the area is automatically divided into smaller areas to prevent drawing the canvas widget in vain.

Parameters

invalidatedArea	The invalidated area.

See also

drawCanvasWidget()

Implements Drawable.

18.25.3.2 bool drawCanvasWidget (const Rect & *invalidatedArea*) const [pure virtual]

Draw canvas widget for the given invalidated area. Similar to draw(), but might be invoked with several times with smaller areas to due to memory constraints from the underlying CanvasWidgetRenderer.

Parameters

invalidatedArea	The invalidated area.

Returns

true the widget was drawn, false if not.

See also

draw()

Implemented in Line, AbstractShape, and Circle.

18.25.3.3 uint8_t getAlpha ( ) const [inline], [virtual]

Gets the current alpha value.

Returns

The current alpha value.

18.25.3.4 Rect getMinimalRect ( ) const [virtual]

Gets minimal rectangle containing the shape drawn by this widget. Default implementation returns the size of the entire widget, but this function should be overwritten in subclasses and return the minimal rectangle containing the shape. See classes such as Circle for example implementations.

Returns

The minimal rectangle containing the shape drawn by this widget.

Reimplemented in Line, AbstractShape, and Circle.

18.25.3.5 AbstractPainter & getPainter ( ) const

Gets the current painter for the CanvasWidget.

Returns

The painter.

See also

AbstractPainter

18.25.3.6 Rect getSolidRect ( ) const [virtual]

Gets the largest solid (non-transparent) rectangle. Since canvas widgets typically do not have a solid rect, it is recommended to return an empty rectangle.

Returns

The largest solid (non-transparent) rectangle.

Implements Drawable.

18.25.3.7 void invalidate ( ) const [virtual]

Invalidates the area covered by this CanvasWidget. Since many widgets are a lot smaller than the actual size of the canvas widget, each widget must be able to tell the smallest rectangle completely containing the shape drawn by the widget. For example a circle arc is typically much smaller than the widget containing the circle.

See also

getMinimalRect()

Reimplemented from Drawable.

18.25.3.8 void setAlpha (uint8_t *alpha*) [inline], [virtual]

Sets the alpha channel for the CanvasWidget.

Parameters

alpha	The alpha value. 255 = completely solid.

18.25.3.9 void setPainter (AbstractPainter & *painter*)

Sets the painter for the CanvasWidget.

Note

The area containing the CanvasWidget is not invalidated.

Parameters

in	*painter*	The painter for the CanvasWidget.

See also

AbstractPainter

18.26 CanvasWidgetRenderer Class Reference

Class for supporting drawing of figures.

```
#include <touchgfx/canvas_widget_renderer/CanvasWidgetRenderer.hpp>
```

Static Public Member Functions

- static void setupBuffer (uint8_t *buffer, unsigned bufsize)

  Setup the buffers used by CanvasWidget.
- static bool setScanlineWidth (unsigned width)

  Sets scanline width.
- static bool hasBuffer ()

  Query if CanvasWidgetRenderer has been initialized with a buffer.
- static unsigned getScanlineWidth ()

  The width of a scanline.
- static void * getScanlineCovers ()

  Gets pointer to memory used for covers in Scanline.
- static void * getScanlineStartIndices ()

  Gets pointer to memory used for indices in Scanline.
- static void * getScanlineCounts ()

  Gets pointer to memory used for counts in Scanline.
- static Cell * getOutlineBuffer ()

  Gets pointer to memory used for Cell objects in Outline.
- static unsigned int getOutlineBufferSize ()

  Gets size of memory area used for Cell objects in Outline.
- static void numCellsUsed (unsigned used)

  Called by the destructor in Outline to help keep track of the memory requirements of CanvasWidgets.
- static void numCellsMissing (unsigned missing)

  Called by the destructor in Outline to help keep track of the memory requirements of CanvasWidgets.
- static unsigned getUsedBufferSize ()

  Calculate how much memory has been required by CanvasWidgets.
- static unsigned getMissingBufferSize ()

  Calculate how much memory was required by CanvasWidgets, but was unavailable.

Additional Inherited Members

18.26.1 Detailed Description

Class for supporting drawing of figures. This class holds the memory which is used by the underlying algorithms. CanvasWidget will not allocate memory dynamically, but will use memory from the buffer kept in CanvasWidget↩ Renderer. When using the TouchGFX simulator, it is also possible to get a report on the actual amount of memory used for the drawings to help adjusting the buffer size.

See also

Widget

18.26.2 Member Function Documentation

18.26.2.1 static unsigned getMissingBufferSize ( ) [inline], [static]

Calculate how much memory was required by CanvasWidgets, but was unavailable. If the value returned is greater than 0 it means the This can be used to fine tune the size of the buffer passed to CanvasWidgetRenderer upon initialization.

Returns

The number of bytes required.

18.26.2.2 static Cell * getOutlineBuffer ( ) [inline], [static]

Gets pointer to memory used for Cell objects in Outline.

Returns

Pointer to memory used internally by Outline.

18.26.2.3 static unsigned int getOutlineBufferSize ( ) [inline], [static]

Gets size of memory area used for Cell objects in Outline.

Returns

Size of memory area used internally by Outline.

18.26.2.4 static void * getScanlineCounts ( ) [inline], [static]

Gets pointer to memory used for counts in Scanline.

Returns

Pointer to memory used internally by Scanline.

18.26.2.5 static void * getScanlineCovers ( ) [inline], [static]

Gets pointer to memory used for covers in Scanline.

Returns

Pointer to memory used internally by Scanline.

18.26.2.6 static void * getScanlineStartIndices ( ) [inline], [static]

Gets pointer to memory used for indices in Scanline.

Returns

Pointer to memory used internally by Scanline.

18.26.2.7 static unsigned getScanlineWidth ( ) [inline],[static]

The width of a scanline. This is the same as the width of the invalidated area. Used to optimize the memory layout of the buffer.

Returns

Scanline width (HAL::FRAME_BUFFER_WIDTH).

18.26.2.8 static unsigned getUsedBufferSize ( ) [inline],[static]

Calculate how much memory has been required by CanvasWidgets. This can be used to fine tune the size of the buffer passed to CanvasWidgetRenderer upon initialization.

Returns

The number of bytes required.

18.26.2.9 static bool hasBuffer ( ) [inline],[static]

Query if CanvasWidgetRenderer has been initialized with a buffer.

Returns

True if a buffer has been setup.

18.26.2.10 static void numCellsMissing (unsigned *missing*) [inline],[static]

Called by the destructor in Outline to help keep track of the memory requirements of CanvasWidgets.

Parameters

missing	Number of Cell objects required, but not available, to Outline.

18.26.2.11 static void numCellsUsed (unsigned *used*) [inline],[static]

Called by the destructor in Outline to help keep track of the memory requirements of CanvasWidgets.

Parameters

used	Number of Cell objects used from the dedicated buffer.

18.26.2.12 static bool setScanlineWidth (unsigned *width*) [inline],[static]

Sets scanline width. Setting the scanline width will initialize the buffers for scanline and outline. If the width set is too large to fit the scanline buffers in the allocated memory buffer, false will be returned and all buffer pointers will be cleared.

Parameters

width	The width of the scanline on screen.

Returns

true if it succeeds, false if it fails.

18.26.2.13 static void setupBuffer (uint8_t * *buffer*, unsigned *bufsize*) [inline], [static]

Setup the buffers used by CanvasWidget.

Parameters

in	*buffer*	Buffer reserved for CanvasWidget.
	bufsize	The size of the buffer.

18.27 Cell Struct Reference

A pixel cell.

`#include <touchgfx/canvas_widget_renderer/Cell.hpp>`

Public Member Functions

- void set (int _x, int _y, int _cover, int _area)

  Sets all the Cell parameters.
- void setCoord (int _x, int _y)

  Sets the coordinate of the Cell.
- void setCover (int _cover, int _area)

  Sets the cover of area cell.
- void addCover (int _cover, int _area)

  Adds a cover to a Cell.
- int packedCoord () const

  Packed coordinates of the Cell.

Public Attributes

- int16_t x

  The x coordinate.
- int16_t y

  The y coordinate.
- int16_t cover

  The cover (see http://projects.tuxee.net/cl-vectors/section-the-cl-aa-algorithm *for further information).*
- int16_t area

  The area (see http://projects.tuxee.net/cl-vectors/section-the-cl-aa-algorithm *for further information).*

18.27.1 Detailed Description

A pixel cell. There are no constructors defined and it was done intentionally in order to avoid extra overhead when allocating an array of cells.

18.27.2 Member Function Documentation

18.27.2.1 void addCover (int _cover, int _area) [inline]

Adds a cover to a Cell.

Parameters

_cover	The cover to add to the Cell.
_area	The area to add to the Cell.

18.27.2.2 int packedCoord ( ) const [inline]

Packed coordinates of the Cell. By packing the x coordinate and y coordinate into one int, it is possible to sort Cells using a single comparison.

Returns

The packed coordinates with y in the high part and x in the low part.

18.27.2.3 void set (int _x, int _y, int _cover, int _area) [inline]

Sets all the Cell parameters.

Parameters

_x	The x coordinate.
_y	The y coordinate.
_cover	The cover.
_area	The area.

18.27.2.4 void setCoord (int _x, int _y) [inline]

Sets the coordinate of the Cell.

Parameters

_x	The Cell's x coordinate.
_y	The Cell's y coordinate.

18.27.2.5 void setCover (int _cover, int _area) [inline]

Sets the cover of area cell.

Parameters

_cover	The cover.
_area	The area.

18.28 Circle Class Reference

Simple widget capable of drawing a circle.

```
#include <touchgfx/widgets/canvas/Circle.hpp>
```

Public Member Functions

- Circle ()

  Constructs a new Circle.
- template<class T >
  void setCircle (T x, T y, T r)

  Sets the center and radius of the Circle.
- template<class T >
  void setCenter (T x, T y)

  Sets the center of the Circle.
- template<class T >
  void getCenter (T &x, T &y)

  Gets the center coordinates of the Circle.
- template<class T >
  void setRadius (T r)

  Sets the radius of the Circle.
- template<class T >
  void getRadius (T &r)

  Gets the radius of the Circle.
- void setArc (int16_t startAngle, int16_t endAngle)

  Sets the start and end angles in degrees of the Circle arc.
- void getArc (int16_t &startAngle, int16_t &endAngle)

  Gets the start and end angles in degrees for the circle arc.
- int16_t getArcStart () const

  Gets the start angle in degrees for the arc.
- int16_t getArcEnd () const

  Gets the end angle in degrees for the arc.
- void updateArcStart (int16_t startAngle)

  Updates the start angle in degrees for this Circle arc.
- void updateArcEnd (int16_t endAngle)

  Updates the end angle in degrees for this Circle arc.
- template<class T >
  void setLineWidth (T width)

  Sets the width for this Circle.
- void setPrecision (int precision)

  Sets a precision of the Circle drawing function.
- void setCapPrecision (int precision)

  Sets a precision of the ends of the Circle arc.
- virtual bool drawCanvasWidget (const Rect &invalidatedArea) const

  Draws the Circle.
- virtual Rect getMinimalRect () const

  Gets minimal rectangle for the current shape of the circle.
- Rect getMinimalRect (int16_t arcStart, int16_t arcEnd) const

  Gets minimal rectangle containing a given circle arc.

Additional Inherited Members

18.28.1 Detailed Description

Simple widget capable of drawing a circle. By tweaking the parameters of the circle, several parameters of the circle can be changed. Center, radius, line width, line cap and partial circle arc. This opens for creation of fascinating graphics.

See also

CanvasWidget

18.28.2 Constructor & Destructor Documentation

18.28.2.1 Circle ( )

Constructs a new Circle.

18.28.3 Member Function Documentation

18.28.3.1 bool drawCanvasWidget (const Rect & *invalidatedArea*) const [virtual]

Draws the Circle. This class supports partial drawing, so only the area described by the rectangle will be drawn.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Returns

true if it succeeds, false if it fails.

Implements CanvasWidget.

18.28.3.2 void getArc (int16_t & *startAngle*, int16_t & *endAngle*)

Gets the start and end angles in degrees for the circle arc.

Parameters

out	*startAngle*	The start.
out	*endAngle*	The end.

See also

setArc

18.28.3.3 int16_t getArcEnd ( ) const

Returns

The finishing angle for the arc.

See also

getArc
setArc

18.28.3.4 int16_t getArcStart ( ) const

Gets the start angle in degrees for the arc.

Returns

The starting angle for the arc.

See also

getArc
setArc

18.28.3.5 template< class T > void getCenter (T & *x*, T & *y*) [inline]

Gets the center coordinates of the Circle.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*x*	The x coordinate of the center.
out	*y*	The y coordinate of the center.

18.28.3.6 Rect getMinimalRect ( ) const [virtual]

Gets minimal rectangle for the current shape of the circle.

Returns

The minimal rectangle.

Reimplemented from CanvasWidget.

18.28.3.7 Rect getMinimalRect (int16_t *arcStart*, int16_t *arcEnd*) const

Gets minimal rectangle containing a given circle arc.

Parameters

arcStart	The arc start.
arcEnd	The arc end.

Returns

The minimal rectangle.

18.28.3.8 template< class T > void getRadius (T & *r*) [inline]

Gets the radius of the Circle.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*r*	The radius.

18.28.3.9 void setArc (int16_t *startAngle*, int16_t *endAngle*)

Sets the start and end angles in degrees of the Circle arc. 0 degrees is straight up (12 o'clock) and 90 degress is to the left (3 o'clock). Any positive or negative degrees can be used to specify the part of the Circle to draw.

Note

The area containing the Circle is not invalidated.

Parameters

startAngle	The start degrees.
endAngle	The end degrees.

See also

getArc
updateArcStart
updateArcEnd

18.28.3.10 void setCapPrecision (int *precision*)

Sets a precision of the ends of the Circle arc. The precision is given in degrees where 180 is the default which results in a square ended arc (aka "butt cap"). 90 will draw "an arrow head" and smaller values gives a round cap. Larger values of precision results in faster rendering of the circle.

Note

The circle is not invalidated.

Parameters

precision	The new cap precision.

18.28.3.11 template< class T > void setCenter (T *x*, T *y*) [inline]

Sets the center of the Circle.

Note

The area containing the Circle is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of center.
y	The y coordinate of center.

18.28.3.12 template< class T > void setCircle (T *x*, T *y*, T *r*) [inline]

Sets the center and radius of the Circle.

Note

The area containing the Circle is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of center.
y	The y coordinate of center.
r	The radius.

18.28.3.13 template< class T > void setLineWidth (T *width*) [inline]

Sets the width for this Circle. If the line width is set to zero, the circle will be filled.

Note

The area containing the Circle is not invalidated.
if the new line with is smaller than the old width, the circle should be invalidated before updating the width to ensure that the old circle is completely erased.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

width	The width of the line measured in pixels.

18.28.3.14 void setPrecision (int *precision*)

Sets a precision of the Circle drawing function. The precision is given in degrees, five being a sensible value. Higher values results in less nice circles but faster rendering. The number passed as precision is the number of degrees used as step counter when drawing smaller line fragments around the circumference of the circle.

Note

The circle is not invalidated.

Parameters

precision	The precision measured in degrees.

18.28.3.15 template< class T > void setRadius (T *r*) [inline]

Sets the radius of the Circle.

Note

The area containing the Circle is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

r	The radius.

18.28.3.16 void updateArcEnd (int16_t *endAngle*)

Updates the end angle in degrees for this Circle arc.

Note

The area containing the updated Circle arc is invalidated.

Parameters

endAngle	The end angle in degrees.

See also

setArc
updateArcStart

18.28.3.17 void updateArcStart (int16_t *startAngle*)

Updates the start angle in degrees for this Circle arc.

Note

The area containing the updated Circle arc is invalidated.

Parameters

startAngle	The start angle in degrees.

See also

setArc
updateArcEnd

18.29 ClickEvent Class Reference

A click event.

```
#include <touchgfx/events/ClickEvent.hpp>
```

Public Types

- enum ClickEventType { PRESSED, RELEASED, CANCEL }

  The click event types.

Public Member Functions

- ClickEvent (ClickEventType type, int16_t x, int16_t y, int16_t force=0)

  Constructor.
- virtual ~ClickEvent ()

  Destructor.
- int16_t getX () const

  Gets the x coordinate of this event.
- int16_t getY () const

  Gets the y coordinate of this event.
- void setX (int16_t x)

  Sets the x coordinate of this event.
- void setY (int16_t y)

  Sets the y coordinate of this event.
- void setType (ClickEventType type)

  Sets the click type of this event.
- ClickEventType getType () const

  Gets the click type of this event.
- int16_t getForce () const

  Gets the force of the click.
- virtual Event::EventType getEventType ()

  Gets event type.

18.29.1 Detailed Description

A click event. The semantics of this event is slightly depending on hardware platform. ClickEvents are generated by the HAL layer.

See also

Event

18.29.2 Member Enumeration Documentation

18.29.2.1 enum enum ClickEventType

Enumerator

PRESSED An enum constant representing the pressed option.

RELEASED An enum constant representing the released option.

CANCEL An enum constant representing the cancel option.

18.29.3 Constructor & Destructor Documentation

18.29.3.1 ClickEvent (ClickEventType *type*, int16_t *x*, int16_t *y*, int16_t *force* = 0) [inline]

Constructor.

Parameters

type	The type of the click event.
x	The x coordinate of the click event.
y	The y coordinate of the click event.
force	The force of the click. On touch displays this usually means how hard the user pressed on the display. On the windows platform, this will always be zero.

18.29.3.2 ~ClickEvent ( ) [inline],[virtual]

Destructor.

18.29.4 Member Function Documentation

18.29.4.1 Event::EventType getEventType ( ) [inline],[virtual]

Gets event type.

Returns

The type of this event.

Implements Event.

18.29.4.2 int16_t getForce ( ) const [inline]

Gets the force of the click. On touch displays this usually means how hard the user pressed on the display. On the windows platform, this will always be zero.

Returns

The force of the click.

18.29.4.3 ClickEventType getType ( ) const [inline]

Gets the click type of this event.

Returns

The click type of this event.

18.29.4.4 int16_t getX ( ) const [inline]

Gets the x coordinate of this event.

Returns

The x coordinate of this event.

18.29.4.5 int16_t getY ( ) const [inline]

Gets the y coordinate of this event.

Returns

The y coordinate of this event.

18.29.4.6 void setType (ClickEventType *type*) [inline]

Sets the click type of this event.

Parameters

type	The type to set.

18.29.4.7 void setX (int16_t *x*) [inline]

Sets the x coordinate of this event.

Parameters

x	The x coordinate of this event.

18.29.4.8 void setY (int16_t *y*) [inline]

Sets the y coordinate of this event.

Parameters

y	The y coordinate of this event.

18.30 ClickListener< T > Class Template Reference

Mix-in class that extends a class with a click action event.

`#include <touchgfx/mixins/ClickListener.hpp>`

Public Member Functions

- ClickListener ()

  Default constructor.
- virtual ~ClickListener ()

  Destructor.
- virtual void handleClickEvent (const ClickEvent &event)

  Ensures that the clickEvent is propagated to the super class.
- void setClickAction (GenericCallback< const T &, const ClickEvent & > &callback)

  Associates an action to be performed when the class T is clicked.

Protected Attributes

- GenericCallback< const T &, const ClickEvent & > * clickAction

  The callback to be executed when T is clicked.

18.30.1 Detailed Description

template<class T>class touchgfx::ClickListener< T >

Mix-in class that extends a class with a click action event that is called when the class receives a click event.

Template Parameters

T	specifies the type to extend with the ClickListener behavior.

18.30.2 Constructor & Destructor Documentation

18.30.2.1 ClickListener ( ) [inline]

Default constructor.

18.30.3 Member Function Documentation

18.30.3.1 void handleClickEvent (const ClickEvent & *event*) [inline], [virtual]

Ensures that the clickEvent is propagated to the super class T and to the clickAction listener.

Parameters

event	Information about the click.

See also

Drawable::handleClickEvent()

18.30.3.2 void setClickAction (GenericCallback< const T &, const ClickEvent & > & *callback*) [inline]

Associates an action to be performed when the class T is clicked.

Parameters

callback	The callback to be executed. The callback will be given a reference to T.

See also

GenericCallback

18.31 Color Class Reference

Contains functionality for color conversion.

```
#include <touchgfx/Color.hpp>
```

Static Public Member Functions

- static colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue)

  Generates a color representation to be used on the LCD, based on 24 bit RGB values. Depending on your chosen color bit depth, the color will be interpreted internally as either a 16 bit or 24 bit color value.
- static uint8_t getRedColor (colortype color)

  Gets the red color part of a color.
- static uint8_t getGreenColor (colortype color)

  Gets the green color part of a color.
- static uint8_t getBlueColor (colortype color)

  Gets the blue color part of a color.

18.31.1 Detailed Description

Contains functionality for color conversion.

18.31.2 Member Function Documentation

18.31.2.1 static inline uint8_t getBlueColor (colortype *color*) [inline],[static]

Gets the blue color part of a color.

Parameters

color	The 16 bit color value.

Returns

The blue part of the color.

18.31.2.2 static colortype getColorFrom24BitRGB (uint8_t *red*, uint8_t *green*, uint8_t *blue*) [static]

Generates a color representation to be used on the LCD, based on 24 bit RGB values. Depending on your chosen color bit depth, the color will be interpreted internally as either a 16 bit or 24 bit color value. This function can be safely used regardless of whether your application is configured for 16 or 24 bit colors.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color, encoded in a 16- or 24-bit representation depending on LCD color depth.

18.31.2.3 static inline uint8_t getGreenColor (colortype *color*) [inline],[static]

Gets the green color part of a color.

Parameters

color	The 16 bit color value.

Returns

The green part of the color.

18.31.2.4 static inline uint8_t getRedColor (colortype *color*) [inline],[static]

Gets the red color part of a color.

Parameters

color	The color value.

Returns

The red part of the color.

18.32 colortype Struct Reference

Type for representing a color.

```
#include <touchgfx/hal/Types.hpp>
```

Public Member Functions

- colortype ()

  Default constructor.
- colortype (uint32_t col)

  Constructor.
- uint32_t getColor32 () const

  Gets color 32 as a 32bit value suitable for passing to Color::getRedColor(), Color::getGreenColor() and Color::get↩ BlueColor() which will handle both 16 bit colortypes and 24 bit colortypes.
- operator uint16_t () const

  Cast that converts the given colortype to an uint16_t.

Public Attributes

- uint32_t color

  The color.

18.32.1 Detailed Description

This type can contain a color value. Note that in order to maintain backwards compatibility, casting this type to an integral value will yield a 16-bit value. To extract a 24/32-bit color from this type, use the getColor32 function.

18.32.2 Constructor & Destructor Documentation

18.32.2.1 colortype ( ) [inline]

Default constructor. Creates a black (0) color.

18.32.2.2 colortype (uint32_t *col*) [inline]

Contructor which creates a colortype with the given color. Use Color::getColorFrom24BitRGB() to create a color that will work on both 16 bit LCD and 24 bit LCD.

Parameters

col	The color.

See also

Color::getColorFrom24BitRGB()

18.32.3 Member Function Documentation

18.32.3.1 uint32_t getColor32 ( ) const [inline]

Returns

The color 32.

See also

Color::getRedColor()
Color::getGreenColor()
Color::getBlueColor()

18.32.3.2 operator uint16_t ( ) const [inline]

Cast that converts the given colortype to an uint16_t. Provided only for backward compatibility. Not recommended to use.

Returns

The result of the operation.

18.33 ConstFont Class Reference

A ConstFont is a Font implementation that has its contents defined at compile-time and usually placed in read-only memory.

```
#include <touchgfx/ConstFont.hpp>
```

Public Member Functions

- ConstFont (const GlyphNode *list, uint16_t size, uint16_t height, uint8_t pixBelowBase, uint8_t bitsPerPixel, uint8_t maxLeft, uint8_t maxRight)

  Creates a font instance.
- virtual const GlyphNode * getGlyph (Unicode::UnicodeChar unicode, const uint8_t *&pixelData, uint8_t &bitsPerPixel) const

  Gets the glyph data associated with the specified unicode.
- virtual const uint8_t * getPixelData (const GlyphNode *glyph) const =0

  Gets the pixel date associated with this glyph.
- virtual int8_t getKerning (Unicode::UnicodeChar prevChar, const GlyphNode *glyph) const =0

  Gets the kerning distance between two characters.

Protected Member Functions

- const GlyphNode * find (Unicode::UnicodeChar unicode) const

  Finds the glyph data associated with the specified unicode.

Protected Attributes

- const GlyphNode * glyphList

  The list of glyphs.
- uint16_t listSize

  The size of the list of glyphs.

Additional Inherited Members

18.33.1 Detailed Description

A ConstFont is a Font implementation that has its contents defined at compile-time and usually placed in read-only memory.

Note

Pure virtual class. Create an application- specific implementation of getPixelData.

See also

Font

18.33.2 Constructor & Destructor Documentation

18.33.2.1 ConstFont (const GlyphNode * *list*, uint16_t *size*, uint16_t *height*, uint8_t *pixBelowBase*, uint8_t *bitsPerPixel*, uint8_t *maxLeft*, uint8_t *maxRight*)

Creates a font instance.

Parameters

list	The array of glyphs known to this font.
size	The number of glyphs in list.
height	The height in pixels of the highest character in this font.
pixBelowBase	The maximum number of pixels that can be drawn below the baseline in this font.
bitsPerPixel	The number of bits per pixel in this font.
maxLeft	The maximum a character extends to the left.
maxRight	The maximum a character extends to the right.

18.33.3 Member Function Documentation

18.33.3.1 const GlyphNode * find (Unicode::UnicodeChar *unicode*) const [protected]

Finds the glyph data associated with the specified unicode.

Parameters

unicode	The character to look up.

Returns

A pointer to the glyph node or null if the glyph was not found.

18.33.3.2 const GlyphNode * getGlyph (Unicode::UnicodeChar *unicode*, const uint8_t *& *pixelData*, uint8_t & *bitsPerPixel*) const [virtual]

Gets the glyph data associated with the specified unicode. An implementation of Font::getGlyph. Searches the glyph list for the specified font.

Complexity O(log n)

Parameters

	unicode	The character to look up.
	pixelData	Pointer to the pixel data for the glyph if the glyph is found. This is set by this method.
out	*bitsPerPixel*	Reference where to place the number of bits per pixel.

Returns

A pointer to the glyph node or null if the glyph was not found.

Implements Font.

18.33.3.3 int8_t getKerning (Unicode::UnicodeChar *prevChar*, const GlyphNode * *glyph*) const [pure virtual]

Gets the kerning distance between two characters.

Parameters

prevChar	The unicode value of the previous character.
glyph	the glyph object for the current character.

Returns

The kerning distance between prevChar and glyph char.

Reimplemented from Font.

Implemented in InternalFlashFont.

18.33.3.4 const uint8_t * getPixelData (const GlyphNode * *glyph*) const [pure virtual]

Gets the pixel date associated with this glyph.

Parameters

glyph	The glyph to get the pixels data from.

Returns

Pointer to the pixel data of this glyph.

Implemented in InternalFlashFont.

18.34 Container Class Reference

A Container is a Drawable that can have child nodes.

```
#include <touchgfx/containers/Container.hpp>
```

Public Member Functions

- Container ()

  Default constructor.
- virtual ~Container ()

Destructor.

- virtual void add (Drawable &d)

  Adds a Drawable instance as child to this Container.
- virtual void remove (Drawable &d)

  Removes a Drawable instance from the list of children.
- virtual void removeAll ()

  Removes all children by resetting their parent and sibling pointers.
- virtual void unlink ()

  Removes all children by unlinking the first child.
- virtual bool contains (const Drawable &d)

  Query if this object contains the given drawable.
- virtual void insert (Drawable *previous, Drawable &d)

  Inserts a Drawable instance after the specified child node.
- virtual void getLastChild (int16_t x, int16_t y, Drawable **last)

  Gets the last child of this container.
- virtual void draw (const Rect &invalidatedArea) const

  Draws the children of this container.
- virtual Rect getSolidRect () const

  Gets solid rectangle of a Container.
- virtual void forEachChild (GenericCallback< Drawable & > *function)

  Calls the specified function for each child in the container.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Member Functions

- virtual Rect getContainedArea () const

  Gets a rectangle describing the total area covered by the children of this container.
- virtual void moveChildrenRelative (int16_t deltaX, int16_t deltaY)

  Calls moveRelative on all children.
- virtual void setupDrawChain (const Rect &invalidatedArea, Drawable **nextPreviousElement)

  For TouchGFX internal use only.

Protected Attributes

- Drawable * firstChild

  Pointer to the first child of this container. Subsequent children can be found through firstChild->nextSibling.

Friends

- class **Screen**

Additional Inherited Members

18.34.1 Detailed Description

A Container is a Drawable that can have child nodes. The z-order of children is determined by the order in which Drawables are added to the container - the Drawable added last will be front-most on the screen.

This class overrides a few functions in Drawable in order to traverse child nodes.

Note that containers act as view ports - that is, only the parts of children that intersect with the geometry of the container will be visible (e.g. setting a container's width to 0 will render all children invisible).

See also

Drawable

18.34.2 Constructor & Destructor Documentation

18.34.2.1 Container ( ) [inline]

Default constructor.

18.34.2.2 ~Container ( ) [inline], [virtual]

Destructor.

18.34.3 Member Function Documentation

18.34.3.1 void add (Drawable & *d*) [virtual]

Adds a Drawable instance as child to this Container.

Note

Never add a drawable more than once (will loop forever)!

Parameters

in	*d*	The Drawable to add.

Reimplemented in ScrollableContainer, and ListLayout.

18.34.3.2 bool contains (const Drawable & *d*) [virtual]

Query if this object contains the given drawable.

Parameters

d	The Drawable to look for.

Returns

True if the specified Drawable instance is direct child of this container, false otherwise.

18.34.3.3 void draw (const Rect & *invalidatedArea*) const [virtual]

Draws the children of this container. Only used when JSMOC is disabled.

Parameters

invalidatedArea	The area to draw.

Implements Drawable.

Reimplemented in Keyboard.

18.34.3.4 void forEachChild (GenericCallback< Drawable & > * *function*) [virtual]

Calls the specified function for each child in the container. Function to call must have the following prototype: void T::func(Drawable&)

Parameters

in	*function*	The function to be executed for each child.

See also

ListLayout::insert for a usage example.

18.34.3.5 Rect getContainedArea ( ) const [protected], [virtual]

Gets a rectangle describing the total area covered by the children of this container.

Returns

Rectangle covering all children.

Reimplemented in ScrollableContainer.

18.34.3.6 void getLastChild (int16_t *x*, int16_t *y*, Drawable ** *last*) [virtual]

Gets the last (=highest Z-order) child of this container that is enabled, visible and intersects with the specified point. Recursive function.

Parameters

	x	The x coordinate of the intersection.
	y	The y coordinate of the intersection.
out	*last*	out parameter in which the result is placed.

See also

Drawable::getLastChild

Implements Drawable.

Reimplemented in ScrollableContainer.

18.34.3.7 Rect getSolidRect ( ) const [virtual]

Gets solid rectangle of a Container. JSMOC does not operate directly on containers.

Returns

An empty rectangle per default.

Implements Drawable.

18.34.3.8 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_CONTAINER.

Reimplemented from Drawable.

Reimplemented in ZoomAnimationImage, ScrollableContainer, Keyboard, Slider, and ListLayout.

18.34.3.9 void insert (Drawable * *previous*, Drawable & *d*) [virtual]

Inserts a Drawable instance after the specified child node.

Note

As with add, do not add the same drawable twice.

Parameters

in	*previous*	The Drawable to look foor.
in	*d*	The Drawable to insert.

Reimplemented in ListLayout.

18.34.3.10 void moveChildrenRelative (int16_t *deltaX*, int16_t *deltaY*) [protected], [virtual]

Calls moveRelative on all children.

Parameters

deltaX	Horizontal displacement.
deltaY	Vertical displacement.

Reimplemented in ScrollableContainer.

18.34.3.11 void remove (Drawable & *d*) [virtual]

Removes a Drawable instance from the list of children.

Note

This is safe to call even if d is not a child (in which case nothing happens).

Parameters

in	*d*	The Drawable to remove.

Reimplemented in ListLayout.

18.34.3.12 void removeAll ( ) [virtual]

Removes all children by resetting their parent and sibling pointers.

Reimplemented in ListLayout.

18.34.3.13 virtual void setupDrawChain (const Rect & *invalidatedArea*, Drawable ** *nextPreviousElement*) [protected], [virtual]

Configure linked list for draw chain.

Note

For TouchGFX internal use only.

Parameters

	invalidatedArea	Include drawables that intersect with this area only.
in,out	*nextPrevious↩ Element*	Modifiable element in linked list.

Reimplemented from Drawable.

Reimplemented in Keyboard.

18.34.3.14 void unlink ( ) [virtual]

Removes all children by unlinking the first child. The parent and sibling pointers of the children are not reset.

18.35 CortexMMCUInstrumentation Class Reference

Interface for instrumenting Cortex-M processors to measure MCU load via measured CPU cycles.

`#include <platform/core/arm/cortex-m/CortexMMCUInstrumentation.hpp>`

Public Member Functions

- virtual void init ()

  Initialization.
- virtual unsigned int getElapsedUS (unsigned int start, unsigned int now, unsigned int clockfrequency)

  Gets elapsed microseconds basedon clockfrequency.
- virtual unsigned int getCPUCycles ()

  Gets CPU cycles from register.
- virtual void setMCUActive (bool active)

  Register if MCU is active by measuring cpu cycles.

Additional Inherited Members

18.35.1 Detailed Description

Interface for instrumenting Cortex-M processors to measure MCU load via measured CPU cycles.

See also

MCUInstrumentation

18.35.2 Member Function Documentation

18.35.2.1 unsigned int getCPUCycles (void) [virtual]

Gets CPU cycles from register.

Returns

CPU cycles.

Implements MCUInstrumentation.

18.35.2.2 unsigned int getElapsedUS (unsigned int *start,* unsigned int *now,* unsigned int *clockfrequency*) [virtual]

Gets elapsed microseconds basedon clockfrequency.

Parameters

start	Start time.
now	Current time.
clockfrequency	Clock frequency of the system.

Returns

Elapsed microseconds start and now.

Implements MCUInstrumentation.

18.35.2.3 void init ( ) [virtual]

Initialization.

Implements MCUInstrumentation.

18.35.2.4 void setMCUActive (bool *active*) [virtual]

Register if MCU is active by measuring cpu cycles. If user wishes to track MCU load, this method should be called whenever the OS Idle task is scheduled in or out. This method makes calls to a concrete implementation of GPIO functionality and a concrete implementation of cpu cycles.

Parameters

active	If true, MCU is registered as being active, inactive otherwise.

Reimplemented from MCUInstrumentation.

18.36 CWRUtil Struct Reference

Helper classes and functions for CanvasWidget.

```
#include <touchgfx/widgets/canvas/CWRUtil.hpp>
```

Classes

- class Q10

  Defines a number with 10 bits reserved for fraction.
- class Q15

  Defines a number with 15 bits reserved for fraction.
- class Q5

  Defines a number with 5 bits reserved for fraction.

Static Public Member Functions

- template<class T >
  static Q5 toQ5 (T value)

  Convert an integer to a fixed point number.
- template<class T >
  static Q10 toQ10 (T value)

  Convert an integer to a fixed point number.
- static Q15 sine (int i)

Find the value of sin(i) with 15 bits precision.

- static Q15 cosine (int i)

  Find the value of cos(i) with 15 bits precision.
- static int8_t arcsine (Q10 q10)

  Gets the arcsine of the given fraction (given as Q10).
- template<class T >
  static int angle (T x, T y)

  Find angle of a coordinate.
- template<class T >
  static int angle (T x, T y, T &d)

  Find angle of a coordinate.
- static int angle (Q5 x, Q5 y)

  Find angle of a coordinate.
- static int angle (Q5 x, Q5 y, Q5 &d)

  Find the angle of the coordinate (x, y).
- static Q5 sqrtQ10 (Q10 value)

  Find the square root of the given value.

18.36.1 Detailed Description

Helper classes and functions for CanvasWidget. A handful of utility functions can be found here. These include helper functions for converting between float, int and Q5/Q10/Q15 format. There are also functions for calculating sin() and cos() in integers with a high number of bits reserved for fraction. Having sin() and cos() pre- calculated in this way allows very fast drawing of circles without the need for floating point arithmetic.

Using Q5, numbers from -1024.00000 to +1024.96875 with a precision of 1/32 = 0.03125 can be represented.

See also

http://en.wikipedia.org/wiki/Q_%28number_format%29
Widget

18.36.2 Member Function Documentation

18.36.2.1 template< class T > static int angle (T *x*, T *y*) [inline],[static]

Template Parameters

T	Generic type parameter (int or float).

Parameters

x	The x coordinate.
y	The y coordinate.

Returns

The angle of the coordinate.

18.36.2.2 template< class T > static int angle (T *x*, T *y*, T & *d*) [inline],[static]

Template Parameters

T	Generic type parameter (int or float).

Parameters

	x	The x coordinate.
	y	The y coordinate.
out	*d*	The distance from (0,0) to (x,y).

Returns

The angle of the coordinate.

18.36.2.3 static int angle (Q5 *x*, Q5 *y*) [inline], [static]

Parameters

x	The x coordinate.
y	The y coordinate.

Returns

The angle of the coordinate.

18.36.2.4 static int angle (Q5 *x*, Q5 *y*, Q5 & *d*) [inline], [static]

Parameters

	x	The x coordinate.
	y	The y coordinate.
out	*d*	The distance from (0,0) to (x,y).

Returns

The angle.

18.36.2.5 static int8_t arcsine (Q10 *q10*) [inline], [static]

Gets the arcsine of the given fraction (given as Q10). The function is most precise for angles 0-45. To limit memory requirements, values above sqrt(1/2) is calculated as 90-arcsine(sqrt(1-q10^2)). Internally.

Parameters

q10	The 10.

Returns

An int8_t.

18.36.2.6 static Q15 cosine (int *i*) [inline], [static]

Find the value of cos(i) with 15 bits precision using the fact that cos(i)=sin(90- i).

Parameters

i	the angle in degrees. The angle follows the angles of the clock, 0 being straight up and 90 being 3 o'clock.

Returns

the value of cos(i) with 15 bits precision on the fractional part.

See also

sine()

18.36.2.7 static Q15 sine (int *i*) [inline],[static]

Find the value of sin(i) with 15 bits precision. The returned value can be converted to a floating point number and divided by (1<<15) to get the rounded value of of sin(i). By using this function, a complete circle can be drawn without the need for using floating point math.

Parameters

i	the angle in degrees. The angle follows the angles of the clock, 0 being straight up and 90 being 3 o'clock.

Returns

the value of sin(i) with 15 bits precision on the fractional part.

18.36.2.8 static Q5 sqrtQ10 (Q10 *value*) [inline],[static]

Parameters

value	The value to find the square root of.

Returns

The square root of the given value.

18.36.2.9 template< class T > inline static Q10 toQ10 (T *value*) [inline],[static]

Convert an integer to a fixed point number. This is done by shifting the integer value 10 places to the left, or multiplying the floating point value by (1 < < 510.

Template Parameters

T	Should be either int or float.

Parameters

value	the integer to convert.

Returns

the converted integer.

18.36.2.10 template< class T > inline static Q5 toQ5 (T *value*) [inline],[static]

Convert an integer to a fixed point number. This is done by shifting the integer value 5 places to the left, or multiplying the floating point value by (1 << 5)

Template Parameters

T	Should be either int or float.

Parameters

value	the integer to convert.

Returns

the converted integer.

18.37 DisplayTransformation Class Reference

Defines transformations from display space to frame buffer space.

`#include <touchgfx/transforms/DisplayTransformation.hpp>`

Static Public Member Functions

- static void transformDisplayToFrameBuffer (int16_t &x, int16_t &y)

  Transform x,y from display to frame buffer coordinates.
- static void transformDisplayToFrameBuffer (float &x, float &y)

  Transform x,y from display to frame buffer coordinates.
- static void transformFrameBufferToDisplay (int16_t &x, int16_t &y)

  Transform x,y from frame buffer to display coordinates.
- static void transformDisplayToFrameBuffer (int16_t &x, int16_t &y, const Rect &in)

  Transform x,y from coordinates relative to the in rect to frame buffer coordinates.
- static void transformDisplayToFrameBuffer (float &x, float &y, const Rect &in)

  Transform x,y from coordinates relative to the in rect to frame buffer coordinates.
- static void transformDisplayToFrameBuffer (Rect &r)

  Transform rectangle from display to frame buffer coordinates.
- static void transformDisplayToFrameBuffer (Rect &r, const Rect &in)

  Transform rectangle r from coordinates relative to the in rect to frame buffer coordinates.

18.37.1 Detailed Description

Defines transformations from display space to frame buffer space. The display might be (considered) in portrait mode from 0,0 to 272,480, while the actual frame buffer is from 0,0 to 480,272. This class handles the transformations.

18.37.2 Member Function Documentation

18.37.2.1 static void transformDisplayToFrameBuffer (int16_t & *x*, int16_t & *y*) [static]

Transform x,y from display to frame buffer coordinates.

Parameters

in,out	*x*	the x part to translate.

in,out	*y*	the y part to translate.

18.37.2.2 static void transformDisplayToFrameBuffer (float & *x*, float & *y*) [static]

Transform x,y from display to frame buffer coordinates.

Parameters

in,out	*x*	the x part to translate.
in,out	*y*	the y part to translate.

18.37.2.3 static void transformDisplayToFrameBuffer (int16_t & *x*, int16_t & *y*, const Rect & *in*) [static]

Transform x,y from coordinates relative to the in rect to frame buffer coordinates.

Parameters

in,out	*x*	the x part to translate.
in,out	*y*	the y part to translate.
	in	the rectangle defining the coordinate space.

18.37.2.4 static void transformDisplayToFrameBuffer (float & *x*, float & *y*, const Rect & *in*) [static]

Transform x,y from coordinates relative to the in rect to frame buffer coordinates.

Parameters

in,out	*x*	the x part to translate.
in,out	*y*	the y part to translate.
	in	the rectangle defining the coordinate space.

18.37.2.5 static void transformDisplayToFrameBuffer (Rect & *r*) [static]

Transform rectangle from display to frame buffer coordinates.

Parameters

in,out	*r*	the rectangle to translate.

18.37.2.6 static void transformDisplayToFrameBuffer (Rect & *r*, const Rect & *in*) [static]

Transform rectangle r from coordinates relative to the in rect to frame buffer coordinates.

Parameters

in,out	*r*	the rectangle to translate.
	in	the rectangle defining the coordinate space.

18.37.2.7 static void transformFrameBufferToDisplay (int16_t & *x*, int16_t & *y*) [static]

Transform x,y from frame buffer to display coordinates.

Parameters

in,out	*x*	the x part to translate.
in,out	*y*	the y part to translate.

18.38 DMA_Interface Class Reference

DMA_Interface provides basic functionality and structure for processing "blit" operations using DMA.

`#include <touchgfx/hal/DMA.hpp>`

Public Member Functions

- virtual BlitOperations getBlitCaps ()=0

  Gets the blit capabilities of this DMA.
- virtual void addToQueue (const BlitOp &op)

  Inserts a BlitOp for processing.
- virtual void flush ()

  This function blocks until all DMA transfers in the queue have been completed.
- virtual void initialize ()

  Perform initialization.
- bool isDMARunning ()

  Query if the DMA is running.
- void setAllowed (bool allowed)

  Sets whether or not a DMA operation is allowed to begin.
- bool getAllowed () const

  Gets whether a DMA operation is allowed to begin.
- virtual void start ()

  Signals that DMA transfers can start.
- virtual void signalDMAInterrupt ()=0

  This function is called automatically by the framework when a DMA interrupt has been received.
- uint8_t isDmaQueueEmpty ()

  Query if the DMA queue is empty.
- uint8_t isDmaQueueFull ()

  Query if the DMA queue is full.
- virtual ~DMA_Interface ()

  Destructor.

Protected Member Functions

- DMA_Interface (DMA_Queue &dmaQueue)

  Constructs a DMA Interface object.
- virtual void execute ()

  Performs a queued blit-op.
- virtual void executeCompleted ()

  To be called when blit-op has been performed.
- virtual void seedExecution ()

  Called when elements are added to the DMA-queue.
- virtual void setupDataCopy (const BlitOp &blitOp)=0

  Configures blit-op hardware for a 2D copy as specified by blitOp.

- virtual void setupDataFill (const BlitOp &blitOp)=0

  Configures blit-op hardware for a 2D fill as specified by blitOp.
- virtual void enableAlpha (uint8_t alpha)

  Configures blit-op hardware for alpha-blending.
- virtual void disableAlpha ()

  Configures blit-op hardware for solid operation (no alpha-blending).
- virtual void enableCopyWithTransparentPixels (uint8_t alpha)

  Configures blit-op hardware for alpha-blending while simultaneously skipping transparent pixels.
- virtual void waitForFrameBufferSemaphore ()

  Waits until frame buffer semaphore is available.

Protected Attributes

- DMA_Queue & queue

  Reference to the DMA queue.
- bool isRunning

  true if a DMA transfer is currently ongoing.
- volatile bool isAllowed

  true if DMA transfers are currently allowed.

18.38.1 Detailed Description

DMA_Interface provides basic functionality and structure for processing "blit" operations using DMA.

18.38.2 Constructor & Destructor Documentation

18.38.2.1 ~DMA_Interface ( ) [inline],[virtual]

Destructor.

18.38.2.2 DMA_Interface (DMA_Queue & *dmaQueue*) [inline],[protected]

Constructs a DMA Interface object.

Parameters

in	*dmaQueue*	Reference to the queue of DMA operations.

18.38.3 Member Function Documentation

18.38.3.1 void addToQueue (const BlitOp & *op*) [virtual]

Inserts a BlitOp for processing. This also potentially starts the DMA controller, if not already running.

Parameters

op	The operation to add.

18.38.3.2 void disableAlpha ( ) [protected],[virtual]

Configures blit-op hardware for solid operation (no alpha-blending)

18.38.3.3 void enableAlpha (uint8_t *alpha*) [protected], [virtual]

Configures blit-op hardware for alpha-blending.

Parameters

alpha	The alpha-blending value to apply.

18.38.3.4 void enableCopyWithTransparentPixels (uint8_t *alpha*) [protected],[virtual]

Configures blit-op hardware for alpha-blending while simultaneously skipping transparent pixels.

Parameters

alpha	The alpha-blending value to apply.

18.38.3.5 void execute ( ) [protected],[virtual]

Performs a queued blit-op.

18.38.3.6 void executeCompleted ( ) [protected],[virtual]

To be called when blit-op has been performed.

18.38.3.7 void flush ( ) [inline],[virtual]

This function blocks until all DMA transfers in the queue have been completed.

Reimplemented in NoDMA.

18.38.3.8 bool getAllowed ( ) const [inline]

Gets whether a DMA operation is allowed to begin. Used in single-buffering to avoid changing the frame buffer while display is being updated.

Returns

true if DMA is allowed to start, false if not.

18.38.3.9 BlitOperations getBlitCaps ( ) [pure virtual]

Gets the blit capabilities of this DMA.

Returns

The blit operations supported by this DMA implementation.

Implemented in NoDMA.

18.38.3.10 void initialize ( ) [inline],[virtual]

Perform initialization. Does nothing in this base class.

18.38.3.11 uint8_t isDmaQueueEmpty ( )

Query if the DMA queue is empty.

Returns

1 if DMA queue is empty, else 0.

18.38.3.12 uint8_t isDmaQueueFull ( )

Query if the DMA queue is full.

Returns

1 if DMA queue is full, else 0.

18.38.3.13 bool isDMARunning ( ) [inline]

Query if the DMA is running.

Returns

true if a DMA operation is currently in progress.

18.38.3.14 void seedExecution ( ) [protected],[virtual]

Called when elements are added to the DMA-queue.

Note

The frame buffer must be locked before this method returns if the DMA-queue is non- empty.

18.38.3.15 void setAllowed (bool *allowed*) [inline]

Sets whether or not a DMA operation is allowed to begin. Used in single-buffering to avoid changing the frame buffer while display is being updated.

Parameters

allowed	true if DMA transfers are allowed.

18.38.3.16 void setupDataCopy (const BlitOp & *blitOp*) [protected],[pure virtual]

Configures blit-op hardware for a 2D copy as specified by blitOp.

Parameters

blitOp	The operation to execute.

Implemented in NoDMA.

18.38.3.17 void setupDataFill (const BlitOp & *blitOp*) [protected],[pure virtual]

Configures blit-op hardware for a 2D fill as specified by blitOp.

Parameters

blitOp	The operation to execute.

Implemented in NoDMA.

18.38.3.18 void signalDMAInterrupt ( ) [pure virtual]

This function is called automatically by the framework when a DMA interrupt has been received.

Implemented in NoDMA.

18.38.3.19 void start ( ) [virtual]

Signals that DMA transfers can start. If any elements are in the queue, start it.

18.38.3.20 void waitForFrameBufferSemaphore ( ) [protected],[virtual]

Waits until frame buffer semaphore is available (i.e. neither DMA or application is accessing the frame buffer).

18.39 DMA_Queue Class Reference

This class provides an interface for a FIFO (circular) list used by DMA_Interface and descendants for storing BlitOp's.

```
#include <touchgfx/hal/DMA.hpp>
```

Public Member Functions

- virtual bool isEmpty ()=0

  Query if this object is empty.
- virtual bool isFull ()=0

  Query if this object is full.
- virtual void pushCopyOf (const BlitOp &op)=0

  Adds the specified blitop to the queue.
- virtual ~DMA_Queue ()

  Destructor.

Protected Member Functions

- DMA_Queue ()

  Default constructor.
- virtual void pop ()=0

  Pops an element from the queue.
- virtual const BlitOp * first ()=0

  Gets the first element in the queue.

Friends

- class **DMA_Interface**

18.39.1 Detailed Description

This class provides an interface for a FIFO (circular) list used by DMA_Interface and descendants for storing BlitOp's.

18.39.2 Constructor & Destructor Documentation

18.39.2.1 ~DMA_Queue ( ) [inline],[virtual]

Destructor.

18.39.2.2 DMA_Queue ( ) [inline],[protected]

Default constructor.

18.39.3 Member Function Documentation

18.39.3.1 const BlitOp * first ( ) [protected],[pure virtual]

Gets the first element in the queue.

Returns

The first element in the queue.

Implemented in LockFreeDMA_Queue.

18.39.3.2 bool isEmpty ( ) [pure virtual]

Query if this object is empty.

Returns

true if the queue is empty.

Implemented in LockFreeDMA_Queue.

18.39.3.3 bool isFull ( ) [pure virtual]

Query if this object is full.

Returns

true if the queue is full.

Implemented in LockFreeDMA_Queue.

18.39.3.4 void pop ( ) [protected],[pure virtual]

Pops an element from the queue.

Implemented in LockFreeDMA_Queue.

18.39.3.5 void pushCopyOf (const BlitOp & *op*) [pure virtual]

Adds the specified blitop to the queue.

Parameters

op	The blitop to add.

Implemented in LockFreeDMA_Queue.

18.40 DragEvent Class Reference

A drag event.

`#include <touchgfx/events/DragEvent.hpp>`

Public Types

- enum DragEventType { **DRAGGED** }

  The drag event types.

Public Member Functions

- DragEvent (DragEventType type, int16_t oldX, int16_t oldY, int16_t newX, int16_t newY)

  Constructor.
- virtual ~DragEvent ()

  Destructor.
- int16_t getOldX () const

  Gets the x coordinate where the drag operation was started (dragged from).
- int16_t getOldY () const

  Gets the y coordinate where the drag operation was started (dragged from).
- int16_t getNewX () const

  Gets the new x coordinate (dragged to).
- int16_t getNewY () const

  Gets the new x coordinate (dragged to).
- DragEventType getType () const

  Gets the type of this drag event.
- int16_t getDeltaX () const

  Gets the distance in x coordinates (how long was the drag).
- int16_t getDeltaY () const

  Gets the distance in y coordinates (how long was the drag).
- virtual Event::EventType getEventType ()

  Gets event type.

18.40.1 Detailed Description

A drag event. The only drag event currently supported is DRAGGED, which will be issued every time the input system detects a drag.

See also

Event

18.40.2 Constructor & Destructor Documentation

18.40.2.1 DragEvent (DragEventType *type,* int16_t *oldX,* int16_t *oldY,* int16_t *newX,* int16_t *newY*) [inline]

Constructor. Create a drag event of the specified type with the specified coordinates.

Parameters

type	The type of the drag event.
oldX	The x coordinate of the drag start position (dragged from)
oldY	The y coordinate of the drag start position (dragged from)
newX	The x coordinate of the new position (dragged to)
newY	The y coordinate of the new position (dragged to)

18.40.2.2 ~DragEvent ( ) [inline],[virtual]

Destructor.

18.40.3 Member Function Documentation

18.40.3.1 int16_t getDeltaX ( ) const [inline]

Gets the distance in x coordinates (how long was the drag).

Returns

The distance of this drag event.

18.40.3.2 int16_t getDeltaY ( ) const [inline]

Gets the distance in y coordinates (how long was the drag).

Returns

The distance of this drag event.

18.40.3.3 Event::EventType getEventType ( ) [inline],[virtual]

Gets event type.

Returns

The type of this event.

Implements Event.

18.40.3.4 int16_t getNewX ( ) const [inline]

Gets the new x coordinate (dragged to).

Returns

The new x coordinate (dragged to).

18.40.3.5 int16_t getNewY ( ) const [inline]

Gets the new x coordinate (dragged to).

Returns

The new y coordinate (dragged to).

18.40.3.6 int16_t getOldX ( ) const [inline]

Gets the x coordinate where the drag operation was started (dragged from).

Returns

The x coordinate where the drag operation was started (dragged from).

18.40.3.7 int16_t getOldY ( ) const [inline]

Gets the y coordinate where the drag operation was started (dragged from).

Returns

The y coordinate where the drag operation was started (dragged from).

18.40.3.8 DragEventType getType ( ) const [inline]

Gets the type of this drag event.

Returns

The type of this drag event.

18.41 Draggable< T > Class Template Reference

Mix-in class that extends a class to become draggable.

`#include <touchgfx/mixins/Draggable.hpp>`

Public Member Functions

- Draggable ()
  Default constructor.
- virtual ~Draggable ()
  Destructor.
- virtual void handleDragEvent (const DragEvent &evt)
  Called when dragging the draggable object.

18.41.1 Detailed Description

template<class T>class touchgfx::Draggable< T >

Mix-in class that extends a class to become draggable.

Template Parameters

T	specifies the type to extend with the draggable behavior.

18.41.2 Constructor & Destructor Documentation

18.41.2.1 Draggable ( ) [inline]

Default constructor.

18.41.2.2 ~Draggable ( ) [inline],[virtual]

Destructor.

18.41.3 Member Function Documentation

18.41.3.1 void handleDragEvent (const DragEvent & *evt*) [inline],[virtual]

Called when dragging the draggable object. The object is moved according to the drag event.

Parameters

evt	The drag event.

Reimplemented in Snapper< T >.

18.42 Drawable Class Reference

The Drawable class is an abstract definition of something that can be drawn.

```
#include <touchgfx/Drawable.hpp>
```

Public Types

- enum DrawableType {
  TYPE_DRAWABLE, **TYPE_WIDGET**, **TYPE_ABSTRACTBUTTON**, **TYPE_ANIMATEDIMAGE**, **TYPE_BOX**, **TYPE_BUTTON**, **TYPE_BUTTONWITHICON**, **TYPE_BUTTONWITHLABEL**, **TYPE_IMAGE**, **TYPE_KEYBOARD**, **TYPE_SCALABLEIMAGE**, **TYPE_SNAPSHOTWIDGET**, **TYPE_TEXTAREA**, **TYPE_TEXTAREAWITHONEWILDCARD**, **TYPE_TEXTAREAWITHTWOWILDCARDS**, **TYPE_TOGGLEBUTTON**, **TYPE_TOUCHAREA**, **TYPE_CONTAINER**, **TYPE_LISTLAYOUT**, **TYPE_SCROLLABLECONTAINER**, **TYPE_ZOOMANIMATIONIMAGE**, **TYPE_RADIOBUTTON**, **TYPE_TEXTUREMAPPER**, **TYPE_SLIDER**, **TYPE_CUSTOMTYPESBEGIN** }

  Enum defining Drawable types.

Public Member Functions

- Drawable ()

  Default constructor.
- virtual ~Drawable ()

  Destructor.
- virtual void draw (const Rect &invalidatedArea) const =0

  Pure virtual function for drawing this drawable.
- virtual Rect getSolidRect () const =0

  Pure virtual function for obtaining the largest possible rectangle that is guaranteed to be solid (non-transparent).
- virtual void invalidateRect (Rect &invalidatedArea) const

  Request that a subregion of this drawable is redrawn.
- virtual void invalidate () const

  Tell the framework that this entire drawable needs to be redrawn.
- Drawable * getNextSibling ()

  Gets the next sibling.
- virtual Rect getSolidRectAbsolute ()

  Helper function for obtaining the largest solid rect.
- virtual void getLastChild (int16_t x, int16_t y, Drawable **last)=0

Function for obtaining the the last child of this drawable that intersects with the specified point.

- virtual void getVisibleRect (Rect &rect) const

  JSMOC Function for obtaining the visible part of this drawable.
- const Rect & getRect () const

  Gets the rectangle this Drawable covers.
- Rect getAbsoluteRect () const

  Helper function for obtaining the rectangle this Drawable covers.
- virtual void translateRectToAbsolute (Rect &r) const

  Helper function for converting a specified subregion of this Drawable to absolute coordinates.
- virtual void setPosition (int16_t x, int16_t y, int16_t width, int16_t height)

  Sets the size and position of this Drawable.
- int16_t getX () const

  Gets the x coordinate of this drawable.
- int16_t getY () const

  Gets the y coordinate of this drawable.
- int16_t getWidth () const

  Gets the width of this drawable.
- int16_t getHeight () const

  Gets the height of this drawable.
- virtual void setX (int16_t x)

  Sets the x coordinate of this drawable.
- virtual void setY (int16_t y)

  Sets the y coordinate of this drawable.
- virtual void setXY (int16_t x, int16_t y)

  Sets the x and y coordinates of this drawable.
- virtual void setWidth (int16_t width)

  Sets the width of this drawable.
- virtual void setHeight (int16_t height)

  Sets the height of this drawable.
- virtual void childGeometryChanged ()

  This function can be called on parent nodes to signal that the size of one or more of its children has changed.
- virtual void handleClickEvent (const ClickEvent &evt)

  Defines the event handler interface for ClickEvents.
- virtual void handleGestureEvent (const GestureEvent &evt)

  Defines the event handler interface for GestureEvents.
- virtual void handleDragEvent (const DragEvent &evt)

  Defines the event handler interface for DragEvents.
- virtual void handleTickEvent ()

  Called periodically by the framework if the Drawable instance has subscribed to timer ticks.
- void setVisible (bool vis)

  Controls whether this Drawable should be visible.
- void setTouchable (bool touch)

  Controls whether this Drawable receives touch events or not.
- bool isVisible () const

  Gets whether this Drawable is visible.
- bool isTouchable () const

  Gets whether this Drawable receives touch events or not.
- Drawable * getParent ()

  Returns the parent node.
- virtual void moveRelative (int16_t x, int16_t y)

  Moves the drawable.

- virtual void moveTo (int16_t x, int16_t y)

  Moves the drawable.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Member Functions

- void resetDrawChainCache ()

  For TouchGFX internal use only.
- Rect & getCachedVisibleRect ()

  For TouchGFX internal use only.
- int16_t getCachedAbsX ()

  For TouchGFX internal use only.
- int16_t getCachedAbsY ()

  For TouchGFX internal use only.
- virtual void setupDrawChain (const Rect &invalidatedArea, Drawable **nextPreviousElement)

  For TouchGFX internal use only.

Protected Attributes

- Rect rect

  The coordinates of this drawable, relative to its parent.
- Rect cachedVisibleRect

  Cached representation of currently visible area. For TouchGFX internal use.
- Drawable * parent

  Pointer to this drawable's parent.
- Drawable * nextSibling

  Pointer to the next drawable. Maintained by containers.
- Drawable * nextDrawChainElement

  Next in draw chain. For TouchGFX internal use.
- int16_t cachedAbsX

  Cached value of absolute X-coord. For TouchGFX internal use.
- int16_t cachedAbsY

  Cached value of absolute Y-coord. For TouchGFX internal use.
- bool touchable

  True if this drawable should receive touch events.
- bool visible

  True if this drawable should be drawn.

Static Protected Attributes

- static const int16_t UNCACHED_INDICATOR = -1

  Constant representing uncached value. For TouchGFX internal use.

Friends

- class **Container**
- class **Screen**

18.42.1 Detailed Description

The Drawable class is an abstract definition of something that can be drawn. In the composite design pattern, the Drawable is the component interface. Drawables can be added to a screen as a tree structure through the leaf node class Widget and the Container class. A Drawable contains a pointer to its next sibling and a pointer to its parent node. These are maintained by the Container to which the Drawable is added.

The Drawable interface contains two pure virtual functions which must be implemented by widgets, namely draw() and getSolidRect(). In addition it contains general functionality for receiving events and navigating the tree structure.

The coordinates of a Drawable are always relative to its parent node.

See also

Widget
Container

18.42.2 Member Enumeration Documentation

18.42.2.1 enum enum DrawableType

Enum defining Drawable types. To be used by automated GUI testing to determine class type of a Drawable object.

18.42.3 Constructor & Destructor Documentation

18.42.3.1 Drawable ( ) [inline]

Default constructor.

18.42.3.2 ~Drawable ( ) [inline],[virtual]

Destructor.

18.42.4 Member Function Documentation

18.42.4.1 void childGeometryChanged ( ) [inline],[virtual]

This function can be called on parent nodes to signal that the size of one or more of its children has changed. Currently only used in ScrollableContainer to redraw scrollbars when the size of the scrolling contents changes.

Reimplemented in ScrollableContainer.

18.42.4.2 void draw (const Rect & *invalidatedArea*) const [pure virtual]

Pure virtual function for drawing this drawable. It is a requirement that the draw implementation does not draw outside the region specified by invalidatedArea.

Parameters

invalidatedArea	The subregion of this drawable that needs to be redrawn, expressed in coordinates relative to its parent (e.g. for a complete redraw, invalidatedArea will be (0, 0, width, height).

Implemented in Keyboard, TextArea, TextAreaWithTwoWildcards, ButtonWithLabel, ScalableImage, Container, ButtonWithIcon, CanvasWidget, Box, TextureMapper, TextAreaWithOneWildcard, Image, RadioButton, Button, SnapshotWidget, PixelDataWidget, and TouchArea.

18.42.4.3 Rect getAbsoluteRect ( ) const

Helper function for obtaining the rectangle this Drawable covers, expressed in absolute coordinates.

Returns

The rectangle this Drawable covers expressed in absolute coordinates.

18.42.4.4 int16_t getCachedAbsX ( ) [inline],[protected]

Obtain cached version of absolute X-coord.

Note

For TouchGFX internal use only.

Returns

The absolute x coordinate for this drawable. Only calculated once.

18.42.4.5 int16_t getCachedAbsY ( ) [inline],[protected]

Obtain cached version of absolute Y-coord.

Note

For TouchGFX internal use only.

Returns

The absolute y coordinate for this drawable. Only calculated once.

18.42.4.6 Rect & getCachedVisibleRect ( ) [inline],[protected]

Obtain cached version of visible rect.

Note

For TouchGFX internal use only.

Returns

The Visible rect for this drawable. Only calculated once.

18.42.4.7 int16_t getHeight ( ) const [inline]

Gets the height of this drawable.

Returns

The height of this drawable.

18.42.4.8 void getLastChild (int16_t *x*, int16_t *y*, Drawable ** *last*) [pure virtual]

Function for obtaining the the last child of this drawable that intersects with the specified point. Used in input event handling for obtaining the appropriate drawable that should receive the event. Note that input events must be delegated to the last drawable of the tree (meaning highest z-order / front-most drawable).

Parameters

	x	The point of intersection expressed in coordinates relative to the parent.
	y	The point of intersection expressed in coordinates relative to the parent.
out	*last*	Result will be placed here.

Implemented in Container, ScrollableContainer, and Widget.

18.42.4.9 Drawable * getNextSibling ( ) [inline]

Gets the next sibling.

Returns

Returns the next sibling or NULL if there is none.

18.42.4.10 Drawable * getParent ( ) [inline]

Returns the parent node. For the root container, the return value is 0.

Returns

The parent node. For the root container, the return value is 0.

18.42.4.11 const Rect & getRect ( ) const [inline]

Gets the rectangle this Drawable covers, in coordinates relative to its parent.

Returns

The rectangle this Drawable covers expressed in coordinates relative to its parent.

18.42.4.12 Rect getSolidRect ( ) const [pure virtual]

Pure virtual function for obtaining the largest possible rectangle that is guaranteed to be solid (non-transparent). Used by JSMOC to prune the draw graph.

Note

The rectangle returned must be relative to (0, 0), meaning that to indicate a completely solid widget, Rect(0, 0, getWidth(), getHeight()) must be returned.

Returns

The solid rect.

Implemented in ScalableImage, CanvasWidget, Container, ButtonWithLabel, TextureMapper, Image, RadioButton, TouchArea, Button, SnapshotWidget, PixelDataWidget, Box, and TextArea.

18.42.4.13 Rect getSolidRectAbsolute ( ) [virtual]

Helper function for obtaining the largest solid rect (as implemented by getSolidRect()) expressed in absolute coordinates. Will recursively traverse to the root of the tree.

Returns

Largest solid rect (as implemented by getSolidRect()) expressed in absolute coordinates.

18.42.4.14 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Call this virtual function to determine the class type of this Drawable object. Can be used in automated GUI testing. Otherwise this function is unused.

Note

If creating custom drawables that need to be manipulated by a test framework, override this function and return a custom value higher than or equal to TYPE_CUSTOMTYPESBEGIN.

Returns

An integer describing the class type of this object, corresponding to the DrawableType enum for built-in Drawables.

Reimplemented in TextureMapper, ZoomAnimationImage, TextArea, TextAreaWithTwoWildcards, ScrollableContainer, Keyboard, Slider, RadioButton, AnimatedImage, Container, ScalableImage, TextAreaWithOneWildcard, ButtonWithLabel, ButtonWithIcon, Box, Image, Button, SnapshotWidget, TouchArea, ListLayout, ToggleButton, AbstractButton, and Widget.

18.42.4.15 void getVisibleRect (Rect & *rect*) const [virtual]

JSMOC Function for obtaining the visible part of this drawable. If the parent node has a smaller area than this Drawable, the parent will act as a viewport, cutting off the parts of this Drawable that are outside the region. Traverses the tree and yields a result expressed in absolute coordinates.

Parameters

out	*rect*	The subregion of the drawable on which to perform the operation.

18.42.4.16 int16_t getWidth ( ) const [inline]

Gets the width of this drawable.

Returns

The width of this drawable.

18.42.4.17 int16_t getX ( ) const [inline]

Gets the x coordinate of this drawable.

Returns

The x value, relative to the parent.

18.42.4.18 int16_t getY ( ) const [inline]

Gets the y coordinate of this drawable.

Returns

The y value, relative to the parent.

18.42.4.19 void handleClickEvent (const ClickEvent & *evt*) [inline], [virtual]

Defines the event handler interface for ClickEvents. The default implementation ignores the event. The event is only received if the drawable is touchable.

Parameters

evt	The ClickEvent received from the HAL.

Reimplemented in Slider, Keyboard, ScrollableContainer, ToggleButton, RadioButton, TouchArea, and AbstractButton.

18.42.4.20 void handleDragEvent (const DragEvent & *evt*) [inline], [virtual]

Defines the event handler interface for DragEvents. The event is only received if the drawable is touchable.

Parameters

evt	The DragEvent received from the HAL.

Reimplemented in Slider, ScrollableContainer, and TouchArea.

18.42.4.21 void handleGestureEvent (const GestureEvent & *evt*) [inline], [virtual]

Defines the event handler interface for GestureEvents. The default implementation ignores the event. The event is only received if the drawable is touchable.

Parameters

evt	The GestureEvent received from the HAL.

Reimplemented in ScrollableContainer.

18.42.4.22 void handleTickEvent ( ) [inline], [virtual]

Called periodically by the framework if the Drawable instance has subscribed to timer ticks.

See also

Application::registerTimerWidget

Reimplemented in ScrollableContainer, ZoomAnimationImage, and AnimatedImage.

18.42.4.23 void invalidate ( ) const [virtual]

Tell the framework that this entire drawable needs to be redrawn.

See also

invalidateRect

Reimplemented in CanvasWidget.

18.42.4.24 void invalidateRect (Rect & *invalidatedArea*) const [virtual]

Request that a subregion of this drawable is redrawn. Will recursively traverse the tree towards the root, and once reached, issue a draw operation. When this function returns, the specified invalidated area has been redrawn for all appropriate Drawables covering the region.

Parameters

in	*invalidatedArea*	The area of this drawable to redraw expressed in coordinates relative to its parent (e.g. to request a complete redraw, invalidatedArea will be (0, 0, width, height).

18.42.4.25 bool isTouchable ( ) const [inline]

Gets whether this Drawable receives touch events or not.

Returns

True if touch events are received.

See also

setTouchable

18.42.4.26 bool isVisible ( ) const [inline]

Gets whether this Drawable is visible.

Returns

true if the Drawable is visible.
True if visible.

See also

setVisible

18.42.4.27 void moveRelative (int16_t *x*, int16_t *y*) [virtual]

Moves the drawable.

Note

Will redraw the appropriate areas of the screen.

Parameters

x	The relative position to move to.
y	The relative position to move to.

18.42.4.28 void moveTo (int16_t *x*, int16_t *y*) [inline], [virtual]

Moves the drawable.

Note

Will redraw the appropriate areas of the screen.

Parameters

x	The absolute position to move to.
y	The absolute position to move to.

18.42.4.29 void resetDrawChainCache ( ) [inline], [protected]

Reset cached coordinate data.

Note

For TouchGFX internal use only.

18.42.4.30 void setHeight (int16_t *height*) [inline], [virtual]

Sets the height of this drawable.

Note

Changing this does not automatically yield a redraw.

Parameters

height	The new height.

Reimplemented in ZoomAnimationImage.

18.42.4.31 void setPosition (int16_t *x*, int16_t *y*, int16_t *width*, int16_t *height*) [inline], [virtual]

Sets the size and position of this Drawable, relative to its parent.

Note

Changing this does not automatically yield a redraw.

Parameters

x	The x coordinate of this Drawable.
y	The y coordinate of this Drawable.
width	The width of this Drawable.
height	The height of this Drawable.

Reimplemented in ZoomAnimationImage.

18.42.4.32 void setTouchable (bool *touch*) [inline]

Controls whether this Drawable receives touch events or not.

Parameters

touch	If true it will receive touch events, if false it will not.

18.42.4.33 void setupDrawChain (const Rect & *invalidatedArea*, Drawable ** *nextPreviousElement*) [inline], [protected], [virtual]

Configure linked list for draw chain.

Note

For TouchGFX internal use only.

Parameters

	invalidatedArea	Include drawables that intersect with this area only.
in,out	*nextPrevious↩ Element*	Modifiable element in linked list.

Reimplemented in Keyboard, and Container.

18.42.4.34 void setVisible (bool *vis*) [inline]

Controls whether this Drawable should be visible. Only visible Drawables will have their draw function called. Additionally, invisible drawables will not receive input events.

Note

Changing this does not automatically yield a redraw.

Parameters

vis	true if this Drawable should be visible. By default, drawables are visible unless this function has been called with false as argument.

18.42.4.35 void setWidth (int16_t *width*) [inline],[virtual]

Sets the width of this drawable.

Note

Changing this does not automatically yield a redraw.

Parameters

width	The new width.

Reimplemented in ZoomAnimationImage.

18.42.4.36 void setX (int16_t *x*) [inline],[virtual]

Sets the x coordinate of this drawable.

Note

Changing this does not automatically yield a redraw.

Parameters

x	The new x value, relative to the parent. A negative value is allowed.

18.42.4.37 void setXY (int16_t *x*, int16_t *y*) [inline],[virtual]

Sets the x and y coordinates of this drawable.

Note

Changing this does not automatically yield a redraw.

Parameters

x	The new x value, relative to the parent. A negative value is allowed.
y	The new y value, relative to the parent. A negative value is allowed.

18.42.4.38 void setY (int16_t *y*) [inline], [virtual]

Sets the y coordinate of this drawable.

Note

Changing this does not automatically yield a redraw.

Parameters

y	The new y value, relative to the parent. A negative value is allowed.

18.42.4.39 void translateRectToAbsolute (Rect & *r*) const [virtual]

Helper function for converting a specified subregion of this Drawable to absolute coordinates.

Parameters

in,out	*r*	The Rect to translate.

18.43 DrawingSurface Struct Reference

The destination of a draw operation. Contains a pointer to where to draw and the stride of the drawing surface.

`#include <touchgfx/hal/Types.hpp>`

Public Attributes

- uint16_t * address

  The bits.
- int32_t stride

  The stride.

18.44 Bitmap::DynamicBitmapData Struct Reference

Data of a dynamic bitmap.

`#include <touchgfx/Bitmap.hpp>`

Public Attributes

- Rect solid

  The solidRect of this bitmap.
- uint16_t width

  The width of the bitmap.
- uint16_t height

The height of the bitmap.

- uint16_t format

  Determine the format of the data.
- uint8_t inuse

  zero if not in use
- uint8_t alpha

  true if separate alpha for 565 format (always false)

18.44.1 Detailed Description

Data of a dynamic bitmap.

18.45 EasingEquations Class Reference

Defines the "Penner easing functions", which are a de facto standard computing aesthetically pleasing motion animations.

```
#include <touchgfx/EasingEquations.hpp>
```

Static Public Member Functions

- static int16_t backEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Back easing in: Overshooting cubic easing: (s+1)∗t^3 - s∗t^2.
- static int16_t backEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Back easing out: Overshooting cubic easing: (s+1)∗t^3 - s∗t^2.
- static int16_t backEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Back easing in/out: Overshooting cubic easing: (s+1)∗t^3 - s∗t^2.
- static int16_t bounceEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Bounce easing in - exponentially decaying parabolic bounce.
- static int16_t bounceEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Bounce easing out - exponentially decaying parabolic bounce.
- static int16_t bounceEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Bounce easing in/out - exponentially decaying parabolic bounce.
- static int16_t circEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Circular easing in: sqrt(1-t^2)
- static int16_t circEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Circular easing out: sqrt(1-t^2)
- static int16_t circEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Circular easing in/out: sqrt(1-t^2)
- static int16_t cubicEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Cubic easing in: t^3.
- static int16_t cubicEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Cubic easing out: t^3.
- static int16_t cubicEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Cubic easing in/out: t^3.
- static int16_t elasticEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Elastic easing in - exponentially decaying sine wave.
- static int16_t elasticEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Elastic easing out - exponentially decaying sine wave.
- static int16_t elasticEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

Elastic easing in/out - exponentially decaying sine wave.

- static int16_t expoEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Exponential easing in: 2^t.
- static int16_t expoEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Exponential easing out: 2^t.
- static int16_t expoEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Exponential easing in/out: 2^t.
- static int16_t linearEaseNone (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Simple linear tweening - no easing.
- static int16_t linearEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Simple linear tweening - no easing.
- static int16_t linearEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Simple linear tweening - no easing.
- static int16_t linearEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Simple linear tweening - no easing.
- static int16_t quadEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Quadratic easing in: t^2.
- static int16_t quadEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Quadratic easing out: t^2.
- static int16_t quadEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Quadratic easing in/out: t^2.
- static int16_t quartEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Quartic easing in: t^4.
- static int16_t quartEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Quartic easing out: t^4.
- static int16_t quartEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Quartic easing in/out: t^4.
- static int16_t quintEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Quintic/strong easing in: t^5.
- static int16_t quintEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Quintic/strong easing out: t^5.
- static int16_t quintEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Quintic/strong easing in/out: t^5.
- static int16_t sineEaseIn (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Sinusoidal easing in: sin(t)
- static int16_t sineEaseOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Sinusoidal easing out: sin(t)
- static int16_t sineEaseInOut (uint16_t t, int16_t b, int16_t c, uint16_t d)

  Sinusoidal easing in/out: sin(t)

18.45.1 Detailed Description

Defines the "Penner easing functions", which are a de facto standard computing aesthetically pleasing motion animations. See `http://easings.net/` for visual illustrations of the easing equations.

See also

`http://easings.net/`

18.45.2 Member Function Documentation

18.45.2.1 static int16_t backEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Back easing in: Overshooting cubic easing: (s+1)∗t^3 - s∗t^2. Backtracking slightly, then reversing direction and moving to target.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.2 static int16_t backEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Back easing in/out: Overshooting cubic easing: (s+1)∗t^3 - s∗t^2. Backtracking slightly, then reversing direction and moving to target, then overshooting target, reversing, and finally coming back to target.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.3 static int16_t backEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Back easing out: Overshooting cubic easing: (s+1)∗t^3 - s∗t^2. Moving towards target, overshooting it slightly, then reversing and coming back to target.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.4 static int16_t bounceEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Bounce easing in - exponentially decaying parabolic bounce.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.

d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.5 static int16_t bounceEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Bounce easing in/out - exponentially decaying parabolic bounce.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.6 static int16_t bounceEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Bounce easing out - exponentially decaying parabolic bounce.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.7 static int16_t circEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Circular easing in: sqrt(1-t^2). Accelerating from zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.8 static int16_t circEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Circular easing in/out: sqrt(1-t^2). Acceleration until halfway, then deceleration.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.9 static int16_t circEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Circular easing out: sqrt(1-t^2). Decelerating to zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.10 static int16_t cubicEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Cubic easing in: t^3. Accelerating from zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.11 static int16_t cubicEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Cubic easing in/out: t^3. Acceleration until halfway, then deceleration.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.12 static int16_t cubicEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Cubic easing out: t^3. Decelerating to zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.13 static int16_t elasticEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Elastic easing in - exponentially decaying sine wave.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.14 static int16_t elasticEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Elastic easing in/out - exponentially decaying sine wave.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.15 static int16_t elasticEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Elastic easing out - exponentially decaying sine wave.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.16 static int16_t expoEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Exponential easing in: 2^t. Accelerating from zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.17 static int16_t expoEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Exponential easing in/out: 2^t. Accelerating until halfway, then decelerating.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.18 static int16_t expoEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Exponential easing out: 2^t. Deceleration to zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.19 static int16_t linearEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Simple linear tweening - no easing.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.20 static int16_t linearEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Simple linear tweening - no easing.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.21 static int16_t linearEaseNone (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Simple linear tweening - no easing.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.22 static int16_t linearEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.23 static int16_t quadEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Quadratic easing in: t^2. Accelerating from zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.24 static int16_t quadEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Quadratic easing in/out: t^2. Acceleration until halfway, then deceleration.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.25 static int16_t quadEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Quadratic easing out: t^2. Decelerating to zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.26 static int16_t quartEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Quartic easing in: t^4. Accelerating from zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.27 static int16_t quartEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Quartic easing in/out: t^4. Acceleration until halfway, then deceleration.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.28 static int16_t quartEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Quartic easing out: t^4. Decelerating to zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.29 static int16_t quintEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Quintic/strong easing in: t^5. Accelerating from zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.30 static int16_t quintEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Quintic/strong easing in/out: t^5. Acceleration until halfway, then deceleration.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.31 static int16_t quintEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Quintic/strong easing out: t^5. Decelerating to zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.32 static int16_t sineEaseIn (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Sinusoidal easing in: sin(t). Accelerating from zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.33 static int16_t sineEaseInOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Sinusoidal easing in/out: sin(t). Acceleration until halfway, then deceleration.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.45.2.34 static int16_t sineEaseOut (uint16_t *t*, int16_t *b*, int16_t *c*, uint16_t *d*) [static]

Sinusoidal easing out: sin(t). Decelerating to zero velocity.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

18.46 Edge Struct Reference

An edge contains information about one edge, between two points, of a triangle, as well as information about how to interpolate values when moving in the vertical direction.

```
#include <touchgfx/TextureMapTypes.hpp>
```

Public Member Functions

- Edge (const Gradients &gradients, const Point3D *vertices, int top, int bottom)

  Constructor.
- int step ()

  Perform a step along the edge.
- int step (int steps)

  Performs a number of steps along the edge.

Public Attributes

- int32_t X

  The X coordinate.
- int32_t XStep

  Amount to increment x.
- int32_t numerator

  The numerator.
- int32_t denominator

  The denominator.
- int32_t errorTerm

  The error term.
- int Y

  The Y coordinate.
- int height

  The height.
- float oneOverZ

  The one over z coordinate.
- float oneOverZStep

  The one over z coordinate step.
- float oneOverZStepExtra

  The one over z coordinate step extra.
- float UOverZ

  The over z coordinate.
- float UOverZStep

  The over z coordinate step.
- float UOverZStepExtra

  The over z coordinate step extra.
- float VOverZ

  The over z coordinate.
- float VOverZStep

  The over z coordinate step.
- float VOverZStepExtra

  The over z coordinate step extra.

18.46.1 Constructor & Destructor Documentation

18.46.1.1 Edge (const Gradients & *gradients*, const Point3D * *vertices*, int *top*, int *bottom*)

Construct the edge between two vertices and using the gradients for calculating the interpolation values.

Parameters

gradients	The gradients for the triangle.
vertices	The vertices for the triangle.
top	The index in the vertices array of the top vertex of this edge.
bottom	The index in the vertices array of the bottom vertex of this edge.

18.46.2 Member Function Documentation

18.46.2.1 int step ( ) [inline]

Perform a step along the edge.

Returns

the Height.

18.46.2.2 int step (int *steps*) [inline]

Performs a number of steps along the edge.

Parameters

steps	The number of steps the perform.

Returns

height.

18.47 Event Class Reference

Simple base class for events.

`#include <touchgfx/Event.hpp>`

Public Types

- enum EventType { EVENT_CLICK, EVENT_DRAG, EVENT_GESTURE }
  The events types.

Public Member Functions

- virtual EventType getEventType ()=0
  Gets event type.
- virtual ~Event ()
  Destructor.

18.47.1 Detailed Description

Simple base class for events.

18.47.2 Member Enumeration Documentation

18.47.2.1 enum enum EventType

The events types.

Enumerator

EVENT_CLICK A click.
EVENT_DRAG A drag.
EVENT_GESTURE A gesture.

18.47.3 Constructor & Destructor Documentation

18.47.3.1 ~Event ( ) [inline],[virtual]

Destructor.

18.47.4 Member Function Documentation

18.47.4.1 EventType getEventType ( ) [pure virtual]

Gets event type.

Returns

The type of this event.

Implemented in ClickEvent, DragEvent, and GestureEvent.

18.48 FadeAnimator< T > Class Template Reference

A FadeAnimator makes the template class T able to animate an alpha fade.

```
#include <touchgfx/mixins/FadeAnimator.hpp>
```

Public Member Functions

- FadeAnimator ()

  Default constructor.
- virtual ~FadeAnimator ()

  Destructor.
- void setFadeAnimationEndedAction (GenericCallback< const FadeAnimator< T > & > &callback)

  Associates an action to be performed when the animation ends.
- virtual void setFadeAnimationDelay (uint16_t delay)

  Sets a delay on animations done by the FadeAnimator.
- virtual uint16_t getFadeAnimationDelay () const

  Gets the current animation delay.
- virtual bool isRunning () const

  Gets whether or not the fade animation is running.
- void startFadeAnimation (uint8_t endAlpha, uint16_t duration, EasingEquation alphaProgression↩Equation=&EasingEquations::linearEaseNone)

  Starts the fade animation.
- void cancelFadeAnimation ()

  Cancel fade animation.

Protected Member Functions

- virtual void handleTickEvent ()

  The tick handler that handles the actual animation steps.

Protected Attributes

- bool fadeAnimationRunning

  Boolean that is true if the animation is running.
- uint16_t fadeAnimationCounter

  Counter that is equal to the current step in the animation.
- uint16_t fadeAnimationDelay

  A delay that is applied before animation start. Expressed in ticks.
- uint16_t fadeAnimationDuration

  The complete duration of the animation. Expressed in ticks.
- int16_t fadeAnimationStartAlpha

  The alpha value at the beginning of the animation.
- int16_t fadeAnimationEndAlpha

  The alpha value at the end of the animation.
- EasingEquation fadeAnimationAlphaEquation

  EasingEquation expressing the development of the alpha value during the animation.
- GenericCallback< const FadeAnimator< T > & > * fadeAnimationEndedCallback

  Animation ended Callback.

18.48.1 Detailed Description

template<class T>class touchgfx::FadeAnimator< T >

A FadeAnimator makes the template class T able to animate an alpha fade from its current alpha value to a specified end alpha value. The alpha development can be described by supplying an EasingEquation. The FadeAnimator performs a callback when the animation has finished.

This mixin can be used on any Drawable that has a 'void setAlpha(uint8_t)' and a 'uint8_t getAlpha()' method.

Template Parameters

T	Specifies the type should have the fade animation capability.

18.48.2 Constructor & Destructor Documentation

18.48.2.1 FadeAnimator ( ) [inline]

Default constructor. Creates and initialize the FadeAnimator.

18.48.2.2 ~FadeAnimator ( ) [inline],[virtual]

Destructor. Destroys the FadeAnimator.

18.48.3 Member Function Documentation

18.48.3.1 uint16_t getFadeAnimationDelay ( ) const [inline],[virtual]

Gets the current animation delay.

Returns

The current animation delay.

18.48.3.2 void handleTickEvent ( ) [inline],[protected],[virtual]

The tick handler that handles the actual animation steps.

18.48.3.3 bool isRunning ( ) const [inline],[virtual]

Gets whether or not the fade animation is running.

Returns

true if the fade animation is running.

18.48.3.4 void setFadeAnimationDelay (uint16_t *delay*) [inline],[virtual]

Sets a delay on animations done by the FadeAnimator.

Parameters

delay	The delay in ticks.

18.48.3.5 void setFadeAnimationEndedAction (GenericCallback< const FadeAnimator< T > & > & *callback*) [inline]

Associates an action to be performed when the animation ends.

Parameters

callback	The callback to be executed. The callback will be given a reference to the FadeAnimator.

See also

GenericCallback

18.48.3.6 void startFadeAnimation (uint8_t *endAlpha*, uint16_t *duration*, EasingEquation *alphaProgressionEquation* = &EasingEquations::linearEaseNone) [inline]

Starts the fade animation from the current alpha value to the specified end alpha value. The development of the alpha value during the animation is described by the supplied EasingEquation.

Parameters

endAlpha	The alpha value of T at animation end.
duration	The duration of the animation measured in ticks.
alpha↩ Progression↩ Equation	The equation that describes the development of the alpha value during the animation. Default = EasingEquations::linearEaseNone.

18.49 Font Class Reference

The font base class.

```
#include <C:\Projects\devel\touchgfx\framework\include\touchgfx\Font.hpp>
```

Public Member Functions

- virtual const GlyphNode * getGlyph (Unicode::UnicodeChar unicode, const uint8_t *&pixelData, uint8_t &bitsPerPixel) const =0

  Gets the glyph data associated with the specified unicode.
- virtual const GlyphNode * getGlyph (Unicode::UnicodeChar unicode) const

  Gets the glyph data associated with the specified unicode.
- virtual uint16_t getStringWidth (const Unicode::UnicodeChar *text,...) const

  Gets the width in pixels of the specified string.
- virtual uint16_t getStringWidth (TextDirection textDirection, const Unicode::UnicodeChar *text,...) const

  Gets the width in pixels of the specified string.
- virtual uint16_t getCharWidth (const Unicode::UnicodeChar c) const

  Gets the width in pixels of the specified character.
- virtual uint8_t getSpacingAbove (const Unicode::UnicodeChar *text,...) const

  Gets the number of blank pixels at the top of the given text.
- virtual uint16_t getMaxTextHeight (const Unicode::UnicodeChar *text,...) const

  Gets the height of the highest character in a given string.
- virtual uint16_t getFontHeight () const

  Returns the height in pixels of this font.
- virtual uint16_t getMinimumTextHeight () const

  Returns the minimum height needed for a text field that uses this font.
- virtual uint8_t getBitsPerPixel () const

  Gets bits per pixel for this font.
- uint8_t getMaxPixelsLeft () const

  Gets maximum pixels left.
- uint8_t getMaxPixelsRight () const

  Gets maximum pixels right.
- virtual int8_t getKerning (Unicode::UnicodeChar prevChar, const GlyphNode *glyph) const

  Gets the kerning distance between two characters.
- virtual uint16_t getNumberOfLines (const Unicode::UnicodeChar *text,...) const

  Gets number of lines.

Protected Types

- typedef uint16_t(Font::* StringWidthFunctionPointer)(TextDirection textDirection, const Unicode::UnicodeChar *text, va_list pArg) const

  Defines an alias representing the constant.

Protected Member Functions

- uint16_t getStringWidthLTR (TextDirection textDirection, const Unicode::UnicodeChar *text, va_list pArg) const

  Gets the width in pixels of the specified string.
- uint16_t getStringWidthRTL (TextDirection textDirection, const Unicode::UnicodeChar *text, va_list pArg) const

  Gets the width in pixels of the specified string.
- Font (uint16_t height, uint8_t pixBelowBase, uint8_t bitsPerPixel, uint8_t maxLeft, uint8_t maxRight)

  Constructor.

Protected Attributes

- uint16_t fontHeight

  The font height in pixels.
- uint8_t pixelsBelowBaseline

  The number of pixels below the base line.
- uint8_t bPerPixel

  The number of bits per pixel.
- uint8_t maxPixelsLeft

  The maximum number of pixels a glyph extends to the left.
- uint8_t maxPixelsRight

  The maximum number of pixels a glyph extends to the right.

Static Protected Attributes

- static StringWidthFunctionPointer getStringWidthFunction

  The getStringWidth function, either LTR (supporting LTR only) or RTL (supporting RTL and LTR)

18.49.1 Detailed Description

The font base class. This class is abstract and requires the implementation of getGlyph.

It provides utility functions such as obtaining string width and font height.

18.49.2 Constructor & Destructor Documentation

18.49.2.1 Font (uint16_t *height*, uint8_t *pixBelowBase*, uint8_t *bitsPerPixel*, uint8_t *maxLeft*, uint8_t *maxRight*) [inline],[protected]

The protected constructor of a Font.

Parameters

height	The font height in pixels.
pixBelowBase	The number of pixels below the base line.
bitsPerPixel	The number of bits per pixel.
maxLeft	The maximum left extend for a glyph in the font.
maxRight	The maximum right extend for a glyph in the font.

18.49.3 Member Function Documentation

18.49.3.1 uint8_t getBitsPerPixel ( ) const [inline],[virtual]

Gets bits per pixel for this font.

Returns

The number of bits used per pixel in this font.

18.49.3.2 uint16_t getCharWidth (const Unicode::UnicodeChar *c*) const [virtual]

Gets the width in pixels of the specified character.

Parameters

c	The unicode character.

Returns

The width in pixels of the specified character.

18.49.3.3 uint16_t getFontHeight ( ) const [inline], [virtual]

Returns the height in pixels of this font. The returned value corresponds to the maximum height occupied by a character in the font.

Note

It is not sufficient to allocate text areas with this height. Use getMinimumTextHeight for this.

Returns

The height in pixels of this font.

18.49.3.4 const GlyphNode * getGlyph (Unicode::UnicodeChar *unicode*, const uint8_t *& *pixelData*, uint8_t & *bitsPerPixel*) const [pure virtual]

Gets the glyph data associated with the specified unicode.

Parameters

	unicode	The character to look up.
	pixelData	Pointer to the pixel data for the glyph if the glyph is found. This is set by this method.
out	*bitsPerPixel*	Reference where to place the number of bits per pixel.

Returns

A pointer to the glyph node or null if the glyph was not found.

Implemented in ConstFont.

18.49.3.5 const GlyphNode * getGlyph (Unicode::UnicodeChar *unicode*) const [inline], [virtual]

Gets the glyph data associated with the specified unicode.

Parameters

unicode	The character to look up.

Returns

A pointer to the glyph node or null if the glyph was not found.

18.49.3.6 int8_t getKerning (Unicode::UnicodeChar *prevChar*, const GlyphNode * *glyph*) const [inline], [virtual]

Gets the kerning distance between two characters.

Parameters

prevChar	The unicode value of the previous character.
glyph	the glyph object for the current character.

Returns

The kerning distance between prevChar and glyph char.

Reimplemented in ConstFont, and InternalFlashFont.

18.49.3.7 uint8_t getMaxPixelsLeft ( ) const [inline]

Gets maximum pixels left for any glyph in the font. This is the max value of "left" for all glyphs. The value is negated so if a "g" has left=-6 maxPixelsLeft is 6. This value is calculated by the font converter.

Returns

The maximum pixels left.

18.49.3.8 uint8_t getMaxPixelsRight ( ) const [inline]

Gets maximum pixels right for any glyph in the font. This is the max value of "width+left-advance" for all glyphs. The is the number of pixels a glyph reaches to the right of its normal area. This value is calculated by the font converter.

Returns

The maximum pixels right.

18.49.3.9 uint16_t getMaxTextHeight (const Unicode::UnicodeChar * *text*, ...) const [virtual]

Gets the height of the highest character in a given string. The height includes the spacing above the text which is included in the font.

Parameters

text	A zero-terminated unicode string.
...	Variable arguments providing additional information.

Returns

The height if the given text.

18.49.3.10 uint16_t getMinimumTextHeight ( ) const [inline], [virtual]

Returns the minimum height needed for a text field that uses this font. Takes into account that certain characters (eg 'g') have pixels below the baseline, thus making the text height larger than the font height.

Returns

The minimum height needed for a text field that uses this font.

18.49.3.11 uint16_t getNumberOfLines (const Unicode::UnicodeChar * *text*, ...) const [virtual]

Count the number of lines in a given text.

Parameters

text	The text.
...	Variable arguments providing additional information.

Returns

The number of lines.

18.49.3.12 uint8_t getSpacingAbove (const Unicode::UnicodeChar * *text*, ...) const [virtual]

Gets the number of blank pixels at the top of the given text.

Parameters

text	A zero-terminated unicode string.
...	Variable arguments providing additional information.

Returns

The number of blank pixels above the text.

18.49.3.13 uint16_t getStringWidth (const Unicode::UnicodeChar * *text*, ...) const [virtual]

Gets the width in pixels of the specified string. If the string contains multiple lines, the width of the widest line is found. Please note that the correct number of arguments must be given if the text contains wildcards.

It is recommended to use the getStringWidth() implementation with the TextDirection parameter to ensure correct calculation of the width. Kerning could result in different results depending on the TextDirection. This method assumes TextDirection to be TEXT_DIRECTION_LTR.

Parameters

text	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
...	Variable arguments providing additional information.

Returns

The width in pixels of the longest line of the specified string.

18.49.3.14 uint16_t getStringWidth (TextDirection *textDirection*, const Unicode::UnicodeChar * *text*, ...) const [virtual]

Gets the width in pixels of the specified string. If the string contains multiple lines, the width of the widest line is found. Please note that the correct number of arguments must be given if the text contains wildcards.

The TextDirection should be set correctly for the text supplied. For example the string "10 20 30" will be calculated differently depending on the TextDirection. If TextDirection is TEXT_DIRECTION_LTR the width is calculated as the with of "10 20 30" (with kerning between all characters) but for TEXT_DIRECTION_RTL it is calculated as "10"+" "+"20"+" "+"30" (with kerning only between characters in the substrings and not between substrings). For most fonts there might not be a difference between the two calculations, but some fonts might cause different results.

Parameters

textDirection	The text direction.
text	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
...	Variable arguments providing additional information.

Returns

The width in pixels of the longest line of the specified string.

18.49.3.15 uint16_t getStringWidthLTR (TextDirection *textDirection*, const Unicode::UnicodeChar * *text*, va_list *pArg*) const [protected]

Gets the width in pixels of the specified string. If the string contains multiple lines, the width of the widest line is found. Please note that the correct number of arguments must be given if the text contains wildcards.

The string is assumed to be purely left-to-right.

Parameters

textDirection	The text direction.
text	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
pArg	Variable arguments providing additional information.

Returns

The width in pixels of the longest line of the specified string.

18.49.3.16 uint16_t getStringWidthRTL (TextDirection *textDirection*, const Unicode::UnicodeChar * *text*, va_list *pArg*) const [protected]

Gets the width in pixels of the specified string. If the string contains multiple lines, the width of the widest line is found. Please note that the correct number of arguments must be given if the text contains wildcards.

The string is handled as a right-to-left string and subdivided into smaller text strings to correctly handle mixing of left-to-right and right-to-left strings.

Parameters

textDirection	The text direction.
text	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
pArg	The argument.

Returns

The string width RTL.

18.50 FontManager Class Reference

This class is the entry point for looking up a font based on a font id.

```
#include <touchgfx/FontManager.hpp>
```

Static Public Member Functions

- static void setFontProvider (FontProvider *fontProvider)

  Sets the font provider.
- static Font * getFont (FontId fontId)

  Gets a font.

18.50.1 Detailed Description

This class is the entry point for looking up a font based on a font id. Must be initialized with the appropriate FontProvider by the application.

18.50.2 Member Function Documentation

18.50.2.1 static Font * getFont (FontId *fontId*) [static]

Gets a font.

Parameters

fontId	The font id of the font to get.

Returns

The font with a font id of fontId.

18.50.2.2 static void setFontProvider (FontProvider * *fontProvider*) [static]

Sets the font provider. Must be initialized with the appropriate FontProvider by the application.

Parameters

in	*fontProvider*	Sets the font provider. Must be initialized with the appropriate FontProvider by the application.

18.51 FontProvider Class Reference

A generic pure virtual definition of a FontProvider.

`#include <touchgfx/FontManager.hpp>`

Public Member Functions

- virtual Font * getFont (FontId fontId)=0

  Gets a font.
- virtual ~FontProvider ()

  Destructor.

18.51.1 Detailed Description

A generic pure virtual definition of a FontProvider, which is a class capable of returning a font based on a font id. An application-specific derivation of this class must be implemented.

18.51.2 Constructor & Destructor Documentation

18.51.2.1 ~FontProvider ( ) [inline], [virtual]

Destructor.

18.51.3 Member Function Documentation

18.51.3.1 Font * getFont (FontId *fontId*) [pure virtual]

Gets a font.

Parameters

fontId	The font id of the font to get.

Returns

The font with a font id of fontId.

18.52 GenericCallback< T1, T2 > Class Template Reference

GenericCallback is the base class for callbacks.

```
#include <touchgfx/Callback.hpp>
```

Public Member Functions

- virtual ~GenericCallback ()

  Destructor.
- virtual void execute (T1 val1, T2 val2)=0

  Calls the member function.
- virtual bool isValid () const =0

  Function to check whether the Callback has been initialized with values.

18.52.1 Detailed Description

template<class T1 = void, class T2 = void>class touchgfx::GenericCallback< T1, T2 >

GenericCallback is the base class for callbacks.

See also

Callback for an explanation of callbacks.

The reason this base class exists, is that a normal Callback requires the class type where the callback function resides to be known. This is problematic for ie. framework widgets like AbstractButton, on which it should be possible to register a callback on object types that are user-specific and thus unknown to AbstractButton. This is solved by having AbstractButton contain a pointer to a GenericCallback instead. This pointer must then be initialized to point on an instance of Callback, created by the user, which is initialized with the appropriate object type.

Note

As with Callback, this class exists in three versions to support callback functions taking zero, one or two arguments.

Template Parameters

T1	The type of the first argument in the member function, or void if none.
T2	The type of the second argument in the member function, or void if none.

18.52.2 Constructor & Destructor Documentation

18.52.2.1 ~GenericCallback ( ) [inline],[virtual]

Empty virtual destructor.

18.52.3 Member Function Documentation

18.52.3.1 virtual void execute (T1 *val1*, T2 *val2*) [pure virtual]

Calls the member function. Do not call execute unless isValid() returns true (ie. a pointer to the object and the function has been set).

Parameters

val1	This value will be passed as the first argument in the function call.
val2	This value will be passed as the second argument in the function call.

Implemented in Callback< dest_type, T1, T2 >, Callback< touchgfx::RadioButtonGroup, const touchgfx::AbstractButton & >, and Callback< touchgfx::SlideTransition, touchgfx::Drawable & >.

18.52.3.2 bool isValid ( ) const [pure virtual]

Function to check whether the Callback has been initialized with values.

Returns

true If the callback is valid (i.e. safe to call execute).

Implemented in Callback< dest_type, void, void >, Callback< dest_type, T1, void >, Callback< dest_type, T1, T2 >, Callback< touchgfx::RadioButtonGroup, const touchgfx::AbstractButton & >, and Callback< touchgfx::SlideTransition, touchgfx::Drawable & >.

18.53 GenericCallback< T1, void > Class Template Reference

GenericCallback is the base class for callbacks.

```
#include <touchgfx/Callback.hpp>
```

Public Member Functions

- virtual ~GenericCallback ()

  Destructor.
- virtual void execute (T1 val1)=0

  Calls the member function.
- virtual bool isValid () const =0

  Function to check whether the Callback has been initialized with values.

18.53.1 Detailed Description

template<class T1>class touchgfx::GenericCallback< T1, void >

GenericCallback is the base class for callbacks.

See also

Callback for an explanation of callbacks.

The reason this base class exists, is that a normal Callback requires the class type where the callback function resides to be known. This is problematic for ie. framework widgets like AbstractButton, on which it should be possible to register a callback on object types that are user-specific and thus unknown to AbstractButton. This is solved by having AbstractButton contain a pointer to a GenericCallback instead. This pointer must then be initialized to point on an instance of Callback, created by the user, which is initialized with the appropriate object type.

Note

As with Callback, this class exists in three versions to support callback functions taking zero, one or two arguments.

Template Parameters

T1	The type of the first argument in the member function, or void if none.

18.53.2 Constructor & Destructor Documentation

18.53.2.1 ~GenericCallback ( ) [inline], [virtual]

Empty virtual destructor.

18.53.3 Member Function Documentation

18.53.3.1 void execute (T1 *val1*) [pure virtual]

Calls the member function. Do not call execute unless isValid() returns true (ie. a pointer to the object and the function has been set).

Parameters

val1	This value will be passed as the first argument in the function call.

18.53.3.2 bool isValid ( ) const [pure virtual]

Function to check whether the Callback has been initialized with values.

Returns

true If the callback is valid (i.e. safe to call execute).

18.54 GenericCallback< void > Class Template Reference

GenericCallback is the base class for callbacks.

```
#include <touchgfx/Callback.hpp>
```

Public Member Functions

- virtual ~GenericCallback ()

  Destructor.
- virtual void execute ()=0

  Calls the member function.
- virtual bool isValid () const =0

  Function to check whether the Callback has been initialized with values.

18.54.1 Detailed Description

template<>class touchgfx::GenericCallback< void >

GenericCallback is the base class for callbacks.

See also

Callback for an explanation of callbacks.

The reason this base class exists, is that a normal Callback requires the class type where the callback function resides to be known. This is problematic for ie. framework widgets like AbstractButton, on which it should be possible to register a callback on object types that are user-specific and thus unknown to AbstractButton. This is solved by having AbstractButton contain a pointer to a GenericCallback instead. This pointer must then be initialized to point on an instance of Callback, created by the user, which is initialized with the appropriate object type.

Note

As with Callback, this class exists in three versions to support callback functions taking zero, one or two arguments.

18.54.2 Constructor & Destructor Documentation

18.54.2.1 ~GenericCallback ( ) [inline],[virtual]

Empty virtual destructor.

18.54.3 Member Function Documentation

18.54.3.1 void execute ( ) [pure virtual]

Calls the member function. Do not call execute unless isValid() returns true (ie. a pointer to the object and the function has been set).

18.54.3.2 bool isValid ( ) const [pure virtual]

Function to check whether the Callback has been initialized with values.

Returns

true If the callback is valid (i.e. safe to call execute).

18.55 GestureEvent Class Reference

A gesture event.

```
#include <touchgfx/events/GestureEvent.hpp>
```

Public Types

- enum GestureType { SWIPE_HORIZONTAL, SWIPE_VERTICAL }

  The gesture event types.

Public Member Functions

- GestureEvent (GestureType t, int16_t v, int16_t x_coord, int16_t y_coord)

  Constructor.
- int16_t getVelocity () const

  Gets the velocity of this gesture event.
- GestureType getType () const

  Gets the type of this gesture event.
- int16_t getX () const

  Gets the x coordinate of this gesture event.
- int16_t getY () const

  Gets the y coordinate of this gesture event.
- virtual Event::EventType getEventType ()

  Gets event type.

18.55.1 Detailed Description

A gesture event. The only gesture events currently supported is SWIPE_HORIZONTAL and SWIPE_VERTICAL, which will be issued every time the input system detects a swipe.

See also

Event

18.55.2 Member Enumeration Documentation

18.55.2.1 enum enum GestureType

Enumerator

SWIPE_HORIZONTAL An enum constant representing a horizontal swipe.

SWIPE_VERTICAL An enum constant representing a vertical swipe.

18.55.3 Constructor & Destructor Documentation

18.55.3.1 GestureEvent (GestureType *t*, int16_t *v*, int16_t *x_coord*, int16_t *y_coord*) [inline]

Constructor. Create a gesture event of the specified type with the specified coordinates.

Parameters

t	The type of the gesture event.
v	The velocity of this gesture (swipe)
x_coord	The x coordinate of the gesture.

y_coord	The y coordinate of the gesture.

18.55.4 Member Function Documentation

18.55.4.1 Event::EventType getEventType ( ) [inline],[virtual]

Returns

The type of this event.

Implements Event.

18.55.4.2 GestureType getType ( ) const [inline]

Gets the type of this gesture event.

Returns

The type of this gesture event.

18.55.4.3 int16_t getVelocity ( ) const [inline]

Gets the velocity of this gesture event.

Returns

The velocity of this gesture event.

18.55.4.4 int16_t getX ( ) const [inline]

Gets the x coordinate of this gesture event.

Returns

The x coordinate of this gesture event.

18.55.4.5 int16_t getY ( ) const [inline]

Gets the y coordinate of this gesture event.

Returns

The y coordinate of this gesture event.

18.56 Gestures Class Reference

This class implements the detection of gestures.

```
#include <touchgfx/hal/Gestures.hpp>
```

Public Member Functions

- Gestures ()

  Default constructor.
- void registerEventListener (UIEventListener &l)

  Register the event listener.
- void tick ()

  Has to be called during the timer tick.
- bool registerDragEvent (uint16_t oldX, uint16_t oldY, uint16_t newX, uint16_t newY)

  Register a drag event.
- void registerClickEvent (ClickEvent::ClickEventType evt, uint16_t x, uint16_t y)

  Register a click event and figure out if this is a drag event, too.
- void setDragThreshold (uint16_t val)

  Configure the threshold for reporting drag events.

18.56.1 Detailed Description

This class implements the detection of gestures.

18.56.2 Constructor & Destructor Documentation

18.56.2.1 Gestures ( ) [inline]

Default constructor. Does nothing.

18.56.3 Member Function Documentation

18.56.3.1 void registerClickEvent (ClickEvent::ClickEventType *evt*, uint16_t *x*, uint16_t *y*)

Register a click event and figure out if this is a drag event, too.

Parameters

evt	The type of the click event.
x	The x coordinate of the click event.
y	The y coordinate of the click event.

18.56.3.2 bool registerDragEvent (uint16_t *oldX*, uint16_t *oldY*, uint16_t *newX*, uint16_t *newY*)

Register a drag event.

Parameters

oldX	The x coordinate of the drag start position (dragged from)
oldY	The y coordinate of the drag start position (dragged from)
newX	The x coordinate of the new position (dragged to)
newY	The y coordinate of the new position (dragged to)

Returns

True if the drag exceeds threshold value (and therefore was reported as a drag), or false if the drag did not exceed threshold (and therefore was discarded).

18.56.3.3 void registerEventListener (UIEventListener & l)

Register the event listener.

Parameters

in	*l*	The EventListener to register.

18.56.3.4 void setDragThreshold (uint16_t *val*) [inline]

Configure the threshold for reporting drag events. A touch input movement must exceed this value in either axis in order to report a drag. Default value is 0.

Parameters

val	New threshold value.

18.56.3.5 void tick ( )

Has to be called during the timer tick.

18.57 GlyphNode Struct Reference

struct providing information about a glyph.

`#include <touchgfx/Font.hpp>`

Public Member Functions

- uint16_t width () const

  Gets the "width" value where the 9th bit is stored in flags.
- uint16_t height () const

  Gets the "height" value where the 9th bit is stored in flags.
- int16_t top () const

  Gets the "top" value where the 9th bit and the sign bit are stored in flags.
- uint16_t advance () const

  Gets the "advance" value where the 9th bit is stored in flags.

Public Attributes

- uint32_t dataOffset

  The index to the data of this glyph.
- Unicode::UnicodeChar unicode

  The unicode of this glyph.
- uint8_t _width

  Width of the actual glyph data.
- uint8_t _height

  Height of the actual glyph data.
- uint8_t _top

  Vertical offset from baseline of the glyph.
- int8_t left

  Horizontal offset from the left of the glyph.
- uint8_t _advance

  Width of the glyph (including space to the left and right)
- uint8_t kerningTablePos

Where are the kerning information for this glyph stored in the kerning table.

- uint8_t kerningTableSize

  How many entries are there in the kerning table (following kerningTablePos) for this glyph.

- uint8_t flags

  Additional glyph flags (font encoding and extra precision for width/height/top/advance)

18.57.1 Detailed Description

struct providing information about a glyph. Used by LCD when rendering.

18.57.2 Member Function Documentation

18.57.2.1 uint16_t advance ( ) const [inline]

Gets the "advance" value where the 9th bit is stored in flags.

Returns

the right value of "advance".

18.57.2.2 uint16_t height ( ) const [inline]

Gets the "height" value where the 9th bit is stored in flags.

Returns

the right value of "height".

18.57.2.3 int16_t top ( ) const [inline]

Gets the "top" value where the 9th bit and the sign bit are stored in flags.

Returns

the right value of "top".

18.57.2.4 uint16_t width ( ) const [inline]

Gets the "width" value where the 9th bit is stored in flags.

Returns

the right value of "width".

18.58 GPIO Class Reference

Interface class for manipulating GPIOs in order to do performance measurements on target.

```
#include <touchgfx/hal/GPIO.hpp>
```

Public Types

- enum GPIO_ID { **VSYNC_FREQ**, RENDER_TIME, FRAME_RATE, MCU_ACTIVE }

  Enum for the GPIOs used.

Static Public Member Functions

- static void init ()

  Perform configuration of IO pins.
- static void set (GPIO_ID id)

  Sets a pin high.
- static void clear (GPIO_ID id)

  Sets a pin low.
- static void toggle (GPIO_ID id)

  Toggles a pin.
- static bool get (GPIO_ID id)

  Gets the state of a pin.

18.58.1 Detailed Description

Interface class for manipulating GPIOs in order to do performance measurements on target. Not used on the PC simulator.

18.58.2 Member Enumeration Documentation

18.58.2.1 enum enum GPIO_ID

Enum for the GPIOs used.

Enumerator

RENDER_TIME Pin is toggled at each VSYNC.

FRAME_RATE Pin is high when frame rendering begins, low when finished.

MCU_ACTIVE Pin is toggled when the frame buffers are swapped. Pin is high when the MCU is doing work (i.e. not in idle task).

18.58.3 Member Function Documentation

18.58.3.1 static void clear (GPIO_ID *id*) [static]

Sets a pin low.

Parameters

id	the pin to set.

18.58.3.2 static bool get (GPIO_ID *id*) [static]

Gets the state of a pin.

Parameters

id	the pin to get.

Returns

true if the pin is high, false otherwise.

18.58.3.3 static void init ( ) [static]

Perform configuration of IO pins.

18.58.3.4 static void set (GPIO_ID *id*) [static]

Sets a pin high.

Parameters

id	the pin to set.

18.58.3.5 static void toggle (GPIO_ID *id*) [static]

Toggles a pin.

Parameters

id	the pin to toggle.

18.59 Gradients Struct Reference

Gradients contains all the data to interpolate u,v texture coordinates and z coordinates across a planar surface.

`#include <touchgfx/TextureMapTypes.hpp>`

Public Member Functions

- Gradients (const Point3D *vertices)

  Constructor. Construct the gradients using 3 3D vertices.

Public Attributes

- float oneOverZ [3]

  1/z for each vertex
- float UOverZ [3]

  u/z for each vertex
- float VOverZ [3]

  v/z for each vertex
- float dOneOverZdX

  d(1/z)/dX
- float dOneOverZdY

  d(1/z)/dY

- float dUOverZdX

  d(u/z)/dX
- float dUOverZdY

  d(u/z)/dY
- float dVOverZdX

  d(v/z)/dX
- float dVOverZdY

  d(v/z)/dY
- fixed16_16 dUdXModifier

  The dUdX x coordinate modifier.
- fixed16_16 dVdXModifier

  The dVdX x coordinate modifier.

18.59.1 Constructor & Destructor Documentation

18.59.1.1 Gradients (const Point3D * *vertices*)

Parameters

vertices	The vertices.

See also

Point3D

18.60 HAL Class Reference

Hardware Abstraction Layer.

```
#include <touchgfx/hal/HAL.hpp>
```

Public Member Functions

- HAL (DMA_Interface &dmaInterface, LCD &display, TouchController &touchCtrl, uint16_t width, uint16_↩ t height)

  Creates a HAL instance.
- virtual ~HAL ()

  Destructor.
- virtual void setDisplayOrientation (DisplayOrientation orientation)

  Sets the desired display orientation (landscape or portrait).
- DisplayOrientation getDisplayOrientation () const

  Gets the current display orientation.
- void signalDMAInterrupt ()

  Notify the framework that a DMA interrupt has occurred.
- void initialize ()

  This function is responsible for initializing the entire framework.
- virtual void taskEntry ()

  Main event loop.
- virtual void flushFrameBuffer ()

  This function is called whenever the framework has performed a complete draw.
- virtual void flushFrameBuffer (const Rect &rect)

This function is called whenever the framework has performed a partial draw.

- virtual void allowDMATransfers ()

  Allow the DMA to start transfers.
- void frontPorchEntered ()

  Has to be called from within the LCD IRQ routine when the Front Porch Entry is reached.
- virtual void flushDMA ()

  This function blocks until the DMA queue (containing BlitOps) is empty.
- virtual uint16_t * lockFrameBuffer ()

  Waits for the frame buffer to become available for use.
- virtual void unlockFrameBuffer ()

  Unlocks the frame buffer (MUST be called exactly once for each call to lockFrameBuffer()).
- void lockDMAToFrontPorch (bool enableLock)

  Function to set whether the DMA transfers are locked to the TFT update cycle.
- virtual void registerTextCache (Unicode::UnicodeChar *str, uint16_t length)

  Configures HAL to use the supplied buffer as text string cache.
- virtual const Unicode::UnicodeChar * cacheTextString (const Unicode::UnicodeChar *str)

  This function can be used to cache a given string.
- virtual bool blockCopy (void *RESTRICT dest, const void *RESTRICT src, uint32_t numBytes)

  This function performs a platform-specific memcpy.
- virtual BlitOperations getBlitCaps ()

  Function for obtaining the blit capabilities of the concrete HAL implementation.
- virtual void blitSetTransparencyKey (uint16_t key)

  Deprecated function which can be ignored.
- virtual void blitCopy (const uint16_t *pSrc, uint16_t x, uint16_t y, uint16_t width, uint16_t height, uint16_t srcWidth, uint8_t alpha, bool hasTransparentPixels)

  Blits a 2D source-array to the frame buffer performing alpha-blending.
- virtual void blitCopyARGB8888 (const uint16_t *pSrc, uint16_t x, uint16_t y, uint16_t width, uint16_t height, uint16_t srcWidth, uint8_t alpha)

  Blits a 2D source-array to the frame buffer performing per-pixel alpha blending.
- virtual void blitCopyGlyph (const uint8_t *pSrc, uint16_t x, uint16_t y, uint16_t width, uint16_t height, uint16_t srcWidth, colortype color, uint8_t alpha, BlitOperations operation)

  Blits a 4bpp or 8bpp glyph - maybe use the same method and supply additional color mode arg.
- virtual void blitFill (colortype color, uint16_t x, uint16_t y, uint16_t width, uint16_t height, uint8_t alpha)

  Blits a color value to the frame buffer performing alpha-blending (and transparency keying) as specified.
- virtual void registerEventListener (UIEventListener &listener)

  Registers an event handler implementation with the underlying event system.
- virtual uint16_t * copyFBRegionToMemory (Rect meAbs)

  Copies a region of the currently displayed frame buffer to memory.
- uint16_t getDisplayWidth () const

  Gets display width.
- uint16_t getDisplayHeight () const

  Gets display height.
- void swapFrameBuffers ()

  Swaps the two frame buffers.
- uint32_t getLCDRefreshCount ()

  Returns the number of VSync interrupts.
- void setFrameRateCompensation (bool enabled)

  Enables or disables compensation for lost frames.
- void vSync ()

  Called by the VSync interrupt.

- virtual void backPorchExited ()

  Has to be called from within the LCD IRQ rutine when the Back Porch Exit is reached.
- virtual void configureInterrupts ()=0

  Configures the interrupts relevant for TouchGFX.
- virtual void enableInterrupts ()=0

  Enables the DMA and LCD interrupts.
- virtual void disableInterrupts ()=0

  Disables the DMA and LCD interrupts.
- virtual void enableLCDControllerInterrupt ()=0

  Configure the LCD controller to fire interrupts at VSYNC.
- virtual bool sampleKey (uint8_t &key)

  Sample external key event.
- void setDragThreshold (uint8_t value)

  Configure the threshold for reporting drag events.
- virtual void setFrameBufferStartAddress (void *adr, uint16_t depth=16, bool useDoubleBuffering=true, bool useAnimationStorage=true)

  Sets the address used for frame buffers, usually located in external memory.
- void setTouchSampleRate (int8_t sampleRateInTicks)

  Sets the number of ticks between each touch screen sample.
- int8_t getTouchSampleRate () const

  Gets the number of ticks between each touch screen sample.
- void setMCUActive (bool active)

  Register if MCU is active by measuring cpu cycles.
- uint32_t getCPUCycles ()

  Gets the current cycle counter.
- void setMCUInstrumentation (MCUInstrumentation *mcuInstr)

  Stores a pointer to an instance of an MCU specific instrumentation class.
- void enableMCULoadCalculation (bool enabled)

  This method sets a flag that determines if generic HAL should calculate MCU load.
- uint8_t getMCULoadPct () const

  Gets the current MCU load.
- void setButtonController (ButtonController *btnCtrl)

  Stores a pointer to an instance of a specific implementation of a ButtonController.
- ButtonController * getButtonController () const

  Gets the ButtonController.
- void setFingerSize (uint8_t size)

  Sets the finger size.
- uint8_t getFingerSize () const

  Gets the finger size.
- uint16_t * getAnimationStorage () const

  Gets the optional frame buffer used for animation storage.

Static Public Member Functions

- static HAL * getInstance ()

  Gets the HAL instance.
- static LCD & lcd ()

  Gets a reference to the LCD.

Static Public Attributes

- static uint16_t DISPLAY_WIDTH

  The width of the LCD display in pixels.
- static uint16_t DISPLAY_HEIGHT

  The height of the LCD display in pixels.
- static DisplayRotation DISPLAY_ROTATION

  The rotation from display to frame buffer.
- static uint16_t FRAME_BUFFER_WIDTH

  The width of the frame buffer in pixels.
- static uint16_t FRAME_BUFFER_HEIGHT

  The height of the frame buffer in pixels.
- static bool USE_DOUBLE_BUFFERING

  Is double buffering enabled?
- static bool USE_ANIMATION_STORAGE

  Is animation storage enabled?

Protected Member Functions

- virtual void tick ()

  This function is called at each timer tick, depending on platform implementation.
- virtual bool beginFrame ()

  Called when beginning to rendering a frame.
- virtual void endFrame ()

  Called when a rendering pass is completed.
- virtual uint16_t * getTFTFrameBuffer () const =0

  Gets the frame buffer address used by the TFT controller.
- virtual void setTFTFrameBuffer (uint16_t *address)=0

  Sets the frame buffer address used by the TFT controller.
- uint16_t * getClientFrameBuffer ()

  Gets client frame buffer.
- virtual void touch (int32_t x, int32_t y)

  Called by the touch driver to indicate a touch.
- virtual void noTouch ()

  Called by the touch driver to indicate that no touch is currently detected.
- virtual void performDisplayOrientationChange ()

  Perform the actual display orientation change.

Protected Attributes

- DMA_Interface & dma

  A reference to the DMA interface.
- LCD & lcdRef

  A reference to the LCD.
- TouchController & touchController

  A reference to the touch controller.
- MCUInstrumentation * mcuInstrumentation

  A reference to an optional MCU instrumentation.
- ButtonController * buttonController

  A reference to an optional ButtonController.

- Gestures gestures

  Class for low-level interpretation of touch events.
- DisplayOrientation nativeDisplayOrientation

  Contains the native display orientation. If desired orientation is different, apply rotation.
- uint16_t * frameBuffer0

  Pointer to the first frame buffer.
- uint16_t * frameBuffer1

  Pointer to the second frame buffer.
- uint16_t * frameBuffer2

  Pointer to the optional third frame buffer used for animation storage.
- uint8_t fingerSize

  The radius of the finger in pixels.
- bool lockDMAToPorch

  Whether or not to lock DMA transfers with TFT porch signal.
- bool frameBufferUpdatedThisFrame

  True if something was drawn in the current frame.

Static Protected Attributes

- static bool isDrawing

  True if currently in the process of rendering a screen.

Friends

- class **LCD1bpp**
- class **LCD16bpp**
- class **LCD24bpp**

18.60.1 Detailed Description

Contains functions that are specific to the hardware platform the code is running on.

18.60.2 Constructor & Destructor Documentation

18.60.2.1 HAL (DMA_Interface & *dmaInterface*, LCD & *display*, TouchController & *touchCtrl*, uint16_t *width*, uint16_t *height*) [inline]

Creates a HAL instance.

Parameters

in	*dmaInterface*	Reference to the DMA interface.
in	*display*	Reference to the LCD.
in	*touchCtrl*	Reference to the touch controller.
	width	The width of the LCD display, in pixels.
	height	The height of the LCD display, in pixels.

18.60.2.2 ~HAL ( ) [inline],[virtual]

Destructor.

18.60.3 Member Function Documentation

18.60.3.1 void allowDMATransfers ( ) [virtual]

Allow the DMA to start transfers. Front Porch Entry is a good place to call this.

18.60.3.2 void backPorchExited ( ) [inline],[virtual]

Has to be called from within the LCD IRQ rutine when the Back Porch Exit is reached.

18.60.3.3 bool beginFrame ( ) [protected],[virtual]

Called when beginning to rendering a frame.

Returns

true if rendering can begin, false otherwise.

18.60.3.4 void blitCopy (const uint16_t * *pSrc*, uint16_t *x*, uint16_t *y*, uint16_t *width*, uint16_t *height*, uint16_t *srcWidth*, uint8_t *alpha*, bool *hasTransparentPixels*) [virtual]

Blits a 2D source-array to the frame buffer performing alpha-blending(and transparency keying) as specified.

Note

Alpha=255 is assumed "solid" and shall be used if HAL does not support BLIT_OP_COPY_WITH_ALPHA.

Parameters

pSrc	The source-array pointer (points to first value to copy)
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
srcWidth	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
alpha	The alpha value to use for blending (255 = solid, no blending)
has↩ Transparent↩ Pixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

18.60.3.5 void blitCopyARGB8888 (const uint16_t * *pSrc*, uint16_t *x*, uint16_t *y*, uint16_t *width*, uint16_t *height*, uint16_t *srcWidth*, uint8_t *alpha*) [virtual]

Blits a 2D source-array to the frame buffer performing per-pixel alpha blending.

Parameters

pSrc	The source-array pointer (points to first value to copy)
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.

width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
srcWidth	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
alpha	The alpha value to use for blending. This is applied on every pixel, in addition to the per-pixel alpha value (255 = solid, no blending)

18.60.3.6 void blitCopyGlyph (const uint8_t * *pSrc*, uint16_t *x*, uint16_t *y*, uint16_t *width*, uint16_t *height*, uint16_t *srcWidth*, colortype *color*, uint8_t *alpha*, BlitOperations *operation*) [virtual]

Blits a 4bpp or 8bpp glyph - maybe use the same method and supply additional color mode arg.

Parameters

pSrc	The source-array pointer (points to first value to copy)
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
srcWidth	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
color	Color of the text.
alpha	The alpha value to use for blending (255 = solid, no blending)
operation	The operation type to use for blit copy.

18.60.3.7 void blitFill (colortype *color*, uint16_t *x*, uint16_t *y*, uint16_t *width*, uint16_t *height*, uint8_t *alpha*) [virtual]

Blits a color value to the frame buffer performing alpha-blending (and transparency keying) as specified.

Note

Alpha=255 is assumed "solid" and shall be used if HAL does not support BLIT_OP_FILL_WITH_ALPHA.

Parameters

color	The desired fill-color.
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
alpha	The alpha value to use for blending (255 = solid, no blending)

18.60.3.8 void blitSetTransparencyKey (uint16_t *key*) [virtual]

Only present for backwards compatibility in TouchGFX 4.x. Will be removed in TouchGFX 5.

Parameters

key	The "transparent" color value.

Reimplemented in HALSDL.

18.60.3.9 bool blockCopy (void *RESTRICT *dest*, const void *RESTRICT *src*, uint32_t *numBytes*) [virtual]

This function performs a platform-specific memcpy, if supported by the hardware.

Parameters

out	*dest*	Pointer to destination memory.
in	*src*	Pointer to source memory.
	numBytes	Number of bytes to copy.

Returns

true if the copy succeeded, false if copy was not performed.

Reimplemented in HALSDL.

18.60.3.10 const Unicode::UnicodeChar * cacheTextString (const Unicode::UnicodeChar * *str*) [virtual]

This function can be used to cache a given string in a platform specific way to e.g. speed up access or in case the string is placed in a memory type that does not support random access such as NAND flash.

Parameters

str	A pointer to the string which may be in external memory.

Returns

A pointer to an identical string which is guaranteed to be directly readable (ie. a copy if the original string was placed in NAND flash).

18.60.3.11 void configureInterrupts ( ) [pure virtual]

Configures the interrupts relevant for TouchGFX. This primarily entails setting the interrupt priorities for the DMA and LCD interrupts.

Implemented in HALSDL.

18.60.3.12 uint16_t * copyFBRegionToMemory (Rect *meAbs*) [virtual]

Copies a region of the currently displayed frame buffer to memory. Used for e.g. SlideTransition and for displaying pre-rendered drawables e.g. in animations where redrawing the drawable is not necessary.

Note

Requires animation storage to be present.

Parameters

meAbs	The frame buffer region to copy.

Returns

A pointer to the memory address containing the copy of the frame buffer.

18.60.3.13 void disableInterrupts ( ) [pure virtual]

Disables the DMA and LCD interrupts.

Implemented in HALSDL.

18.60.3.14 void enableInterrupts ( ) [pure virtual]

Enables the DMA and LCD interrupts.

Implemented in HALSDL.

18.60.3.15 void enableLCDControllerInterrupt ( ) [pure virtual]

Configure the LCD controller to fire interrupts at VSYNC. Called automatically once TouchGFX initialization has completed.

Implemented in HALSDL.

18.60.3.16 void enableMCULoadCalculation (bool *enabled*) [inline]

This method sets a flag that determines if generic HAL should calculate MCU load based on concrete MCU instrumentation.

Parameters

enabled	If true, set flag to update MCU load.

18.60.3.17 void endFrame ( ) [protected],[virtual]

Called when a rendering pass is completed.

18.60.3.18 void flushDMA ( ) [virtual]

This function blocks until the DMA queue (containing BlitOps) is empty.

18.60.3.19 void flushFrameBuffer ( ) [virtual]

On some platforms, a local frame buffer needs to be pushed to the display through a SPI channel or similar. Implement that functionality here. This function is called whenever the framework has performed a complete draw.

Reimplemented in HALSDL.

18.60.3.20 void flushFrameBuffer (const Rect & *rect*) [virtual]

This function is called whenever the framework has performed a partial draw.

Parameters

rect	The area of the screen that has been drawn, expressed in absolute coordinates.

See also

flushFrameBuffer().

Reimplemented in HALSDL.

18.60.3.21 void frontPorchEntered ( ) [inline]

Has to be called from within the LCD IRQ routine when the Front Porch Entry is reached.

18.60.3.22 uint16_t * getAnimationStorage ( ) const [inline]

Gets the optional frame buffer used for animation storage.

Returns

The address or 0 if unused.

18.60.3.23 BlitOperations getBlitCaps ( ) [inline], [virtual]

Function for obtaining the blit capabilities of the concrete HAL implementation. As default, will return whatever blitcaps are reported by the associated DMA object.

Returns

a bitmask of the supported blitcaps.

18.60.3.24 ButtonController * getButtonController ( ) const [inline]

Gets the associated ButtonController.

Returns

A pointer to the ButtonController, or zero if no ButtonController has been set.

18.60.3.25 uint16_t * getClientFrameBuffer ( ) [inline], [protected]

Gets client frame buffer.

Returns

The address of the framebuffer currently used by the framework to draw in.

18.60.3.26 uint32_t getCPUCycles ( )

Gets the current cycle counter.

Returns

the cycle counter.

18.60.3.27 uint16_t getDisplayHeight ( ) const [inline]

Gets display height.

Returns

The display height.

18.60.3.28 DisplayOrientation getDisplayOrientation ( ) const [inline]

Gets the current display orientation. Will be equal to the native orientation of the display unless setDisplayOrientation has been explicitly called earlier.

Returns

The current display orientation.

18.60.3.29 uint16_t getDisplayWidth ( ) const [inline]

Gets display width.

Returns

The display width.

18.60.3.30 uint8_t getFingerSize ( ) const [inline]

Gets the finger size in pixels.

Returns

The size of the finger in pixels, 1 is the default value.

18.60.3.31 static HAL * getInstance ( ) [inline], [static]

Gets the HAL instance.

Returns

The HAL instance.

18.60.3.32 uint32_t getLCDRefreshCount ( ) [inline]

Returns the number of VSync interrupts between the current drawing operation and the last drawing operation, i.e. the number of lost frames.

Returns

Number of VSync since previous draw.

18.60.3.33 uint8_t getMCULoadPct ( ) const [inline]

Gets the current MCU load.

Returns

mcuLoadPct the MCU Load in %.

18.60.3.34 uint16_t * getTFTFrameBuffer ( ) const [protected], [pure virtual]

Gets the frame buffer address used by the TFT controller.

Returns

The address of the frame buffer currently being displayed on the TFT.

Implemented in HALSDL.

18.60.3.35 int8_t getTouchSampleRate ( ) const [inline]

Gets the number of ticks between each touch screen sample.

Returns

Number of ticks between each touch screen sample.

18.60.3.36 void initialize ( )

This function is responsible for initializing the entire framework.

18.60.3.37 static LCD & lcd ( ) [inline], [static]

Gets a reference to the LCD.

Returns

A reference to the LCD.

18.60.3.38 void lockDMAToFrontPorch (bool *enableLock*) [inline]

Function to set whether the DMA transfers are locked to the TFT update cycle. If locked, DMA transfer will not begin until the TFT controller has finished updating the display. If not locked, DMA transfers will begin as soon as possible. Default is true (DMA is locked with TFT).

Disabling the lock will in most cases significantly increase rendering performance. It is therefore strongly recommended to disable it. Depending on platform this may in rare cases cause rendering problems (visible tearing on display). Please see the chapter "Optimizing DMA During TFT Controller Access" for details on this setting.

Note

This setting only has effect when using double buffering.

Parameters

enableLock	True to lock DMA transfers to the front porch signal. Conservative, default setting. False to disable, which will normally yield substantial performance improvement.

18.60.3.39 uint16_t * lockFrameBuffer ( ) [virtual]

Waits for the frame buffer to become available for use (i.e. not used by DMA transfers).

Note

Function blocks until frame buffer is available. Client code MUST call unlockFrameBuffer() when frame buffer operation has completed.

Returns

A pointer to the beginning of the currently used frame buffer.

18.60.3.40 void noTouch ( ) [protected],[virtual]

Called by the touch driver to indicate that no touch is currently detected.

18.60.3.41 void performDisplayOrientationChange ( ) [inline],[protected],[virtual]

Perform the actual display orientation change.

Reimplemented in HALSDL.

18.60.3.42 void registerEventListener (UIEventListener & *listener*) [virtual]

Registers an event handler implementation with the underlying event system. The actual HAL implementation decides whether or not multiple UIEventListener instances are allowed (including execution order).

Parameters

in	*listener*	The listener to register.

18.60.3.43 void registerTextCache (Unicode::UnicodeChar * *str*, uint16_t *length*) [virtual]

Configures HAL to use the supplied buffer as text string cache. The buffer must be large enough to hold the longest string in the system. Setting this buffer is only required if cacheTextString() is actually used and its implementation requires a buffer.

Parameters

in	*str*	Pointer to buffer location.
	length	Buffer length (in UnicodeChar's)

See also

cacheTextString

18.60.3.44 bool sampleKey (uint8_t & *key*) [inline],[virtual]

Sample external key event.

Parameters

out	*key*	Output parameter that will be set to the key value if a keypress was detected.

Returns

True if a keypress was detected and the "key" parameter is set to a value.

Reimplemented in HALSDL.

18.60.3.45 void setButtonController (ButtonController * *btnCtrl*) [inline]

Stores a pointer to an instance of a specific implementation of a ButtonController.

Parameters

in	*btnCtrl*	pointer to button controller.

18.60.3.46 void setDisplayOrientation (DisplayOrientation *orientation*) [inline],[virtual]

Sets the desired display orientation (landscape or portrait). If desired orientation is different from the native orientation of the display, a rotation is automatically applied. The rotation does not incur any performance cost.

Note

A screen transition must occur before this takes effect!

Parameters

orientation	The desired display orientation.

18.60.3.47 void setDragThreshold (uint8_t *value*) [inline]

Configure the threshold for reporting drag events. A touch input movement must exceed this value in either axis in order to report a drag. Default value is 0.

Note

Use if touch controller is not completely accurate to avoid "false" drags.

Parameters

value	New threshold value.

18.60.3.48 void setFingerSize (uint8_t *size*) [inline]

Sets the finger size in pixels.

Setting the finger size to a size of more than 1 pixel will emulate a finger of width and height of 2∗(fingersize-1)+1. This can be especially useful when trying to interact with small elements on a high ppi display. The finger size will influence which element is chosen as the point of interaction, when clicking, dragging, ... the display. A number of samples will be drawn from within the finger area and a best matching drawable will be chosen. The best matching algorithm will consider the size of the drawable and the distance from the touch point.

Parameters

in	*size*	the size of the finger.

18.60.3.49 void setFrameBufferStartAddress (void ∗ *adr*, uint16_t *depth* = 16, bool *useDoubleBuffering* = true, bool *useAnimationStorage* = true) [inline],[virtual]

Sets the address used for frame buffers, usually located in external memory. Will reserve memory for one or two frame buffers based on display size. Will optionally also reserve memory for a third frame buffer used for animation↩ Storage.

Parameters

in	*adr*	Starting address to use for frame buffers.
	depth	Depth of each pixel in bits, default is 16.
	useDouble↩ Buffering	If true, reserve memory for an extra frame buffer.
	useAnimation↩ Storage	If true, reserve memory for animation storage.

18.60.3.50 void setFrameRateCompensation (bool *enabled*) [inline]

Enables or disables compensation for lost frames. See knowledge base article.

Parameters

enabled	true to enable, false to disable.

18.60.3.51 void setMCUActive (bool *active*)

Register if MCU is active by measuring cpu cycles. If user wishes to track MCU load, this method should be called whenever the OS Idle task is scheduled in or out. This method makes calls to a concrete implementation of GPIO functionality and a concrete implementation of cpu cycles.

Parameters

active	If true, MCU is registered as being active, inactive otherwise.

18.60.3.52 void setMCUInstrumentation (MCUInstrumentation * *mcuInstr*) [inline]

Stores a pointer to an instance of an MCU specific instrumentation class.

Parameters

in	*mcuInstr*	pointer to MCU instrumentation.

18.60.3.53 void setTFTFrameBuffer (uint16_t * *address*) [protected],[pure virtual]

Sets the frame buffer address used by the TFT controller.

Parameters

in	*address*	New frame buffer address.

Implemented in HALSDL.

18.60.3.54 void setTouchSampleRate (int8_t *sampleRateInTicks*) [inline]

Sets the number of ticks between each touch screen sample.

Parameters

sampleRateIn↩ Ticks	Sample rate. Default is 1 (every tick).

18.60.3.55 void signalDMAInterrupt ( ) [inline]

Notify the framework that a DMA interrupt has occurred.

18.60.3.56 void swapFrameBuffers ( )

Swaps the two frame buffers.

18.60.3.57 void taskEntry ( ) [virtual]

Main event loop. Will wait for VSYNC signal, and then process next frame. Call this function from your GUI task.

Note

This function never returns!

Reimplemented in HALSDL.

18.60.3.58 void tick ( ) [protected], [virtual]

This function is called at each timer tick, depending on platform implementation.

18.60.3.59 void touch (int32_t *x*, int32_t *y*) [protected], [virtual]

Called by the touch driver to indicate a touch.

Parameters

x	The x coordinate of the touch.
y	The y coordinate of the touch.

18.60.3.60 void unlockFrameBuffer ( ) [virtual]

Unlocks the frame buffer (MUST be called exactly once for each call to lockFrameBuffer()).

18.60.3.61 void vSync ( ) [inline]

Called by the VSync interrupt for counting of LCD refreshes.

18.61 HALSDL Class Reference

HAL implementation for the TouchGFX simulator.

```
#include <platform/hal/simulator/sdl/HALSDL.hpp>
```

Public Member Functions

- HALSDL (DMA_Interface &dma, LCD &lcd, TouchController &touchCtrl, uint16_t width, uint16_t height)

  Constructor. Initializes members.
- virtual void taskEntry ()

  Main event loop.
- virtual bool sampleKey (uint8_t &key)

  Sample key event from keyboard.
- virtual void flushFrameBuffer ()

  This function is called whenever the framework has performed a complete draw.

- virtual void flushFrameBuffer (const Rect &rect)

  This function is called whenever the framework has performed a partial draw.
- virtual bool blockCopy (void ∗RESTRICT dest, const void ∗RESTRICT src, uint32_t numBytes)

  This function performs a platform-specific memcpy.
- virtual void blitSetTransparencyKey (uint16_t key)

  If Blit-operations are supported, transparency-keying support is implicitly assumed.
- bool doSampleTouch (int32_t &x, int32_t &y)

  Samples the position of the mouse cursor.
- virtual bool sdl_init (int argcount, char ∗∗args)

  Initializes SDL.

Protected Member Functions

- virtual uint16_t ∗ getTFTFrameBuffer () const

  Gets TFT frame buffer.
- void setTFTFrameBuffer (uint16_t ∗addr)

  Sets TFT frame buffer.
- virtual void renderLCD_FrameBufferToMemory (const Rect &rectToUpdate, uint16_t ∗frameBuffer)

  Update frame buffer using an SDL Surface.
- virtual void saveScreenshot (char ∗folder, char ∗filename)

  Saves a screen shot.
- virtual void disableInterrupts ()

  Disables the DMA and LCD interrupts.
- virtual void enableInterrupts ()

  Enables the DMA and LCD interrupts.
- virtual void configureLCDInterrupt ()

  Configures LCD interrupt.
- virtual void enableLCDControllerInterrupt ()

  Enables the LCD interrupt.
- virtual void configureInterrupts ()

  Configures the interrupts relevant for TouchGFX.
- void performDisplayOrientationChange ()

  Perform the actual display orientation change.

Protected Attributes

- Rect screenArea

  Screen area rectangle.

Additional Inherited Members

18.61.1 Detailed Description

HAL implementation for the TouchGFX simulator.

See also

HAL

18.61.2 Constructor & Destructor Documentation

18.61.2.1 HALSDL (DMA_Interface & *dma,* LCD & *lcd,* TouchController & *touchCtrl,* uint16_t *width,* uint16_t *height*) [inline]

Constructor. Initializes members.

Parameters

in	*dma*	Reference to DMA interface.
in	*lcd*	Reference to the LCD.
in	*touchCtrl*	Reference to Touch Controller driver.
	width	Width of the display.
	height	Height of the display.

18.61.3 Member Function Documentation

18.61.3.1 void blitSetTransparencyKey (uint16_t *key*) [virtual]

If Blit-operations are supported, transparency-keying support is implicitly assumed.

Parameters

key	The "transparent" color value.

Reimplemented from HAL.

18.61.3.2 bool blockCopy (void *RESTRICT *dest*, const void *RESTRICT *src*, uint32_t *numBytes*) [virtual]

This function performs a platform-specific memcpy, if supported by the hardware.

Parameters

out	*dest*	Pointer to destination memory.
	src	Pointer to source memory.
	numBytes	Number of bytes to copy.

Returns

true if the copy succeeded, false if copy was not performed.

Reimplemented from HAL.

18.61.3.3 void configureInterrupts ( ) [inline], [protected], [virtual]

Configures the interrupts relevant for TouchGFX. This primarily entails setting the interrupt priorities for the DMA and LCD interrupts.

Implements HAL.

18.61.3.4 void configureLCDInterrupt ( ) [inline], [protected], [virtual]

Configures LCD interrupt.

18.61.3.5 void disableInterrupts ( ) [inline], [protected], [virtual]

Disables the DMA and LCD interrupts.

Implements HAL.

18.61.3.6 bool doSampleTouch (int32_t & *x*, int32_t & *y*)

Samples the position of the mouse cursor.

Parameters

out	*x*	The x coordinate.
out	*y*	The y coordinate.

Returns

True if touch detected, false otherwise.

18.61.3.7 void enableInterrupts ( ) [inline], [protected], [virtual]

Enables the DMA and LCD interrupts.

Implements HAL.

18.61.3.8 void enableLCDControllerInterrupt ( ) [inline], [protected], [virtual]

Enables the LCD interrupt.

Implements HAL.

18.61.3.9 void flushFrameBuffer ( ) [virtual]

On some platforms, a local frame buffer needs to be pushed to the display through a SPI channel or similar. Implement that functionality here. This function is called whenever the framework has performed a complete draw.

Reimplemented from HAL.

18.61.3.10 void flushFrameBuffer (const Rect & *rect*) [virtual]

This function is called whenever the framework has performed a partial draw.

Parameters

rect	The area of the screen that has been drawn, expressed in absolute coordinates.

See also

flushFrameBuffer(). This function is called whenever the framework has performed a partial draw.

Reimplemented from HAL.

18.61.3.11 uint16_t * getTFTFrameBuffer ( ) const [protected], [virtual]

Gets TFT frame buffer.

Returns

null if it fails, else the TFT frame buffer.

Implements HAL.

18.61.3.12 void performDisplayOrientationChange ( ) [protected], [virtual]

Perform the actual display orientation change.

Reimplemented from HAL.

18.61.3.13 void renderLCD_FrameBufferToMemory (const Rect & *rectToUpdate,* uint16_t * *frameBuffer*) [protected],[virtual]

Update frame buffer using an SDL Surface.

Parameters

	rectToUpdate	Area to update.
in	*frameBuffer*	Target frame buffer.

18.61.3.14 bool sampleKey (uint8_t & *key*) [virtual]

Sample key event from keyboard.

Parameters

out	*key*	Output parameter that will be set to the key value if a key press was detected.

Returns

True if a key press was detected and the "key" parameter is set to a value.

Reimplemented from HAL.

18.61.3.15 void saveScreenshot (char * *folder*, char * *filename*) [protected], [virtual]

Saves a screen shot.

Parameters

in	*folder*	Folder name to place the screen shot in.
in	*filename*	Filename to save the screen shot to.

18.61.3.16 bool sdl_init (int *argcount*, char ** *args*) [virtual]

Initializes SDL.

Parameters

	argcount	Number of arguments.
in	*args*	Arguments.

Returns

True if init went well, false otherwise.

18.61.3.17 void setTFTFrameBuffer (uint16_t * *addr*) [protected], [virtual]

Sets TFT frame buffer.

Parameters

in	*addr*	The address of the TFT frame buffer.

Implements HAL.

18.61.3.18 void taskEntry ( ) [virtual]

Main event loop. Will wait for VSYNC signal, and then process next frame. Call this function from your GUI task.

Note

This function never returns!

Reimplemented from HAL.

18.62 I2C Class Reference

Platform independent interface for I2C drivers.

`#include <platform/driver/i2c/I2C.hpp>`

Public Member Functions

- I2C (uint8_t ch)

  Stores the channel of the I2C bus to be configured.
- virtual ~I2C ()

  Destructor.
- virtual void init ()=0

  Initializes the I2C driver.
- virtual bool readRegister (uint8_t addr, uint8_t reg, uint8_t *data, uint32_t cnt)=0

  Reads the specified register on the device with the specified address.
- virtual bool writeRegister (uint8_t addr, uint8_t reg, uint8_t val)=0

  Writes the specified value in a register.

Protected Attributes

- uint8_t channel

  I2c channel is stored in order to initialize and recover a specific I2C channel.

18.62.1 Detailed Description

Platform independent interface for I2C drivers.

18.62.2 Constructor & Destructor Documentation

18.62.2.1 I2C (uint8_t *ch*) [inline]

Stores the channel of the I2C bus to be configured.

Parameters

ch	I2C channel.

18.62.2.2 ~I2C ( ) [inline],[virtual]

Destructor.

18.62.3 Member Function Documentation

18.62.3.1 void init ( ) [pure virtual]

Initializes the I2C driver.

Implemented in I2C_LPC43xx, and I2C_LPC1788.

18.62.3.2 bool readRegister (uint8_t *addr*, uint8_t *reg*, uint8_t * *data*, uint32_t *cnt*) [pure virtual]

Reads the specified register on the device with the specified address.

Parameters

	addr	The I2C device address.
	reg	The register.
out	*data*	Pointer to buffer in which to place the result.
	cnt	Size of buffer in bytes.

Returns

true on success, false otherwise.

Implemented in I2C_LPC43xx, and I2C_LPC1788.

18.62.3.3 bool writeRegister (uint8_t *addr*, uint8_t *reg*, uint8_t *val*) [pure virtual]

Writes the specified value in a register.

Parameters

addr	The I2C device address.
reg	The register.
val	The new value.

Returns

true on success, false otherwise.

Implemented in I2C_LPC43xx, and I2C_LPC1788.

18.63 I2C_LPC1788 Class Reference

```
#include <I2C_LPC1788.hpp>
```

Public Member Functions

- I2C_LPC1788 (uint8_t ch)
- void init ()
- bool readRegister (uint8_t addr, uint8_t reg, uint8_t *data, uint32_t cnt)
- bool writeRegister (uint8_t addr, uint8_t reg, uint8_t val)

Additional Inherited Members

18.63.1 Detailed Description

NXP LPC1788 MCU specific I2C driver.

18.63.2 Constructor & Destructor Documentation

18.63.2.1 I2C_LPC1788 (uint8_t *ch*) [inline]

Constructor. Calls super constructor with channel parameter.

Parameters

ch	I2C channel

18.63.3 Member Function Documentation

18.63.3.1 void init ( ) [virtual]

Initializes the specified channel of the I2C bus Sets up SDA and SDL and intialize I2C peripheral.

Implements I2C.

18.63.3.2 bool readRegister (uint8_t *addr*, uint8_t *reg*, uint8_t * *data*, uint32_t *cnt*) [virtual]

Reads the specificed register on the device with the specified address.

Parameters

addr	The 7-bit I2C device address
reg	The register
data	Pointer to buffer in which to place the result
cnt	Size of buffer in bytes

Returns

true on success, false otherwise.

Implements I2C.

18.63.3.3 bool writeRegister (uint8_t *addr*, uint8_t *reg*, uint8_t *val*) [virtual]

Writes the specified value in a register.

Parameters

addr	The I2C device address
reg	The register
val	The new value

Returns

true on success, false otherwise.

Implements I2C.

18.64 I2C_LPC43xx Class Reference

```
#include <I2C_LPC43xx.hpp>
```

Public Member Functions

- I2C_LPC43xx (uint8_t ch)
- virtual void init ()
- virtual bool readRegister (uint8_t addr, uint8_t reg, uint8_t *data, uint32_t cnt)
- virtual bool writeRegister (uint8_t addr, uint8_t reg, uint8_t val)

Additional Inherited Members

18.64.1 Detailed Description

NXP LPC43XX MCU specific I2C driver.

18.64.2 Constructor & Destructor Documentation

18.64.2.1 I2C_LPC43xx (uint8_t *ch*) [inline]

Constructor. Calls super constructor with channel parameter.

Parameters

ch	I2C channel

18.64.3 Member Function Documentation

18.64.3.1 virtual void init ( ) [virtual]

Initializes the I2C driver. Sets up SDA and SDL and intialize I2C peripheral.

Implements I2C.

18.64.3.2 virtual bool readRegister (uint8_t *addr*, uint8_t *reg*, uint8_t * *data*, uint32_t *cnt*) [virtual]

Reads the specificed register on the device with the specified address.

Parameters

addr	The 7-bit I2C device address
reg	The register
data	Pointer to buffer in which to place the result
cnt	Size of buffer in bytes

Returns

true on success, false otherwise.

Implements I2C.

18.64.3.3 virtual bool writeRegister (uint8_t *addr*, uint8_t *reg*, uint8_t *val*) [virtual]

Writes the specified value in a register.

Parameters

addr	The I2C device address
reg	The register
val	The new value

Returns

true on success, false otherwise.

Implements I2C.

18.65 I2CTouchController Class Reference

Specific I2C-enabled type of Touch Controller.

```
#include <platform/driver/touch/I2CTouchController.hpp>
```

Public Member Functions

- I2CTouchController (I2C &i2c)

  Constructor.
- virtual ~I2CTouchController ()

  Destructor.
- virtual void init ()=0

  Initializes touch controller.
- virtual bool sampleTouch (int32_t &x, int32_t &y)=0

  Checks whether the touch screen is being touched.

Protected Attributes

- I2C & i2c

  I2C driver.

18.65.1 Detailed Description

Specific I2C-enabled type of Touch Controller.

See also

TouchController

18.65.2 Constructor & Destructor Documentation

18.65.2.1 I2CTouchController (I2C & *i2c*) [inline]

Constructor. Initializes I2C driver.

Parameters

in,out	*i2c*	I2C driver.

18.65.2.2 ~I2CTouchController ( ) [inline],[virtual]

Destructor.

18.65.3 Member Function Documentation

18.65.3.1 void init ( ) [pure virtual]

Initializes touch controller.

Implements TouchController.

18.65.3.2 bool sampleTouch (int32_t & *x*, int32_t & *y*) [pure virtual]

Checks whether the touch screen is being touched, and if so, what coordinates.

Parameters

out	*x*	The x position of the touch
out	*y*	The y position of the touch

Returns

True if a touch has been detected, otherwise false.

Implements TouchController.

18.66 Image Class Reference

Simple widget capable of showing a bitmap.

```
#include <touchgfx/widgets/Image.hpp>
```

Public Member Functions

- Image (const Bitmap &bmp=Bitmap())
  Default Constructor.
- void setBitmap (const Bitmap &bmp)
  Sets the bitmap ID for this Image.
- void setAlpha (uint8_t alpha)
  Sets the alpha channel for the image.
- virtual void draw (const Rect &invalidatedArea) const
  Draws the image.
- BitmapId getBitmap () const
  Gets the BitmapId currently contained by the widget.
- uint8_t getAlpha () const
  Gets the current alpha value.
- virtual Rect getSolidRect () const
  Gets the largest solid (non-transparent) rectangle.
- virtual uint16_t getType () const
  For GUI testing only.

Protected Attributes

- Bitmap bitmap

  The Bitmap to display.
- uint8_t alpha

  The Alpha for this image.
- bool hasTransparentPixels

  true if this object has transparent pixels

Additional Inherited Members

18.66.1 Detailed Description

Simple widget capable of showing a bitmap. The bitmap can be alpha-blended with the background and have areas of transparency (LCD::TRANSPARENT_COL).

See also

Widget

18.66.2 Constructor & Destructor Documentation

18.66.2.1 Image (const Bitmap & *bmp* = Bitmap ()) [inline]

Constructs a new Image with a default alpha value of 255 (solid) and a default Bitmap if none is specified.

Parameters

bmp	The bitmap to display.

18.66.3 Member Function Documentation

18.66.3.1 void draw (const Rect & *invalidatedArea*) const [virtual]

Draws the image. This class supports partial drawing, so only the area described by the rectangle will be drawn.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Implements Drawable.

18.66.3.2 uint8_t getAlpha ( ) const [inline]

Gets the current alpha value.

Returns

The current alpha value.

18.66.3.3 BitmapId getBitmap ( ) const [inline]

Gets the BitmapId currently contained by the widget.

Returns

The current BitmapId of the widget.

18.66.3.4 Rect getSolidRect ( ) const [virtual]

Gets the largest solid (non-transparent) rectangle. This value is pre-calculated by the imageconverter tool.

Returns

The largest solid (non-transparent) rectangle.

Implements Drawable.

18.66.3.5 uint16_t getType ( ) const [inline],[virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_IMAGE.

Reimplemented from Widget.

Reimplemented in AnimatedImage.

18.66.3.6 void setAlpha (uint8_t *alpha*) [inline]

Sets the alpha channel for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

18.66.3.7 void setBitmap (const Bitmap & *bmp*)

Sets the bitmap ID for this Image. Updates the width and height of this widget to match that of the bitmap.

Parameters

bmp	The bitmap instance.

See also

Bitmap

18.67 InternalFlashFont Class Reference

An InternalFlashFont has both glyph table and glyph data placed in a flash which supports random access read (i.e. not a NAND flash).

```
#include <touchgfx/InternalFlashFont.hpp>
```

Public Member Functions

- InternalFlashFont (const GlyphNode *list, uint16_t size, uint16_t height, uint8_t pixBelowBase, uint8_t bitsPerPixel, uint8_t maxLeft, uint8_t maxRight, const uint8_t *glyphDataInternalFlash, const KerningNode *kerningList)

  Constructor.
- virtual const uint8_t * getPixelData (const GlyphNode *glyph) const

  Obtains a RAM-based pointer to the pixel data for the specified glyph.
- virtual int8_t getKerning (Unicode::UnicodeChar prevChar, const GlyphNode *glyph) const

  Gets the kerning distance between two characters.

Additional Inherited Members

18.67.1 Detailed Description

An InternalFlashFont has both glyph table and glyph data placed in a flash which supports random access read (i.e. not a NAND flash)

See also

ConstFont

18.67.2 Constructor & Destructor Documentation

18.67.2.1 InternalFlashFont (const GlyphNode * *list*, uint16_t *size*, uint16_t *height*, uint8_t *pixBelowBase*, uint8_t *bitsPerPixel*, uint8_t *maxLeft*, uint8_t *maxRight*, const uint8_t * *glyphDataInternalFlash*, const KerningNode * *kerningList*)

Construct the InternalFlashFont.

Parameters

list	The array of glyphs known to this font.
size	The number of glyphs in list.
height	The height in pixels of the highest character in this font.
pixBelowBase	The maximum number of pixels that can be drawn below the baseline in this font.
bitsPerPixel	The number of bits per pixel in this font.
maxLeft	The maximum a character extends to the left.
maxRight	The maximum a character extends to the right.
glyphDataInternalFlash	Pointer to the glyph data for the font, placed in internal flash.
kerningList	pointer to the kerning data for the font, placed in internal flash.

18.67.3 Member Function Documentation

18.67.3.1 int8_t getKerning (Unicode::UnicodeChar *prevChar*, const GlyphNode * *glyph*) const [virtual]

Gets the kerning distance between two characters.

Parameters

prevChar	The unicode value of the previous character.

glyph	the glyph object for the current character.

Returns

The kerning distance between prevChar and glyph char.

Implements ConstFont.

18.67.3.2 const uint8_t * getPixelData (const GlyphNode * *glyph*) const [virtual]

Obtains a RAM-based pointer to the pixel data for the specified glyph.

Parameters

glyph	The glyph to get the pixels data of.

Returns

The pixel data of the glyph.

Implements ConstFont.

18.68 Scanline::iterator Class Reference

An iterator to help go through all the elements that make up a Scanline.

```
#include <touchgfx/canvas_widget_renderer/Scanline.hpp>
```

Public Member Functions

- iterator (const Scanline &scanline)

  Constructor.
- int next ()

  Gets the next element on the Scanline.
- int getNumPix () const

  Gets number of consecutive pixels in the current run on the Scanline.
- const uint8_t * getCovers () const

  Gets the covers in the current run on the Scanline.

18.68.1 Detailed Description

An iterator to help go through all the elements that make up a Scanline. Each part of the Scanline has a different Cover.

18.68.2 Constructor & Destructor Documentation

18.68.2.1 iterator (const Scanline & *scanline*) [inline]

Constructor. Creates an iterator to help go through all the Scanline parts of the polygon on a single Scanline.

Parameters

scanline	The scanline to iterate.

18.68.3 Member Function Documentation

18.68.3.1 const uint8_t * getCovers ( ) const [inline]

Gets the covers in the current run on the Scanline.

Returns

array of covers of each individual pixel.

18.68.3.2 int getNumPix ( ) const [inline]

Gets number of consecutive pixels in the current run on the Scanline.

Returns

The number of consecutive pixels.

18.68.3.3 int next ( ) [inline]

Gets the next element on the Scanline.

Returns

An the next index in the array of Scanline elements.

18.69 JSMOCHelper Class Reference

Helper class providing caching of certain information while the JSMOC algorithm runs during draw operations.

`#include <touchgfx/JSMOCHelper.hpp>`

Public Member Functions

- JSMOCHelper ()

  Default constructor.
- void setWidget (Drawable *newWidget)

  Sets a widget.
- Drawable * getWidget ()

  Gets the widget.
- Rect & getCachedVisibleRect ()

  Gets the visible rect for the widget of this helper.
- int16_t getCachedAbsX ()

  Gets the absolute x coordinate for the widget of this helper.
- int16_t getCachedAbsY ()

  Gets the absolute y coordinate for the widget of this helper.
- int16_t getWidth ()

  Gets the width of the widget of this helper.

- int16_t getHeight ()

  Gets the height of the widget of this helper.
- void draw (const Rect &invalidatedArea)

  Draws the widget of this helper.

18.69.1 Detailed Description

Helper class providing caching of certain information while the JSMOC algorithm runs during draw operations. Not intented for application-level use.

18.69.2 Constructor & Destructor Documentation

18.69.2.1 JSMOCHelper ( ) [inline]

Default constructor.

18.69.3 Member Function Documentation

18.69.3.1 void draw (const Rect & *invalidatedArea*) [inline]

Draws the widget of this helper.

Parameters

invalidatedArea	The area of the widget to draw.

18.69.3.2 int16_t getCachedAbsX ( ) [inline]

Gets the absolute x coordinate for the widget of this helper.

Returns

The absolute x coordinate for the widget of this helper. Only calculated once.

18.69.3.3 int16_t getCachedAbsY ( ) [inline]

Gets the absolute y coordinate for the widget of this helper.

Returns

The absolute y coordinate for the widget of this helper. Only calculated once.

18.69.3.4 Rect & getCachedVisibleRect ( ) [inline]

Gets the visible rect for the widget of this helper.

Returns

The visible rect for the widget of this helper. Only calculated once.

18.69.3.5 int16_t getHeight ( ) [inline]

Gets the height of the widget of this helper.

Returns

The height of the widget of this helper.

18.69.3.6 Drawable * getWidget ( ) [inline]

Gets the widget.

Returns

The widget this helper operates on.

18.69.3.7 int16_t getWidth ( ) [inline]

Gets the width of the widget of this helper.

Returns

The width of the widget of this helper.

18.69.3.8 void setWidget (Drawable * *newWidget*) [inline]

Sets a widget.

Parameters

in	*newWidget*	The widget to operate on.

18.70 KerningNode Struct Reference

Structure providing information about a kerning for a given char pair.

`#include <touchgfx/Font.hpp>`

Public Attributes

- Unicode::UnicodeChar unicodePrevChar

  The unicode for the first character in the kerning pair.
- int8_t distance

  The kerning distance.

18.70.1 Detailed Description

Structure providing information about a kerning for a given char pair. Used by LCD when rendering.

18.71 Keyboard::Key Struct Reference

Mapping from rectangle to key id.

```
#include <touchgfx/widgets/Keyboard.hpp>
```

Public Attributes

- uint8_t keyId

  The id of a key.
- Rect keyArea

  The area occupied by the key.
- BitmapId highlightBitmapId

  A bitmap to show when the area is "pressed".

18.72 Keyboard Class Reference

The keyboard provides text input for touch devices.

```
#include <touchgfx/widgets/Keyboard.hpp>
```

Classes

- struct CallbackArea

  Mapping from rectangle to a callback method to execute.
- struct Key

  Mapping from rectangle to key id.
- struct KeyMapping

  Mapping from key id to Unicode character.
- struct KeyMappingList

  List of KeyMappings to use.
- struct Layout

  Definition of the keyboard layout. The keyboard can handle changing layouts, so different keyboard modes can be implemented by changing layouts and key mappings.

Public Member Functions

- Keyboard ()

  Default Constructor.
- virtual ~Keyboard ()

  Destructor.
- void setBuffer (Unicode::UnicodeChar *newBuffer, uint16_t newBufferSize)

  Sets the buffer to be used by the keyboard.
- void setLayout (const Layout *newLayout)

  Set/change the Keyboard::Layout to use.
- void setTextIndentation ()

  Sets text indentation.
- const Layout * getLayout () const

  Gets the layout.
- void setKeymappingList (const KeyMappingList *newKeyMappingList)

Set/change the KeyMappingList to use.

- const KeyMappingList * getKeyMappingList () const

  Gets key mapping list.
- void setBufferPosition (uint16_t newPos)

  Change the buffer position.
- uint16_t getBufferPosition ()

  Gets buffer position.
- Unicode::UnicodeChar * getBuffer () const

  Gets the buffer.
- virtual void draw (const Rect &invalidatedArea) const

  Overrides the draw implementation on the Container.
- virtual void handleClickEvent (const ClickEvent &evt)

  Overrides the handleClickEvent on the container.
- void setKeyListener (GenericCallback< Unicode::UnicodeChar > &callback)

  Sets the callback for the keyboard.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Member Functions

- Key getKeyForCoordinates (int16_t x, int16_t y)

  Gets key for coordinates.
- Unicode::UnicodeChar getCharForKey (uint8_t keyId) const

  Maps a keyId to the UnicodeChar being displayed by that key.
- CallbackArea getCallbackAreaForCoordinates (int16_t x, int16_t y)

  Gets the callback area defined by the layout for the specified coordinates.
- virtual void setupDrawChain (const Rect &invalidatedArea, Drawable **nextPreviousElement)

  This function is merely overridden to also add the Keyboard itself to the list of drawables.

Protected Attributes

- GenericCallback
  < Unicode::UnicodeChar > * keyListener

  Pointer to callback being executed when a key is pressed.
- Unicode::UnicodeChar * buffer

  Pointer to zero-terminated buffer where the entered text is being displayed.
- uint16_t bufferSize

  Size of the buffer.
- uint16_t bufferPosition

  Current position in buffer.
- Image image

  Layout bitmap.
- TextAreaWithOneWildcard enteredText

  Widget capable of displaying the entered text buffer.
- const Layout * layout

  Pointer to layout.
- const KeyMappingList * keyMappingList

  Pointer to key mapping.
- Image highlightImage

  Image to display when a key is highlighted.

Additional Inherited Members

18.72.1 Detailed Description

The keyboard provides text input for touch devices. It is configured using a Layout and a KeyMappingList, which both can be changed at runtime. The class using the keyboard must provide a buffer where the entered text is placed. The Layout contains a bitmap id for the image to display and two mappings: rectangles to key ids and rectangles to callback methods.

The KeyMappingList maps key ids to unicode characters. When the user presses a key, the keyboard looks in its layout for a rectangle containing the coordinates pressed. If it finds a mapping to a callback method, it will invoke that method. If it finds a mapping to a key it will look up the unicode character for that key and place it in a text buffer. The sequence is: (x,y) -> KeyId -> UnicodeChar.

A keyboard with multiple key mappings e.g. lower case alpha, upper case alpha and numeric mappings can be created by implementing callback methods for shift and mode areas in the provided bitmap and then changing the KeyMappingList when those areas are pressed.

See also

Container

18.72.2 Constructor & Destructor Documentation

18.72.2.1 Keyboard ( )

Creates a new Keyboard.

18.72.2.2 ~Keyboard ( ) [inline],[virtual]

Destructor.

18.72.3 Member Function Documentation

18.72.3.1 void draw (const Rect & *invalidatedArea*) const [virtual]

Overrides the draw implementation on the Container. First invokes the container draw implementation to draw the keyboard bitmap and text area holding the entered text. If additional drawables have been added to the keyboard, they will also be draw. After invoking the container draw, the glyphs mapped to keys are drawn and if a key has been pressed, it will be highlighted.

Parameters

invalidatedArea	The area to draw.

Reimplemented from Container.

18.72.3.2 Unicode::UnicodeChar * getBuffer ( ) const [inline]

Gets the buffer.

Returns

The buffer containing entered text currently being displayed.

18.72.3.3 uint16_t getBufferPosition ( ) [inline]

Gets buffer position.

Returns

the buffer position, i.e. the current index where new characters will be placed.

18.72.3.4 CallbackArea getCallbackAreaForCoordinates (int16_t *x*, int16_t *y*) [protected]

Gets the callback area defined by the layout for the specified coordinates.

Parameters

x	The x coordinate to perform key look up with.
y	The y coordinate to perform key look up with.

Returns

The CallbackArea, which is empty if not found.

18.72.3.5 Unicode::UnicodeChar getCharForKey (uint8_t *keyId*) const [protected]

Maps a keyId to the UnicodeChar being displayed by that key.

Parameters

keyId	The id of the key to perform lookup with.

Returns

the UnicodeChar used for the specified key.

18.72.3.6 Key getKeyForCoordinates (int16_t *x*, int16_t *y*) [protected]

Gets key for coordinates.

Parameters

x	The x coordinate to perform key look up with.
y	The y coordinate to perform key look up with.

Returns

The key for the given coordinates.

18.72.3.7 const KeyMappingList * getKeyMappingList ( ) const [inline]

Gets key mapping list.

Returns

The KeyMappingList used by the Keyboard.

18.72.3.8 const Layout * getLayout ( ) const [inline]

Gets the layout.

Returns

The layout used by the Keyboard.

18.72.3.9 uint16_t getType ( ) const [inline],[virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_KEYBOARD.

Reimplemented from Container.

18.72.3.10 void handleClickEvent (const ClickEvent & *evt*) [virtual]

Overrides the handleClickEvent on the container. The keyboard handles all click events internally and click events are *not* propagated to drawables added to the keyboard.

Parameters

evt	The ClickEvent.

Reimplemented from Drawable.

18.72.3.11 void setBuffer (Unicode::UnicodeChar * *newBuffer*, uint16_t *newBufferSize*)

Sets the buffer to be used by the keyboard.

Parameters

in	*newBuffer*	Pointer to a buffer holding the text edited by the keyboard. If the buffer is not empty, the edit position for the keyboard will be set to the end of the provided text.
	newBufferSize	Length of the buffer, i.e. number of UnicodeChar's.

18.72.3.12 void setBufferPosition (uint16_t *newPos*)

Change the buffer position i.e. the next index to place a character when a key is pressed. This can be used to implement backspace functionality if the class using the Keyboard implements a callback and maps it to a backspace implementation. Setting the position will cause the TextArea displaying the text to be invalidated to request a redraw.

Parameters

newPos	The buffer position.

18.72.3.13 void setKeyListener (GenericCallback< Unicode::UnicodeChar > & *callback*) [inline]

Sets the callback for the keyboard. The callback will be executed every time a key is clicked. The callback argument contains the key that was just pressed.

Note

Backspace, shift and mode keys report a 0 as value.

Parameters

in	*callback*	The Callback to invoke.

18.72.3.14 void setKeymappingList (const KeyMappingList * *newKeyMappingList*)

Set/change the KeyMappingList to use. The Keyboard will invalidate the space it occupies to request a redraw.

Parameters

newKeyMappingList	The new KeyMappingList.

18.72.3.15 void setLayout (const Layout * *newLayout*)

Set/change the Keyboard::Layout to use.The Keyboard will invalidate the space it occupies to request a redraw.

Parameters

newLayout	The new layout.

18.72.3.16 void setTextIndentation ( )

Sets text indentation by making the area for entered text slightly larger. The result is that some characters (often 'j' and '_') will not be cut off. Indentation is added to both sides of the text area in case the text is right-to- left. Indentation is automatically set so all characters will display properly.

18.73 Keyboard::KeyMapping Struct Reference

Mapping from key id to Unicode character.

```
#include <touchgfx/widgets/Keyboard.hpp>
```

Public Attributes

- uint8_t keyId

  Id of a key.
- Unicode::UnicodeChar keyValue

  Unicode equivalent of the key id.

18.74 Keyboard::KeyMappingList Struct Reference

List of KeyMappings to use.

```
#include <touchgfx/widgets/Keyboard.hpp>
```

Public Attributes

- const KeyMapping * keyMappingArray

  The array of key mappings used by the keyboard.
- uint8_t numberOfKeys

  The number of keys in the list.

18.75 Keyboard::Layout Struct Reference

Definition of the keyboard layout. The keyboard can handle changing layouts, so different keyboard modes can be implemented by changing layouts and key mappings.

```
#include <touchgfx/widgets/Keyboard.hpp>
```

Public Attributes

- BitmapId bitmap

  The bitmap used for the keyboard layout.
- const Key * keyArray

  The keys on the keyboard layout.
- uint8_t numberOfKeys

  The number of keys on this keyboard layout.
- CallbackArea * callbackAreaArray

  The array of areas and corresponding callbacks.
- uint8_t numberOfCallbackAreas

  The number of areas and corresponding callbacks.
- Rect textAreaPosition

  The area where text is written.
- TypedText textAreaFont

  The font used for typing text.
- colortype textAreaFontColor

  The color used for the typing text.
- FontId keyFont

  The font used for the keys.
- colortype keyFontColor

  The color used for the keys.

18.76 LCD Class Reference

This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles.

```
#include <touchgfx/lcd/LCD.hpp>
```

Classes

- struct StringVisuals

  The visual elements when writing a string.

Public Member Functions

- virtual ~LCD ()

  Destructor.
- virtual void init ()

  Performs initialization.
- void drawGlyph (int16_t x, int16_t y, uint16_t offsetX, uint16_t offsetY, int16_t maxX, int16_t maxY, const GlyphNode *glyph, const uint8_t *glyphData, colortype color, uint8_t bitsPerPixel, uint8_t alpha=255)

  This method is deprecated and will be removed.
- virtual void drawPartialBitmap (const Bitmap &bitmap, int16_t x, int16_t y, const Rect &rect, uint8_t alpha=255, bool useOptimized=true)=0

  Draws a portion of a bitmap.
- virtual void blitCopy (const uint16_t *sourceData, const Rect &source, const Rect &blitRect, uint8_t alpha, bool hasTransparentPixels)=0

  Blits a 2D source-array to the framebuffer.
- virtual void blitCopy (const uint8_t *sourceData, Bitmap::BitmapFormat sourceFormat, const Rect &source, const Rect &blitRect, uint8_t alpha, bool hasTransparentPixels)=0

  Blits a 2D source-array to the framebuffer while converting the format.
- virtual uint16_t * copyFrameBufferRegionToMemory (const Rect &region)=0

  Copies a part of the frame buffer.
- virtual void fillRect (const Rect &rect, colortype color, uint8_t alpha=255)=0

  Draws a filled rectangle in the specified color.
- void drawHorizontalLine (int16_t x, int16_t y, uint16_t width, uint16_t lineWidth, colortype color, uint8_t alpha=255)

  Draws a horizontal line with the specified color.
- void drawVerticalLine (int16_t x, int16_t y, uint16_t height, uint16_t lineWidth, colortype color, uint8_t alpha=255)

  Draws a vertical line with the specified color.
- void drawRect (const Rect &rect, colortype color, uint8_t alpha=255)

  Draws a rectangle using the specified line color.
- void drawBorder (const Rect &rect, uint16_t lineWidth, colortype color, uint8_t alpha=255)

  Draws a rectangle width the specified line width and color.
- void drawString (Rect widgetArea, const Rect &invalidatedArea, StringVisuals &stringVisuals, const Unicode::UnicodeChar *format,...)

  Draws the specified unicode string.
- virtual uint8_t bitDepth () const =0

  Number of bits per pixel used by the display.
- void drawTextureMapTriangle (const DrawingSurface &dest, const Point3D *vertices, const TextureSurface &texture, const Rect &absoluteRect, const Rect &dirtyAreaAbsolute, RenderingVariant renderVariant, uint8_t alpha=255, uint16_t subDivisionSize=12)

  Texture map triangle. Draw a perspective correct texture mapped triangle. The vertices describes the surface, the x,y,z coordinates and the u,v coordinates of the texture. The texture contains the image data to be drawn The triangle line will be placed and clipped using the absolute and dirty rectangles The alpha will determine how the triangle should be alpha blended. The subDivisionSize will determine the size of the piecewise affine texture mapped portions of the triangle.

Protected Member Functions

- virtual void drawTextureMapScanLine (const DrawingSurface &dest, const Gradients &gradients, const Edge *leftEdge, const Edge *rightEdge, const TextureSurface &texture, const Rect &absoluteRect, const Rect &dirtyAreaAbsolute, RenderingVariant renderVariant, uint8_t alpha, uint16_t subDivisionLength)=0

Draw scan line. Draw one horizontal line of the texture map on screen. The scan line will be drawn using perspective correct texture mapping. The appearance of the line is determined by the left and right edge and the gradients structure. The edges contain the information about the x,y,z coordinates of the left and right side respectively and also information about the u,v coordinates of the texture map used. The gradients structure contains information about how to interpolate all the values across the scan line. The data drawn should be present in the texture argument.

- virtual void drawGlyph (uint16_t *wbuf, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offset↩Y, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation=TEXT_ROTATE_0)=0

  Private version of draw-glyph.
- void drawStringLTR (Rect widgetArea, const Rect &invalidatedArea, StringVisuals &stringVisuals, const Unicode::UnicodeChar *format, va_list pArg)

  Draws the specified unicode string.
- void drawStringRTL (Rect widgetArea, const Rect &invalidatedArea, StringVisuals &stringVisuals, const Unicode::UnicodeChar *format, va_list pArg)

  Draws the specified unicode string.

Static Protected Member Functions

- static void rotateRect (Rect &rect, const Rect &canvas, const TextRotation rotation)

  Rotate a rectangle inside another rectangle.
- static int realX (Rect &widgetArea, int16_t x, int16_t y, TextRotation rotation)

  Find the real, absolute x coordinate of a point inside a widget.
- static int realY (Rect &widgetArea, int16_t x, int16_t y, TextRotation rotation)

  Find the real, absolute y coordinate of a point inside a widget.
- static uint16_t stringWidth (TextProvider &textProvider, const Font &font, const int numChars, TextDirection textDirection)

  Find string width.

Static Protected Attributes

- static const uint16_t newLine = 10

  NewLine value.

Friends

- class **Font**

18.76.1 Detailed Description

This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles.

Note

All coordinates are expected to be in absolute coordinates!

18.76.2 Constructor & Destructor Documentation

18.76.2.1 ~LCD ( ) [inline],[virtual]

Destructor.

18.76.3 Member Function Documentation

18.76.3.1 uint8_t bitDepth ( ) const [pure virtual]

Number of bits per pixel used by the display.

Returns

The number of bits per pixel.

Implemented in LCD24bpp, LCD16bpp, and LCD1bpp.

18.76.3.2 void blitCopy (const uint16_t * *sourceData,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha,* bool *hasTransparentPixels*) [pure virtual]

Blits a 2D source-array to the framebuffer perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
has↩ Transparent↩ Pixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implemented in LCD24bpp, LCD16bpp, and LCD1bpp.

18.76.3.3 void blitCopy (const uint8_t * *sourceData,* Bitmap::BitmapFormat *sourceFormat,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha,* bool *hasTransparentPixels*) [pure virtual]

Blits a 2D source-array to the framebuffer perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
has↩ Transparent↩ Pixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implemented in LCD24bpp, LCD16bpp, and LCD1bpp.

18.76.3.4 uint16_t * copyFrameBufferRegionToMemory (const Rect & *region*) [pure virtual]

Copies a part of the frame buffer.

Parameters

region	The part to copy.

Returns

A pointer to the copy.

Implemented in LCD24bpp, LCD16bpp, and LCD1bpp.

18.76.3.5 void drawBorder (const Rect & *rect*, uint16_t *lineWidth*, colortype *color*, uint8_t *alpha* = 255)

Draws a rectangle width the specified line width and color.

Parameters

rect	The rectangle x, y, width, height in absolute coordinates.
lineWidth	The width of the line.
color	The color to use.
alpha	The alpha value to use (default=solid)

18.76.3.6 void drawGlyph (int16_t *x*, int16_t *y*, uint16_t *offsetX*, uint16_t *offsetY*, int16_t *maxX*, int16_t *maxY*, const GlyphNode * *glyph*, const uint8_t * *glyphData*, colortype *color*, uint8_t *bitsPerPixel*, uint8_t *alpha* = 255)

This method is deprecated and will be removed. Use drawString() instead. This function might not correctly render glyphs with negative value for "left" and could draw outside the allowed rectangle.

Draws a glyph with support for gray-map glyphs i.e. anti-aliasing using the gray color to determine the actual color to draw. Supports 1, 2, 4 and 8 bits per pixel.

Parameters

x	The x coordinate of the leftmost pixel.
y	The y coordinate of the topmost pixel.
offsetX	The offset x coordinate.
offsetY	The offset y coordinate.
maxX	The maximum x coordinate.
maxY	The maximum y coordinate.
glyph	The glyph.
glyphData	Information describing the glyph.
color	The color.
bitsPerPixel	The number of bits per pixel.
alpha	Default value is 255 (solid)

18.76.3.7 void drawGlyph (uint16_t * *wbuf*, Rect *widgetArea*, int16_t *x*, int16_t *y*, uint16_t *offsetX*, uint16_t *offsetY*, const Rect & *invalidatedArea*, const GlyphNode * *glyph*, const uint8_t * *glyphData*, colortype *color*, uint8_t *bitsPerPixel*, uint8_t *alpha*, TextRotation *rotation* = TEXT_ROTATE_0) [protected], [pure virtual]

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public version of drawGlyph()

Parameters

out	*wbuf*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw inside.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implemented in LCD24bpp, LCD16bpp, and LCD1bpp.

18.76.3.8 void drawHorizontalLine (int16_t *x*, int16_t *y*, uint16_t *width*, uint16_t *lineWidth*, **colortype** *color*, uint8_t *alpha* = 255)

Draws a horizontal line with the specified color.

Parameters

x	The x coordinate of the starting point, in absolute coordinates.
y	The y coordinate of the starting point, in absolute coordinates.
width	The length of the line.
lineWidth	The width of the line.
color	The color to use.
alpha	The alpha value to use (default=solid)

18.76.3.9 void drawPartialBitmap (const **Bitmap** & *bitmap*, int16_t *x*, int16_t *y*, const **Rect** & *rect*, uint8_t *alpha* = 255, bool *useOptimized* = true) [pure virtual]

Draws a portion of a bitmap.

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0,0) on the screen.
y	The absolute y coordinate to place pixel (0,0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implemented in LCD24bpp, LCD16bpp, and LCD1bpp.

18.76.3.10 void drawRect (const **Rect** & *rect*, **colortype** *color*, uint8_t *alpha* = 255)

Draws a rectangle using the specified line color. Same as calling drawBorder with a line width of 1.

Parameters

rect	The rectangle x, y, width, height in absolute coordinates.

color	The color to use.
alpha	The alpha value to use (default=solid)

18.76.3.11 void drawString (Rect *widgetArea*, const Rect & *invalidatedArea*, StringVisuals & *stringVisuals*, const Unicode::UnicodeChar * *format*, ...)

Draws the specified unicode string. Breaks line on newline.

Parameters

	widgetArea	The area covered by the drawing widget in absolute coordinates.
	invalidatedArea	The (sub)region of the widget area to draw, expressed relative to the widget area. If the widgetArea is (x, y, width, height) = (10, 10, 20, 20) and invalidatedArea is (x, y ,width, height) = (5, 5, 6, 6) the widgetArea drawn on the LCD is (x, y, width, height) = (15, 15, 6, 6).
in	*stringVisuals*	The string visuals (font, alignment, line space, color) with which to draw this string.
	format	A pointer to a zero terminated text string with optional additional wildcard arguments.
	...	Variable arguments providing additional information.

18.76.3.12 void drawStringLTR (Rect *widgetArea*, const Rect & *invalidatedArea*, StringVisuals & *stringVisuals*, const Unicode::UnicodeChar * *format*, va_list *pArg*) [protected]

Draws the specified unicode string. Breaks line on newline. The string is assumed to contain only latin characters written left-to-right.

Parameters

	widgetArea	The area covered by the drawing widget in absolute coordinates.
	invalidatedArea	The (sub)region of the widget area to draw, expressed relative to the widget area. If the widgetArea is (x, y, width, height) = (10, 10, 20, 20) and invalidatedArea is (x, y ,width, height) = (5, 5, 6, 6) the widgetArea drawn on the LCD is (x, y, width, height) = (15, 15, 6, 6).
in	*stringVisuals*	The string visuals (font, alignment, line space, color) with which to draw this string.
	format	A pointer to a zero terminated text string with optional additional wildcard arguments.
	pArg	Variable arguments providing additional information.

See also

drawString

18.76.3.13 void drawStringRTL (Rect *widgetArea*, const Rect & *invalidatedArea*, StringVisuals & *stringVisuals*, const Unicode::UnicodeChar * *format*, va_list *pArg*) [protected]

Draws the specified unicode string. Breaks line on newline. The string can be either right-to-left or left-to-right and may contain sequences of Arabic /Hebrew and Latin characters.

Parameters

	widgetArea	The area covered by the drawing widget in absolute coordinates.
	invalidatedArea	The (sub)region of the widget area to draw, expressed relative to the widget area. If the widgetArea is (x, y, width, height) = (10, 10, 20, 20) and invalidatedArea is (x, y ,width, height) = (5, 5, 6, 6) the widgetArea drawn on the LCD is (x, y, width, height) = (15, 15, 6, 6).
in	*stringVisuals*	The string visuals (font, alignment, line space, color) with which to draw this string.
	format	A pointer to a zero terminated text string with optional additional wildcard arguments.
	pArg	Variable arguments providing additional information.

See also

drawString

18.76.3.14 void drawTextureMapScanLine (const DrawingSurface & *dest*, const Gradients & *gradients*, const Edge * *leftEdge*, const Edge * *rightEdge*, const TextureSurface & *texture*, const Rect & *absoluteRect*, const Rect & *dirtyAreaAbsolute*, RenderingVariant *renderVariant*, uint8_t *alpha*, uint16_t *subDivisionLength*) [protected],[pure virtual]

The scan line will be drawn using the additional arguments. The scan line will be placed and clipped using the absolute and dirty rectangles The alpha will determine how the scan line should be alpha blended. The subDivisionSize will determine the size of the piecewise affine texture mapped lines.

Parameters

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionLength	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implemented in LCD24bpp, LCD16bpp, and LCD1bpp.

18.76.3.15 void drawTextureMapTriangle (const DrawingSurface & *dest*, const Point3D * *vertices*, const TextureSurface & *texture*, const Rect & *absoluteRect*, const Rect & *dirtyAreaAbsolute*, RenderingVariant *renderVariant*, uint8_t *alpha* = 255, uint16_t *subDivisionSize* = 12)

Parameters

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
vertices	The vertices of the triangle.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.

renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	the alpha. Default = 255.
subDivisionSize	the size of the subdivisions of the scan line. Default = 12.

18.76.3.16 void drawVerticalLine (int16_t *x*, int16_t *y*, uint16_t *height*, uint16_t *lineWidth*, colortype *color*, uint8_t *alpha* = 255)

Draws a vertical line with the specified color.

Parameters

x	The x coordinate of the starting point, in absolute coordinates.
y	The y coordinate of the starting point, in absolute coordinates.
height	The length of the line.
lineWidth	The width of the line.
color	The color to use.
alpha	The alpha value to use (default=solid)

18.76.3.17 void fillRect (const Rect & *rect*, colortype *color*, uint8_t *alpha* = 255) [pure virtual]

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implemented in LCD24bpp, LCD16bpp, and LCD1bpp.

18.76.3.18 void init ( ) [inline],[virtual]

Performs initialization.

Reimplemented in LCD16bpp, and LCD24bpp.

18.76.3.19 static int realX (Rect & *widgetArea*, int16_t *x*, int16_t *y*, TextRotation *rotation*) [static], [protected]

Find the real, absolute x coordinate of a point inside a widget with regards to rotation.

Parameters

in	*widgetArea*	The widget containing the point.
	x	The x coordinate.
	y	The y coordinate.
	rotation	Rotation to perform.

Returns

The absolute x coordinate after applying appropriate rotation.

18.76.3.20 static int realY (Rect & *widgetArea*, int16_t *x*, int16_t *y*, TextRotation *rotation*) [static], [protected]

Find the real, absolute y coordinate of a point inside a widget with regards to rotation.

Parameters

in	*widgetArea*	The widget containing the point.
	x	The x coordinate.
	y	The y coordinate.
	rotation	Rotation to perform.

Returns

The absolute y coordinate after applying appropriate rotation.

18.76.3.21 static void rotateRect (Rect & *rect*, const Rect & *canvas*, const TextRotation *rotation*) [static], [protected]

Rotate a rectangle inside another rectangle.

Parameters

in,out	*rect*	The rectangle to rotate.
	canvas	The rectangle containing the rect to rotate.
	rotation	Rotation to perform on rect.

18.76.3.22 static uint16_t stringWidth (TextProvider & *textProvider*, const Font & *font*, const int *numChars*, TextDirection *textDirection*) [static],[protected]

Find string with of the given number of characters read from the given TextProvider.

Parameters

in,out	*textProvider*	The text provider.
	font	The font.
	numChars	Number of characters.
	textDirection	The text direction.

Returns

An int16_t.

18.77 LCD16bpp Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

```
#include <platform/driver/lcd/LCD16bpp.hpp>
```

Public Member Functions

- virtual void init ()

  Performs initialization.
- virtual void drawPartialBitmap (const Bitmap &bitmap, int16_t x, int16_t y, const Rect &rect, uint8_t alpha=255, bool useOptimized=true)

  Draws a portion of a bitmap.
- virtual void blitCopy (const uint16_t *sourceData, const Rect &source, const Rect &blitRect, uint8_t alpha, bool hasTransparentPixels)

  Blits a 2D source-array to the framebuffer.

- virtual void blitCopy (const uint8_t *sourceData, Bitmap::BitmapFormat sourceFormat, const Rect &source, const Rect &blitRect, uint8_t alpha, bool hasTransparentPixels)

  Blits a 2D source-array to the framebuffer while converting the format.
- virtual uint16_t * copyFrameBufferRegionToMemory (const Rect &region)

  Copies a part of the frame buffer.
- virtual void fillRect (const Rect &rect, colortype color, uint8_t alpha=255)

  Draws a filled rectangle in the specified color.
- virtual uint8_t bitDepth () const

  Number of bits per pixel used by the display.

Protected Member Functions

- virtual void drawTextureMapScanLine (const DrawingSurface &dest, const Gradients &gradients, const Edge *leftEdge, const Edge *rightEdge, const TextureSurface &texture, const Rect &absoluteRect, const Rect &dirtyAreaAbsolute, RenderingVariant renderVariant, uint8_t alpha, uint16_t subDivisionSize)

  Draw scan line. Draw one horizontal line of the texture map on screen. The scan line will be drawn using perspective correct texture mapping. The appearance of the line is determined by the left and right edge and the gradients structure. The edges contain the information about the x,y,z coordinates of the left and right side respectively and also information about the u,v coordinates of the texture map used. The gradients structure contains information about how to interpolate all the values across the scan line. The data drawn should be present in the texture argument.
- virtual void drawGlyph (uint16_t *wbuf, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offset↩Y, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation=TEXT_ROTATE_0)

  Private version of draw-glyph with explicit destination buffer pointer argument.

Static Protected Member Functions

- static int nextPixel (bool portrait, TextRotation rotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool portrait, TextRotation rotation)

  Find out how much to advance in the display buffer to get to the next line.
- static void blitCopyARGB8888 (const uint32_t *sourceData, const Rect &source, const Rect &blitRect, uint8↩_t alpha)

  Blits a 2D source-array to the framebuffer.
- static void blitCopyAlphaPerPixel (const uint16_t *sourceData, const uint8_t *alphaData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D source-array to the framebuffer.

Static Protected Attributes

- static const uint16_t TRANSPARENT_COL = 0xABCD

  Transparency color. Deprecated, do not use.

18.77.1 Detailed Description

This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles on 16 bits per pixel displays.

Note

All coordinates are expected to be in absolute coordinates!

See also

LCD

18.77.2 Member Function Documentation

18.77.2.1 uint8_t bitDepth ( ) const [inline],[virtual]

Number of bits per pixel used by the display.

Returns

The number of bits per pixel.

Implements LCD.

18.77.2.2 void blitCopy (const uint16_t * *sourceData,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha,* bool *hasTransparentPixels*) [virtual]

Blits a 2D source-array to the framebuffer perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
has↩ Transparent↩ Pixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

18.77.2.3 void blitCopy (const uint8_t * *sourceData,* Bitmap::BitmapFormat *sourceFormat,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha,* bool *hasTransparentPixels*) [virtual]

Blits a 2D source-array to the framebuffer perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD16 supports source data formats: RGB565 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
has↩ Transparent↩ Pixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

18.77.2.4 static void blitCopyAlphaPerPixel (const uint16_t * *sourceData,* const uint8_t * *alphaData,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha*) [static], [protected]

Blits a 2D source-array to the framebuffer perfoming alpha-blending per pixel as specified Performs always a software blend.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16- bits RGB565 values.
alphaData	The alpha channel array pointer (points to the beginning of the data)
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

18.77.2.5 static void blitCopyARGB8888 (const uint32_t * *sourceData,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha*) [static],[protected]

Blits a 2D source-array to the framebuffer perfoming alpha-blending per pixel as specified if ARGB8888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

18.77.2.6 uint16_t * copyFrameBufferRegionToMemory (const Rect & *region*) [virtual]

Copies a part of the frame buffer.

Parameters

region	The part to copy.

Returns

A pointer to the copy.

Implements LCD.

18.77.2.7 void drawGlyph (uint16_t * *wbuf,* Rect *widgetArea,* int16_t *x,* int16_t *y,* uint16_t *offsetX,* uint16_t *offsetY,* const Rect & *invalidatedArea,* const GlyphNode * *glyph,* const uint8_t * *glyphData,* colortype *color,* uint8_t *bitsPerPixel,* uint8_t *alpha,* TextRotation *rotation* = TEXT_ROTATE_0) [protected],[virtual]

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public version of drawGlyph()

Parameters

in	*wbuf*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.

	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

18.77.2.8 void drawPartialBitmap (const Bitmap & *bitmap*, int16_t *x*, int16_t *y*, const Rect & *rect*, uint8_t *alpha* = 255, bool *useOptimized* = true) [virtual]

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

18.77.2.9 void drawTextureMapScanLine (const DrawingSurface & *dest*, const Gradients & *gradients*, const Edge * *leftEdge*, const Edge * *rightEdge*, const TextureSurface & *texture*, const Rect & *absoluteRect*, const Rect & *dirtyAreaAbsolute*, RenderingVariant *renderVariant*, uint8_t *alpha*, uint16_t *subDivisionSize*) [protected], [virtual]

The scan line will be drawn using the additional arguments. The scan line will be placed and clipped using the absolute and dirty rectangles The alpha will determine how the scan line should be alpha blended. The sub↩DivisionSize will determine the size of the piecewise affine texture mapped lines.

Parameters

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyArea↩Absolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

18.77.2.10 void fillRect (const Rect & *rect*, colortype *color*, uint8_t *alpha* = 255) [virtual]

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

18.77.2.11 void init ( ) [virtual]

Performs initialization.

Reimplemented from LCD.

18.77.2.12 static int nextLine (bool *portrait*, TextRotation *rotation*) [static], [protected]

Find out how much to advance in the display buffer to get to the next line.

Parameters

portrait	Is the display running in portrait mode?
rotation	Rotation to perform.

Returns

How much to advance to get to the next line.

18.77.2.13 static int nextPixel (bool *portrait*, TextRotation *rotation*) [static], [protected]

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

portrait	Is the display running in portrait mode?
rotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

18.78 LCD1bpp Class Reference

This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles.

```
#include <platform/driver/lcd/LCD1bpp.hpp>
```

Public Member Functions

- virtual void drawPartialBitmap (const Bitmap &bitmap, int16_t x, int16_t y, const Rect &rect, uint8_t alpha=255, bool useOptimized=true)

  Draws a portion of a bitmap.
- virtual void blitCopy (const uint16_t *sourceData, const Rect &source, const Rect &blitRect, uint8_t alpha, bool hasTransparentPixels)

  Blits a 2D source-array to the frame buffer.
- virtual void blitCopy (const uint8_t *sourceData, Bitmap::BitmapFormat sourceFormat, const Rect &source, const Rect &blitRect, uint8_t alpha, bool hasTransparentPixels)

Blits a 2D source-array to the framebuffer while converting the format.

- virtual uint16_t * copyFrameBufferRegionToMemory (const Rect &region)

  Copies a part of the frame buffer.
- virtual void fillRect (const Rect &rect, colortype color, uint8_t alpha=255)

  Draws a filled rectangle in the specified color.
- virtual uint8_t bitDepth () const

  Number of bits per pixel used by the display.

Protected Member Functions

- virtual void drawTextureMapScanLine (const DrawingSurface &dest, const Gradients &gradients, const Edge *leftEdge, const Edge *rightEdge, const TextureSurface &texture, const Rect &absoluteRect, const Rect &dirtyAreaAbsolute, RenderingVariant renderVariant, uint8_t alpha, uint16_t subDivisionSize)

  Draw scan line. Not supported for 1bpp.
- virtual void drawGlyph (uint16_t *wbuf, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offset↩Y, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation=TEXT_ROTATE_0)

  Private version of draw-glyph with explicit destination buffer pointer argument.
- virtual void blitCopyRLE (const uint16_t *_sourceData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a run-length encoded 2D source-array to the frame buffer.

Static Protected Member Functions

- static int nextPixel (bool portrait, TextRotation rotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool portrait, TextRotation rotation)

  Find out how much to advance in the display buffer to get to the next line.
- static void fillMemory (void *RESTRICT dst, colortype color, uint16_t bytesToFill)

  Fill memory efficiently.

Additional Inherited Members

18.78.1 Detailed Description

This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles on 1 bits per pixel displays.

Note

All coordinates are expected to be in absolute coordinates!

See also

LCD

18.78.2 Member Function Documentation

18.78.2.1 uint8_t bitDepth ( ) const [inline], [virtual]

Number of bits per pixel used by the display.

Returns

1.

Implements LCD.

18.78.2.2 void blitCopy (const uint16_t * *sourceData,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha,* bool *hasTransparentPixels*) [virtual]

Blits a 2D source-array to the frame buffer unless alpha is zero.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (0 = invisible, otherwise solid).
has↩ Transparent↩ Pixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

18.78.2.3 void blitCopy (const uint8_t * *sourceData,* Bitmap::BitmapFormat *sourceFormat,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha,* bool *hasTransparentPixels*) [virtual]

Blits a 2D source-array to the framebuffer perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
has↩ Transparent↩ Pixels	Ignored

Implements LCD.

18.78.2.4 void blitCopyRLE (const uint16_t * *_sourceData,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha*) [protected], [virtual]

Blits a run-length encoded2D source-array to the frame buffer unless alpha is zero.

Parameters

_sourceData	The source-array pointer (points to the beginning of the data). Data stored in RLE format, where each byte indicates number of pixels with certain color, alternating between black and white. First byte represents black.

source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (0 = invisible, otherwise solid).

18.78.2.5 uint16_t * copyFrameBufferRegionToMemory (const Rect & *region*) [virtual]

Copies a part of the frame buffer.

Parameters

region	The part to copy.

Returns

A pointer to the copy.

Implements LCD.

18.78.2.6 void drawGlyph (uint16_t * *wbuf*, Rect *widgetArea*, int16_t *x*, int16_t *y*, uint16_t *offsetX*, uint16_t *offsetY*, const Rect & *invalidatedArea*, const GlyphNode * *glyph*, const uint8_t * *glyphData*, colortype *color*, uint8_t *bitsPerPixel*, uint8_t *alpha*, TextRotation *rotation* = TEXT_ROTATE_0) [protected], [virtual]

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public version of drawGlyph()

Parameters

in	*wbuf*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

18.78.2.7 void drawPartialBitmap (const Bitmap & *bitmap*, int16_t *x*, int16_t *y*, const Rect & *rect*, uint8_t *alpha* = 255, bool *useOptimized* = true) [virtual]

Draws a portion of a bitmap.

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.

rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value (0 = invisible, otherwise solid). Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

18.78.2.8 void drawTextureMapScanLine (const DrawingSurface & *dest*, const Gradients & *gradients*, const Edge * *leftEdge*, const Edge * *rightEdge*, const TextureSurface & *texture*, const Rect & *absoluteRect*, const Rect & *dirtyAreaAbsolute*, RenderingVariant *renderVariant*, uint8_t *alpha*, uint16_t *subDivisionSize*) [inline], [protected], [virtual]

Parameters

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyArea↩ Absolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

18.78.2.9 static void fillMemory (void *RESTRICT *dst*, colortype *color*, uint16_t *bytesToFill*) [static], [protected]

Fill memory efficiently. Try to get 32bit aligned or 16bit aligned and then copy as quickly as possible.

Parameters

out	*dst*	Pointer to memory to fill.
	color	Color to write to memory, either 0 => 0x00000000 or 1 => 0xFFFFFFFF.
	bytesToFill	Number of bytes to fill.

18.78.2.10 void fillRect (const Rect & *rect*, colortype *color*, uint8_t *alpha* = 255) [virtual]

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color (values other than 0 or 1 are treated as being 1).
alpha	The rectangle opacity (0 = invisible, otherwise solid). Default is 255 (solid).

Implements LCD.

18.78.2.11 static int nextLine (bool *portrait*, TextRotation *rotation*) [static], [protected]

Find out how much to advance in the display buffer to get to the next line.

Parameters

portrait	Is the display running in portrait mode?
rotation	Rotation to perform.

Returns

How much to advance to get to the next line.

18.78.2.12 static int nextPixel (bool *portrait*, TextRotation *rotation*) [static],[protected]

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

portrait	Is the display running in portrait mode?
rotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

18.79 LCD24bpp Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

```
#include <platform/driver/lcd/LCD24bpp.hpp>
```

Public Member Functions

- virtual void init ()

  Performs initialization.
- virtual void drawPartialBitmap (const Bitmap &bitmap, int16_t x, int16_t y, const Rect &rect, uint8_t alpha=255, bool useOptimized=true)

  Draws a portion of a bitmap.
- virtual void blitCopy (const uint16_t *sourceData, const Rect &source, const Rect &blitRect, uint8_t alpha, bool hasTransparentPixels)

  Blits a 2D source-array to the framebuffer.
- virtual void blitCopy (const uint8_t *sourceData, Bitmap::BitmapFormat sourceFormat, const Rect &source, const Rect &blitRect, uint8_t alpha, bool hasTransparentPixels)

  Blits a 2D source-array to the framebuffer while converting the format.
- virtual uint16_t * copyFrameBufferRegionToMemory (const Rect &region)

  Copies a part of the frame buffer.
- virtual void fillRect (const Rect &rect, colortype color, uint8_t alpha=255)

  Draws a filled rectangle in the specified color.
- virtual uint8_t bitDepth () const

  Number of bits per pixel used by the display.

Protected Member Functions

- virtual void drawTextureMapScanLine (const DrawingSurface &dest, const Gradients &gradients, const Edge *leftEdge, const Edge *rightEdge, const TextureSurface &texture, const Rect &absoluteRect, const Rect &dirtyAreaAbsolute, RenderingVariant renderVariant, uint8_t alpha, uint16_t subDivisionSize)

Draw scan line. Draw one horizontal line of the texture map on screen. The scan line will be drawn using perspective correct texture mapping. The appearance of the line is determined by the left and right edge and the gradients structure. The edges contain the information about the x,y,z coordinates of the left and right side respectively and also information about the u,v coordinates of the texture map used. The gradients structure contains information about how to interpolate all the values across the scan line. The data drawn should be present in the texture argument.

- virtual void drawGlyph (uint16_t *wbuf, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offset↩Y, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation=TEXT_ROTATE_0)

  Private version of draw-glyph with explicit destination buffer pointer argument.

Static Protected Member Functions

- static int nextPixel (bool portrait, TextRotation rotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool portrait, TextRotation rotation)

  Find out how much to advance in the display buffer to get to the next line.
- static void blitCopyARGB8888 (const uint32_t *sourceData, const Rect &source, const Rect &blitRect, uint8↩_t alpha)

  Blits a 2D source-array to the framebuffer.

Static Protected Attributes

- static const uint16_t TRANSPARENT_COL = 0xABCD

  Transparency color.

18.79.1 Detailed Description

This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles on 16 bits per pixel displays.

Note

All coordinates are expected to be in absolute coordinates!

See also

LCD

18.79.2 Member Function Documentation

18.79.2.1 uint8_t bitDepth ( ) const [inline], [virtual]

Number of bits per pixel used by the display.

Returns

The number of bits per pixel.

Implements LCD.

18.79.2.2 void blitCopy (const uint16_t * *sourceData*, const Rect & *source*, const Rect & *blitRect*, uint8_t *alpha*, bool *hasTransparentPixels*) [virtual]

Blits a 2D source-array to the framebuffer perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
has↩ Transparent↩ Pixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

18.79.2.3 void blitCopy (const uint8_t * *sourceData,* Bitmap::BitmapFormat *sourceFormat,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha,* bool *hasTransparentPixels*) [virtual]

Blits a 2D source-array to the framebuffer perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD16 supports source data formats: RGB888 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
has↩ Transparent↩ Pixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

18.79.2.4 static void blitCopyARGB8888 (const uint32_t * *sourceData,* const Rect & *source,* const Rect & *blitRect,* uint8_t *alpha*) [static],[protected]

Blits a 2D source-array to the framebuffer perfoming alpha-blending per pixel as specified if ARGB8888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

18.79.2.5 uint16_t * copyFrameBufferRegionToMemory (const Rect & *region*) [virtual]

Copies a part of the frame buffer.

Parameters

region	The part to copy.

Returns

A pointer to the copy.

Implements LCD.

18.79.2.6 void drawGlyph (uint16_t * *wbuf*, Rect *widgetArea*, int16_t *x*, int16_t *y*, uint16_t *offsetX*, uint16_t *offsetY*, const Rect & *invalidatedArea*, const GlyphNode * *glyph*, const uint8_t * *glyphData*, colortype *color*, uint8_t *bitsPerPixel*, uint8_t *alpha*, TextRotation *rotation* = TEXT_ROTATE_0) [protected], [virtual]

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public version of drawGlyph()

Parameters

in	*wbuf*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

18.79.2.7 void drawPartialBitmap (const Bitmap & *bitmap*, int16_t *x*, int16_t *y*, const Rect & *rect*, uint8_t *alpha* = 255, bool *useOptimized* = true) [virtual]

Draws a portion of a bitmap. Skips pixels with color = TRANSPARENT_COL.

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

18.79.2.8 void drawTextureMapScanLine (const DrawingSurface & *dest*, const Gradients & *gradients*, const Edge * *leftEdge*, const Edge * *rightEdge*, const TextureSurface & *texture*, const Rect & *absoluteRect*, const Rect & *dirtyAreaAbsolute*, RenderingVariant *renderVariant*, uint8_t *alpha*, uint16_t *subDivisionSize*) [protected], [virtual]

The scan line will be drawn using the additional arguments. The scan line will be placed and clipped using the absolute and dirty rectangles The alpha will determine how the scan line should be alpha blended. The sub↩DivisionSize will determine the size of the piecewise affine texture mapped lines.

Parameters

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyArea↩ Absolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

18.79.2.9 void fillRect (const Rect & *rect*, colortype *color*, uint8_t *alpha* = 255) [virtual]

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

18.79.2.10 void init ( ) [virtual]

Performs initialization.

Reimplemented from LCD.

18.79.2.11 static int nextLine (bool *portrait*, TextRotation *rotation*) [static], [protected]

Find out how much to advance in the display buffer to get to the next line.

Parameters

portrait	Is the display running in portrait mode?
rotation	Rotation to perform.

Returns

How much to advance to get to the next line.

18.79.2.12 static int nextPixel (bool *portrait*, TextRotation *rotation*) [static], [protected]

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

portrait	Is the display running in portrait mode?
rotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

18.80 LED Class Reference

A led.

`#include <touchgfx\hal\LED.hpp>`

Static Public Member Functions

- static void init ()

  Perform configuration of IO pins.
- static void on (uint8_t nr)

  Turn on a LED.
- static void off (uint8_t nr)

  Turn off a LED.
- static void toggle (uint8_t nr)

  Toggles a LED.
- static bool get (uint8_t nr)

  Get state of a LED.

18.80.1 Member Function Documentation

18.80.1.1 static bool get (uint8_t *nr*) [static]

Get state of a LED.

Parameters

nr	of the LED to toggle.

Returns

the state of the LED.

18.80.1.2 static void init ( ) [static]

Perform configuration of IO pins.

18.80.1.3 static void off (uint8_t *nr*) [static]

Turn off a LED.

Parameters

nr	of the LED to switch off.

18.80.1.4 static void on (uint8_t *nr*) [static]

Turn on a LED.

Parameters

nr	of the LED to switch on.

18.80.1.5 static void toggle (uint8_t *nr*) [static]

Toggles a LED.

Parameters

nr	of the LED to toggle.

18.81 Line Class Reference

Simple CanvasWidget capable of drawing a line.

```
#include <touchgfx/widgets/canvas/Line.hpp>
```

Public Types

- enum LINE_ENDING_STYLE { BUTT_CAP_ENDING, SQUARE_CAP_ENDING }

  Values that represent line ending styles.

Public Member Functions

- Line ()

  Construct a new Line.
- template<class T >
  void setLine (T x1, T y1, T x2, T y2)

  Sets the endpoints of the line.
- template<class T >
  void setStart (T x, T y)

  Sets the start point for this Line.
- template<class T >
  void updateStart (T x, T y)

  Update the start point for this Line.
- template<class T >
  void getStart (T &x, T &y) const

  Gets the start coordinates for the line.
- template<class T >
  void setEnd (T x, T y)

  Sets the end point for this Line.
- template<class T >
  void updateEnd (T x, T y)

Update the end point for this Line.

- template<class T >
  void getEnd (T &x, T &y) const

  Gets the end coordinates for the line.
- template<class T >
  void setLineWidth (T width)

  Sets the width for this Line.
- template<class T >
  void updateLineWidth (T width)

  Update the width for this Line.
- template<class T >
  void getLineWidth (T &width) const

  Gets line width.
- void setLineEndingStyle (LINE_ENDING_STYLE lineEnding)

  Sets line ending style.
- LINE_ENDING_STYLE getLineEndingStyle ()

  Gets line ending style.
- virtual bool drawCanvasWidget (const Rect &invalidatedArea) const

  Draws the Line.
- virtual Rect getMinimalRect () const

  Gets minimal rectangle containing the shape drawn by this widget.

Additional Inherited Members

18.81.1 Detailed Description

Simple CanvasWidget capable of drawing a line from one point to another point. The end points can be moved to new locations and the line width can easily be set and changed. A 10 pixel long line along the top of the screen with a width on 1 pixel has endpoints in (0, 0.5) and (10, 0.5) and line width 1. The Line class calculates the corners of the shape, which in this case would be (0, 0), (10, 0), (10, 1) and (0, 1) and tells CWR to moveTo the first coordinate and then lineTo the next coordinates in order. Finally it tells CWR to render the inside of the shape using a Painter object.

The Line class caches the four corners of the shape to speed up redrawing. In general, drawing lines involve some extra mathematics for calculating the normal vector of the line and this computation would slow down re-draws if not cached.

See also

CanvasWidget

18.81.2 Member Enumeration Documentation

18.81.2.1 enum LINE_ENDING_STYLE

Enumerator

BUTT_CAP_ENDING The line ending is cut 90 degrees at the end of the line.

SQUARE_CAP_ENDING The line ending is cut 90 degrees, but extends half the width of the line.

18.81.3 Constructor & Destructor Documentation

18.81.3.1 Line ( )

Construct a new line.

18.81.4 Member Function Documentation

18.81.4.1 bool drawCanvasWidget (const Rect & *invalidatedArea*) const [virtual]

Draws the Line. This class supports partial drawing, so only the area described by the rectangle will be drawn. As the corners of the shape are cached, the line can quickly be redrawn when required.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Returns

true if it succeeds, false if it fails.

Implements CanvasWidget.

18.81.4.2 template< class T > void getEnd (T & *x*, T & *y*) const [inline]

Gets the end coordinates for the line.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*x*	The x coordinate.
out	*y*	The y coordinate.

See also

setEnd
setLine

18.81.4.3 LINE_ENDING_STYLE getLineEndingStyle ( )

Gets line ending style. See LINE_ENDING_STYLE for possible styles.

Returns

The line ending style.

See also

LINE_ENDING_STYLE
setLineEndingStyle

18.81.4.4 template< class T > void getLineWidth (T & *width*) const [inline]

Gets line width.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*width*	The line width.

See also

setLineWidth

18.81.4.5 Rect getMinimalRect ( ) const [virtual]

Gets minimal rectangle containing the shape drawn by this widget. Default implementation returns the size of the entire widget, but this function should be overwritten in subclasses and return the minimal rectangle containing the shape. See classes such as Circle for example implementations.

Returns

The minimal rectangle containing the shape drawn by this widget.

Reimplemented from CanvasWidget.

18.81.4.6 template< class T > void getStart (T & *x*, T & *y*) const [inline]

Gets the start coordinates for the line.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*x*	The x coordinate.
out	*y*	The y coordinate.

See also

setStart
setLine

18.81.4.7 template< class T > void setEnd (T *x*, T *y*) [inline]

Sets the end point for this Line.

Note

The area containing the Line is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the end point.
y	The y coordinate of the end point.

See also

updateEnd
getEnd

18.81.4.8 template< class T > void setLine (T *x1*, T *y1*, T *x2*, T *y2*) [inline]

Sets the endpoints of the line.

Note

The area containing the Line is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x1	The x coordinate of the start point.
y1	The y coordinate of the start point.
x2	The x coordinate of the end point.
y2	The y coordinate of the end point.

See also

setStart
setEnd

18.81.4.9 void setLineEndingStyle (LINE_ENDING_STYLE *lineEnding*)

Sets line ending style. See LINE_ENDING_STYLE for possible styles.

Note

The area containing the Line is not invalidated.

Parameters

lineEnding	The line ending style.

See also

LINE_ENDING_STYLE
getLineEndingStyle

18.81.4.10 template< class T > void setLineWidth (T *width*) [inline]

Sets the width for this Line.

Note

The area containing the Line is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

width	The width of the line measured in pixels.

See also

updateLineWidth

18.81.4.11 template< class T > void setStart (T *x*, T *y*) [inline]

Sets the start point for this Line.

Note

The area containing the Line is not invalidated.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the start point.
y	The y coordinate of the start point.

See also

updateStart
getStart
setLine
setEnd

18.81.4.12 template< class T > void updateEnd (T *x*, T *y*) [inline]

Update the end point for this Line. The rectangle that surrounds the line before and after will be invalidated. This method is intended for small changes in the end point of the line.

Note

The area containing the Line is invalidated before and after the change.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the end point.
y	The y coordinate of the end point.

18.81.4.13 template< class T > void updateLineWidth (T *width*) [inline]

Update the width for this Line and invalidates the minimal rectangle surrounding the line on screen.

Note

The area containing the Line is invalidated before and after the change.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

width	The width of the line measured in pixels.

See also

setLineWidth

18.81.4.14 template< class T > void updateStart (T *x*, T *y*) [inline]

Update the start point for this Line. The rectangle that surrounds the line before and after will be invalidated. This method is intended for small changes in the start point of the line.

Note

The area containing the Line is invalidated before and after the change.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the start point.
y	The y coordinate of the start point.

See also

setStart
updateEnd

18.82 ListLayout Class Reference

This class provides a layout mechanism for arranging Drawable instances adjacently in the specified Direction.

```
#include <touchgfx/containers/ListLayout.hpp>
```

Public Member Functions

- ListLayout (const Direction d=SOUTH)

  Constructor.
- virtual ~ListLayout ()

  Destructor.
- virtual void add (Drawable &d)

  Adds a Drawable instance to the end of the list.
- virtual void remove (Drawable &d)

  Removes a Drawable.
- virtual void insert (Drawable *previousElement, Drawable &d)

Inserts a Drawable.

- virtual void removeAll ()

  Removes all children.

- virtual uint16_t getType () const

  For GUI testing only.

Additional Inherited Members

18.82.1 Detailed Description

This class provides a layout mechanism for arranging Drawable instances adjacently in the specified Direction. The first element in the ListLayout is positioned in the ListLayout origin (0,0). The dimension of this class is automatically expanded to cover the area of the added Drawable instances.

See also

Container

18.82.2 Constructor & Destructor Documentation

18.82.2.1 ListLayout (const Direction *d* = SOUTH) [inline]

Constructor. Constructs a ListLayout instance that arranges the added elements in the specified Direction.

Parameters

d	The direction to grow in when adding children.

18.82.2.2 ~ListLayout ( ) [inline], [virtual]

Destructor.

18.82.3 Member Function Documentation

18.82.3.1 void add (Drawable & *d*) [virtual]

Adds a Drawable instance to the end of the list. The Drawable dimensions shall be set prior to addition.

Parameters

in	*d*	The Drawable to add.

Reimplemented from Container.

18.82.3.2 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_LISTLAYOUT.

Reimplemented from Container.

18.82.3.3 void insert (Drawable * *previousElement*, Drawable & *d*) [virtual]

Inserts a Drawable.

Parameters

in	*previousElement*	The element to insert the new element after.
in	*d*	The element to insert.

Reimplemented from Container.

18.82.3.4 void remove (Drawable & *d*) [virtual]

Removes a Drawable. Safe to call even if drawable has not been added.

Parameters

in	*d*	The drawable to remove.

Reimplemented from Container.

18.82.3.5 void removeAll ( ) [virtual]

Removes all children by resetting their parent and sibling pointers. In addition, the geometry is reset and any parent is signaled of the change.

Reimplemented from Container.

18.83 LockFreeDMA_Queue Class Reference

This implements a simple lock-free FIFO queue (single producer, single consumer).

```
#include <touchgfx/hal/DMA.hpp>
```

Public Member Functions

- LockFreeDMA_Queue (BlitOp *mem, atomic_t n)

  Constructs a lockfree queue.
- virtual bool isEmpty ()

  Query if this object is empty.
- virtual bool isFull ()

  Query if this object is full.
- virtual void pushCopyOf (const BlitOp &op)

  Push copy of the given operation.

Protected Member Functions

- virtual void pop ()

  Removes the top-of-stack object.
- virtual const BlitOp * first ()

  Gets the first blit operation.

Protected Attributes

- BlitOp * q

  Pointer to the queue memory.
- atomic_t capacity

The number of elements the queue can contain.

- atomic_t head

  Index to the head element.

- atomic_t tail

  Index to the tail element.

18.83.1 Detailed Description

This implements a simple lock-free FIFO queue (single producer, single consumer)

See also

DMA_Queue

18.83.2 Constructor & Destructor Documentation

18.83.2.1 LockFreeDMA_Queue (BlitOp * *mem*, atomic_t *n*)

Constructs a lockfree queue.

Parameters

out	*mem*	Pointer to the memory used by the queue to store elements.
	n	Number of elements the memory provided can contain.

18.83.3 Member Function Documentation

18.83.3.1 const BlitOp * first ( ) [protected], [virtual]

Returns

the first blitop.

Implements DMA_Queue.

18.83.3.2 bool isEmpty ( ) [virtual]

Query if this object is empty.

Returns

true if empty, false if not.

Implements DMA_Queue.

18.83.3.3 bool isFull ( ) [virtual]

Query if this object is full.

Returns

true if full, false if not.

Implements DMA_Queue.

18.83.3.4 void pop ( ) [protected],[virtual]

Removes the top-of-stack object.

Implements DMA_Queue.

18.83.3.5 void pushCopyOf (const BlitOp & *op*) [virtual]

Push copy of the given operation.

Parameters

op	The operation.

Implements DMA_Queue.

18.84 Matrix4x4 Class Reference

This class represents row major 4x4 homogeneous matrices.

`#include <C:\Projects\devel\touchgfx\framework\include\touchgfx\Math3D.hpp>`

Public Member Functions

- Matrix4x4 ()

  Default constructor.
- float getElement (int row, int column) const

  Gets an element.
- void setViewDistance (float distance)

  Sets view distance.
- Matrix4x4 setElement (int row, int column, float value)

  Sets an element.
- Matrix4x4 & concatenateXRotation (float radians)

  Concatenate x coordinate rotation.
- Matrix4x4 & concatenateYRotation (float radians)

  Concatenate y coordinate rotation.
- Matrix4x4 & concatenateZRotation (float radians)

  Concatenate z coordinate rotation.
- Matrix4x4 & concatenateXTranslation (float distance)

  Concatenate x coordinate translation.
- Matrix4x4 & concatenateYTranslation (float distance)

  Concatenate y coordinate translation.
- Matrix4x4 & concatenateZTranslation (float distance)

  Concatenate z coordinate translation.
- Matrix4x4 & concatenateXScale (float distance)

  Concatenate x coordinate scale.
- Matrix4x4 & concatenateYScale (float distance)

  Concatenate y coordinate scale.
- Matrix4x4 & concatenateZScale (float distance)

  Concatenate z coordinate scale.

Static Public Member Functions

- static Matrix4x4 identity ()

  Gets the identity.

Protected Attributes

- float elements [4][4]

  The elements[4][4].

18.84.1 Detailed Description

This class represents row major 4x4 homogeneous matrices.

18.84.2 Constructor & Destructor Documentation

18.84.2.1 Matrix4x4 ( )

Default constructor.

18.84.3 Member Function Documentation

18.84.3.1 Matrix4x4 & concatenateXRotation (float *radians*)

Concatenate x coordinate rotation.

Parameters

radians	The radians.

Returns

A matrix_4x4&

18.84.3.2 Matrix4x4 & concatenateXScale (float *distance*)

Concatenate x coordinate scale.

Parameters

distance	The distance.

Returns

A matrix_4x4&

18.84.3.3 Matrix4x4 & concatenateXTranslation (float *distance*)

Concatenate x coordinate translation.

Parameters

distance	The distance.

Returns

A matrix_4x4&

18.84.3.4 Matrix4x4 & concatenateYRotation (float *radians*)

Concatenate y coordinate rotation.

Parameters

radians	The radians.

Returns

A matrix_4x4&

18.84.3.5 Matrix4x4 & concatenateYScale (float *distance*)

Concatenate y coordinate scale.

Parameters

distance	The distance.

Returns

A matrix_4x4&

18.84.3.6 Matrix4x4 & concatenateYTranslation (float *distance*)

Concatenate y coordinate translation.

Parameters

distance	The distance.

Returns

A matrix_4x4&

18.84.3.7 Matrix4x4 & concatenateZRotation (float *radians*)

Concatenate z coordinate rotation.

Parameters

radians	The radians.

Returns

A matrix_4x4&

18.84.3.8 Matrix4x4 & concatenateZScale (float *distance*)

Concatenate z coordinate scale.

Parameters

distance	The distance.

Returns

A matrix_4x4&

18.84.3.9 Matrix4x4 & concatenateZTranslation (float *distance*)

Concatenate z coordinate translation.

Parameters

distance	The distance.

Returns

A matrix_4x4&

18.84.3.10 float getElement (int *row,* int *column*) const [inline]

Gets an element.

Parameters

row	The row.
column	The column.

Returns

The element.

18.84.3.11 static Matrix4x4 identity ( ) [inline],[static]

Gets the identity.

Returns

A matrix_4x4.

18.84.3.12 Matrix4x4 setElement (int *row,* int *column,* float *value*) [inline]

Sets an element.

Parameters

row	The row.
column	The column.
value	The value.

Returns

A matrix_4x4&

18.84.3.13 void setViewDistance (float *distance*)

Sets view distance.

Parameters

distance	The distance.

18.85 MCUInstrumentation Class Reference

Interface for instrumenting processors to measure MCU load via measured CPU cycles.

`#include <platform/core/MCUInstrumentation.hpp>`

Public Member Functions

- MCUInstrumentation ()

  Constructor.
- virtual void init ()=0

  Initialize.
- virtual ~MCUInstrumentation ()

  Virtual destructor.
- virtual unsigned int getElapsedUS (unsigned int start, unsigned int now, unsigned int clockfrequency)=0

  Gets elapsed microseconds based on clock frequency.
- virtual unsigned int getCPUCycles (void)=0

  Gets CPU cycles from register.
- virtual void setMCUActive (bool active)

  Sets MCU activity high.
- virtual uint32_t getCCConsumed ()

  Gets number of consumed clock cycles.
- virtual void setCCConsumed (uint32_t val)

  Sets number of consumed clock cycles.

Protected Attributes

- volatile uint32_t cc_consumed

  Amount of consumed CPU cycles.
- volatile uint32_t cc_in

  Current CPU cycles.

18.85.1 Detailed Description

Interface for instrumenting processors to measure MCU load via measured CPU cycles.

18.85.2 Constructor & Destructor Documentation

18.85.2.1 MCUInstrumentation ( ) [inline]

Constructor. Initializes members.

18.85.2.2 ~MCUInstrumentation ( ) [inline],[virtual]

Virtual destructor.

18.85.3 Member Function Documentation

18.85.3.1 uint32_t getCCConsumed ( ) [inline],[virtual]

Gets number of consumed clock cycles.

Returns

clock cycles.

18.85.3.2 unsigned int getCPUCycles (void) [pure virtual]

Gets CPU cycles from register.

Returns

CPU cycles.

Implemented in CortexMMCUInstrumentation.

18.85.3.3 unsigned int getElapsedUS (unsigned int *start*, unsigned int *now*, unsigned int *clockfrequency*) [pure virtual]

Gets elapsed microseconds based on clock frequency.

Parameters

start	Start time.
now	Current time.
clockfrequency	Clock frequency of the system expressed in MHz.

Returns

Elapsed microseconds start and now.

Implemented in CortexMMCUInstrumentation.

18.85.3.4 void init ( ) [pure virtual]

Initialize.

Implemented in CortexMMCUInstrumentation.

18.85.3.5 void setCCConsumed (uint32_t *val*) [inline],[virtual]

Sets number of consumed clock cycles.

Parameters

val	number of clock cycles.

18.85.3.6 void setMCUActive (bool *active*) [inline],[virtual]

Sets MCU activity high.

Parameters

active	if True, inactive otherwise.

Reimplemented in CortexMMCUInstrumentation.

18.86 MoveAnimator< T > Class Template Reference

A MoveAnimator makes the template class T able to animate a movement.

```
#include <touchgfx/mixins/MoveAnimator.hpp>
```

Public Member Functions

- MoveAnimator ()

  Default constructor.
- virtual ~MoveAnimator ()

  Destructor.
- void setMoveAnimationEndedAction (GenericCallback< const MoveAnimator< T > & > &callback)

  Associates an action to be performed when the animation ends.
- virtual void setMoveAnimationDelay (uint16_t delay)

  Sets a delay on animations done by the MoveAnimator.
- virtual uint16_t getMoveAnimationDelay () const

  Gets the current animation delay.
- virtual bool isRunning () const

  Gets whether or not the move animation is running.
- void startMoveAnimation (int16_t endX, int16_t endY, uint16_t duration, EasingEquation xProgressionEquation=&EasingEquations::linearEaseNone, EasingEquation yProgressionEquation=&EasingEquations::linearEaseNone)

  Starts the move animation.
- void cancelMoveAnimation ()

  Cancel move animation.

Protected Member Functions

- virtual void handleTickEvent ()

  The tick handler that handles the actual animation steps.

Protected Attributes

- bool moveAnimationRunning

  Boolean that is true if the animation is running.
- uint16_t moveAnimationCounter

  Counter that is equal to the current step in the animation.
- uint16_t moveAnimationDelay

  A delay that is applied before animation start. Expressed in ticks.
- uint16_t moveAnimationDuration

  The complete duration of the animation. Expressed in ticks.
- int16_t moveAnimationStartX

  The X value at the beginning of the animation.
- int16_t moveAnimationStartY

The Y value at the beginning of the animation.

- int16_t moveAnimationEndX

  The X value at the end of the animation.

- int16_t moveAnimationEndY

  The Y value at the end of the animation.

- EasingEquation moveAnimationXEquation

  EasingEquation expressing the development of the X value during the animation.

- EasingEquation moveAnimationYEquation

  EasingEquation expressing the development of the Y value during the animation.

- GenericCallback< const MoveAnimator< T > & > * moveAnimationEndedCallback

  Animation ended Callback.

18.86.1 Detailed Description

template<class T>class touchgfx::MoveAnimator< T >

A MoveAnimator makes the template class T able to animate a movement from its current position to a specified end position. The movement in both the X and Y direction can be described by supplying EasingEquations. The MoveAnimator performs a callback when the animation has finished.

This mixin can be used on any Drawable.

Template Parameters

T	Specifies the type should have the move animation capability.

18.86.2 Constructor & Destructor Documentation

18.86.2.1 MoveAnimator ( ) [inline]

Default constructor. Creates and initialize the MoveAnimator.

18.86.2.2 ~MoveAnimator ( ) [inline],[virtual]

Destructor. Destroys the MoveAnimator.

18.86.3 Member Function Documentation

18.86.3.1 void cancelMoveAnimation ( ) [inline]

Cancel move animation.

18.86.3.2 uint16_t getMoveAnimationDelay ( ) const [inline],[virtual]

Gets the current animation delay.

Returns

The current animation delay.

18.86.3.3 void handleTickEvent ( ) [inline], [protected], [virtual]

The tick handler that handles the actual animation steps.

18.86.3.4 bool isRunning ( ) const [inline], [virtual]

Gets whether or not the move animation is running.

Returns

true if the move animation is running.

18.86.3.5 void setMoveAnimationDelay (uint16_t *delay*) [inline], [virtual]

Sets a delay on animations done by the MoveAnimator.

Parameters

delay	The delay in ticks.

18.86.3.6 void setMoveAnimationEndedAction (GenericCallback< const MoveAnimator< T > & > & *callback*) [inline]

Associates an action to be performed when the animation ends.

Parameters

callback	The callback to be executed. The callback will be given a reference to the MoveAnimator.

See also

GenericCallback

18.86.3.7 void startMoveAnimation (int16_t *endX*, int16_t *endY*, uint16_t *duration*, EasingEquation *xProgressionEquation* = &EasingEquations::linearEaseNone, EasingEquation *yProgressionEquation* = &EasingEquations::linearEaseNone) [inline]

Starts the move animation from the current position to the specified end position. The development of the position (X, Y) during the animation is described by the supplied EasingEquations.

Parameters

endX	The X position of T at animation end. Relative to the container or view that holds T.
endY	The Y position of T at animation end. Relative to the container or view that holds T.
duration	The duration of the animation measured in ticks.
xProgression↩ Equation	The equation that describes the development of the X position during the animation. Default = EasingEquations::linearEaseNone.
yProgression↩ Equation	The equation that describes the development of the Y position during the animation. Default = EasingEquations::linearEaseNone.

18.87 MVPApplication Class Reference

A specialization of the TouchGFX Application class.

```
#include <mvp\MVPApplication.hpp>
```

Public Member Functions

- MVPApplication ()

  Default constructor.
- virtual ~MVPApplication ()

  Destructor.
- virtual void handlePendingScreenTransition ()

  Handles the pending screen transition.

Protected Member Functions

- void evaluatePendingScreenTransition ()

  Evaluates the pending Callback instances.

Protected Attributes

- Presenter * currentPresenter

  Pointer to the currently active presenter.
- GenericCallback * pendingScreenTransitionCallback

  Callback for screen transitions. Will be set to something valid when a transition request is made.

Additional Inherited Members

18.87.1 Detailed Description

A specialization of the TouchGFX Application class that provides the necessary glue for transitioning between presenter/view pairs.

It maintains a callback for transitioning and evaluates this at each tick.

See also

Application

18.87.2 Constructor & Destructor Documentation

18.87.2.1 MVPApplication ( ) [inline]

Default constructor.

18.87.2.2 ~MVPApplication ( ) [inline], [virtual]

Destructor.

18.87.3 Member Function Documentation

18.87.3.1 void evaluatePendingScreenTransition ( ) [inline], [protected]

Evaluates the pending Callback instances. If a callback is valid, it is executed and a Screen transition is executed.

18.87.3.2 void handlePendingScreenTransition ( ) [inline],[virtual]

Delegates the work to evaluatePendingScreenTransition()

Reimplemented from Application.

18.88 MVPHeap Class Reference

Generic heap class for MVP applications.

`#include <mvp/MVPHeap.hpp>`

Public Member Functions

- MVPHeap (AbstractPartition &pres, AbstractPartition &scr, AbstractPartition &tra, MVPApplication &app)

  Constructor.
- virtual ~MVPHeap ()

  Destructor.

Public Attributes

- AbstractPartition & presenterStorage

  A memory partition containing enough memory to hold the largest presenter.
- AbstractPartition & screenStorage

  A memory partition containing enough memory to hold the largest view.
- AbstractPartition & transitionStorage

  A memory partition containing enough memory to hold the largest transition.
- MVPApplication & frontendApplication

  A reference to the MVPApplication instance.

18.88.1 Detailed Description

Generic heap class for MVP applications. Serves as a way of obtaining the memory storage areas for presenters, screens, transitions and the concrete application.

Subclassed by an application-specific heap which provides the actual storage areas. This generic interface is used only in makeTransition.

18.88.2 Constructor & Destructor Documentation

18.88.2.1 MVPHeap (AbstractPartition & *pres*, AbstractPartition & *scr*, AbstractPartition & *tra*, MVPApplication & *app*) [inline]

Constructs an MVPHeap.

Parameters

in	*pres*	A memory partition containing enough memory to hold the largest presenter.
in	*scr*	A memory partition containing enough memory to hold the largest view.

in	*tra*	A memory partition containing enough memory to hold the largest transition.
in	*app*	A reference to the MVPApplication instance.

18.88.2.2 ~MVPHeap ( ) [inline],[virtual]

Destructor.

18.89 NoDMA Class Reference

This is an "empty" DMA subclass that does nothing except assert if accidentally used.

`#include <touchgfx/hal/NoDMA.hpp>`

Public Member Functions

- NoDMA ()

  Default constructor.
- virtual BlitOperations getBlitCaps ()

  No blit operations supported by this DMA implementation.
- virtual void setupDataCopy (const BlitOp &blitOp)

  Asserts if used.
- virtual void setupDataFill (const BlitOp &blitOp)

  Asserts if used.
- virtual void signalDMAInterrupt ()

  Does nothing.
- virtual void flush ()

  Does nothing.

Additional Inherited Members

18.89.1 Detailed Description

This is an "empty" DMA subclass that does nothing except assert if accidentally used. An instance of this object can be used if DMA support is not desired.

See also

DMA_Interface

18.89.2 Constructor & Destructor Documentation

18.89.2.1 NoDMA ( ) [inline]

Constructs a NoDMA object, with a queue of 1 element.

18.89.3 Member Function Documentation

18.89.3.1 void flush ( ) [inline],[virtual]

Block until all DMA transfers are complete. Since this particular DMA does not do anything, return immediately.

Reimplemented from DMA_Interface.

18.89.3.2 BlitOperations getBlitCaps ( ) [inline],[virtual]

No blit operations supported by this DMA implementation.

Returns

Zero (no blit ops supported).

Implements DMA_Interface.

18.89.3.3 void setupDataCopy (const BlitOp & *blitOp*) [inline],[virtual]

Asserts if used.

Parameters

blitOp	The blit operation to be performed by this DMA instance.

Implements DMA_Interface.

18.89.3.4 void setupDataFill (const BlitOp & *blitOp*) [inline],[virtual]

Asserts if used.

Parameters

blitOp	The blit operation to be performed by this DMA instance.

Implements DMA_Interface.

18.89.3.5 void signalDMAInterrupt ( ) [inline],[virtual]

Does nothing.

Implements DMA_Interface.

18.90 NoTouchController Class Reference

```
#include <NoTouchController.hpp>
```

Public Member Functions

- virtual void init ()

  Empty initialization.
- virtual ~NoTouchController ()

  Destructor.
- virtual bool sampleTouch (int32_t &x, int32_t &y)

  Does nothing.

18.90.1 Detailed Description

Empty TouchController implementation which does nothing. Use this if your display does not have touch input capabilities.

18.90.2 Constructor & Destructor Documentation

18.90.2.1 ~NoTouchController ( ) [inline], [virtual]

Destructor.

18.90.3 Member Function Documentation

18.90.3.1 void init ( ) [inline], [virtual]

Empty initialization.

Implements TouchController.

18.90.3.2 bool sampleTouch (int32_t & *x*, int32_t & *y*) [inline], [virtual]

Parameters

out	*x*	unused.
out	*y*	unused.

Returns

false.

Implements TouchController.

18.91 NoTransition Class Reference

The most simple Transition without any visual effects.

`#include <touchgfx/transitions/NoTransition.hpp>`

Public Member Functions

- NoTransition ()

  Default constructor.
- virtual ~NoTransition ()

  Destructor.
- virtual void handleTickEvent ()

  Indicates that the transition is done after the first tick.

Additional Inherited Members

18.91.1 Detailed Description

The most simple Transition without any visual effects.

See also

Transition

18.91.2 Constructor & Destructor Documentation

18.91.2.1 NoTransition ( ) [inline]

Default constructor.

18.91.2.2 ~NoTransition ( ) [inline],[virtual]

Destructor.

18.91.3 Member Function Documentation

18.91.3.1 void handleTickEvent ( ) [inline],[virtual]

Indicates that the transition is done after the first tick.

Reimplemented from Transition.

18.92 OSWrappers Class Reference

This class specifies OS wrappers for dealing with the frame buffer semaphore and the VSYNC signal.

`#include <touchgfx/hal/OSWrappers.hpp>`

Static Public Member Functions

- static void initialize ()

  Initialize frame buffer semaphore and queue/mutex for VSYNC signal.
- static void signalVSync ()

  Signal that a VSYNC has occurred.
- static void waitForVSync ()

  This function blocks until a VSYNC occurs.
- static void takeFrameBufferSemaphore ()

  Take the frame buffer semaphore.
- static void tryTakeFrameBufferSemaphore ()

  Attempt to obtain the frame buffer semaphore.
- static void giveFrameBufferSemaphore ()

  Release the frame buffer semaphore.
- static void giveFrameBufferSemaphoreFromISR ()

  Release the frame buffer semaphore in a way that is safe in interrupt context. Called from ISR.

18.92.1 Detailed Description

This class specifies OS wrappers for dealing with the frame buffer semaphore and the VSYNC signal.

18.92.2 Member Function Documentation

18.92.2.1 static void giveFrameBufferSemaphore ( ) [static]

Release the frame buffer semaphore.

18.92.2.2 static void giveFrameBufferSemaphoreFromISR ( ) [static]

Release the frame buffer semaphore in a way that is safe in interrupt context. Called from ISR.

18.92.2.3 static void initialize ( ) [static]

Initialize frame buffer semaphore and queue/mutex for VSYNC signal.

18.92.2.4 static void signalVSync ( ) [static]

Signal that a VSYNC has occurred. Should make the vsync queue/mutex available.

Note

This function is called from an ISR, and should (depending on OS) trigger a scheduling.

18.92.2.5 static void takeFrameBufferSemaphore ( ) [static]

Take the frame buffer semaphore. Blocks until semaphore is available.

18.92.2.6 static void tryTakeFrameBufferSemaphore ( ) [static]

Attempt to obtain the frame buffer semaphore. If semaphore is not available, do nothing.

Note

must return immediately! This function does not care who has the taken the semaphore, it only serves to make sure that the semaphore is taken by someone.

18.92.2.7 static void waitForVSync ( ) [static]

This function blocks until a VSYNC occurs.

Note

This function must first clear the mutex/queue and then wait for the next one to occur.

18.93 Outline Class Reference

An internal class that implements the main rasterization algorithm.

```
#include <touchgfx/canvas_widget_renderer/Outline.hpp>
```

Public Types

- typedef unsigned int OutlineFlags_t

  Defines an alias representing the outline flags.

Public Member Functions

- Outline ()

  Default constructor.
- ~Outline ()

  Destructor.
- void reset ()

  Resets this object.
- void moveTo (int x, int y)

  Move a virtual pen to the specified coordinate.
- void lineTo (int x, int y)

  Create a line from the current virtual pen coordinate to the given coordinate creating an Outline.
- unsigned getNumCells () const

  Gets number cells registered in the current drawn path for the Outline.
- const Cell * getCells ()

  Gets a pointer to the the Cell objects in the Outline.
- void setMaxRenderY (int y)

  Sets maximum render y coordinate.
- bool wasOutlineTooComplex ()

  Determines if there was enough memory to register the entire outline.

Static Public Attributes

- static const OutlineFlags_t OUTLINE_NOT_CLOSED = 1U

  If this bit is set in flags, the current Outline has not yet been closed. Used for automatic closing an Outline before rendering the Outline.
- static const OutlineFlags_t OUTLINE_SORT_REQUIRED = 2U

  If this bit is set in flags, Cell objects have been added to the Outline requiring the Cell list needs to be sorted.

18.93.1 Detailed Description

An internal class that implements the main rasterization algorithm. Used in the Rasterizer. Should not be used direcly.

18.93.2 Member Typedef Documentation

18.93.2.1 unsigned int OutlineFlags_t

Defines an alias representing the outline flags.

18.93.3 Constructor & Destructor Documentation

18.93.3.1 Outline ( )

Default constructor.

18.93.3.2 ~Outline ( )

Destructor.

18.93.4 Member Function Documentation

18.93.4.1 const Cell * getCells ( )

Gets a pointer to the the Cell objects in the Outline. If the Outline is not closed, it is closed. If the Outline is unsorted, it will be quick sorted first.

Returns

A pointer to the sorted list of Cell objects in the Outline.

18.93.4.2 unsigned getNumCells ( ) const [inline]

Gets number cells registered in the current drawn path for the Outline.

Returns

The number of cells.

18.93.4.3 void lineTo (int *x*, int *y*)

Create a line from the current virtual pen coordinate to the given coordinate creating an Outline.

Parameters

x	The x coordinate.
y	The y coordinate.

18.93.4.4 void moveTo (int *x*, int *y*)

Move a virtual pen to the specified coordinate.

Parameters

x	The x coordinate.
y	The y coordinate.

18.93.4.5 void reset ( )

Resets this object. This implies removing the current Cell objects and preparing for a new Outline.

18.93.4.6 void setMaxRenderY (int *y*) [inline]

Sets maximum render y coordinate. This is used to avoid registering any Cell that has a y coordinate less than zero of higher than the given y.

Parameters

y	The max y coordinate to render for the Outline.

18.93.4.7 bool wasOutlineTooComplex ( ) [inline]

Determines if there was enough memory to register the entire outline, of if the outline was too complex.

Returns

false if the buffer for Outline Cell objects was too small.

18.94 PainterBW Class Reference

A Painter that will paint using a color on a LCD1bpp display.

```
#include <touchgfx/widgets/canvas/PainterBW.hpp>
```

Public Member Functions

- void setColor (uint8_t color)

  Sets color to use when drawing the CanvasWidget.

Static Public Member Functions

- static unsigned bw (unsigned red, unsigned green, unsigned blue)

  Convert color to black/white.

Protected Member Functions

- virtual bool renderNext (uint8_t &color)

  Get the color of the next pixel in the scan line.

Protected Attributes

- uint8_t painterColor

  The color to use when painting.

18.94.1 Detailed Description

PainterBW is used for drawing one 1bpp displays. The color is either on or off No transparency is supported.

See also

AbstractPainter

18.94.2 Member Function Documentation

18.94.2.1 static unsigned bw (unsigned *red*, unsigned *green*, unsigned *blue*) [static]

Converts the selected color to either white (1) or black (0) depending on the converted gray value.

Parameters

red	The red color.
green	The green color.

blue	The blue color.

Returns

1 (white) if the brightness of the RGB color is above 50% and 0 (black) otherwise.

18.94.2.2 virtual bool renderNext (uint8_t & *color*) [protected], [virtual]

Get the color of the next pixel in the scan line.

Parameters

out	*color*	Color of the pixel, 0 or 1.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterBW.

18.94.2.3 void setColor (uint8_t *color*)

Sets color to use when drawing the CanvasWidget.

Parameters

color	The color, 0=black, otherwise white.

18.95 PainterRGB565 Class Reference

A Painter that will paint using a color and an alpha value.

```
#include <touchgfx/widgets/canvas/PainterRGB565.hpp>
```

Public Member Functions

- PainterRGB565 (colortype color=0, uint8_t alpha=255)

  Constructor.
- void setColor (colortype color, uint8_t alpha=255)

  Sets color and alpha to use when drawing the CanvasWidget.

Protected Member Functions

- virtual bool renderNext (uint8_t &red, uint8_t &green, uint8_t &blue, uint8_t &alpha)

  Get the color of the next pixel in the scan line.

Protected Attributes

- uint8_t painterRed

  The red part of the color.
- uint8_t painterGreen

  The green part of the color.

- uint8_t painterBlue

  The blue part of the color.
- uint8_t painterAlpha

  The alpha value.

18.95.1 Detailed Description

The PainterRGB565 class allows a shape to be filled with a given color and alpha value. This allows transparent, anti-aliased elements to be drawn.

See also

AbstractPainter

18.95.2 Constructor & Destructor Documentation

18.95.2.1 PainterRGB565 (colortype *color* = 0, uint8_t *alpha* = 255)

Constructor.

Parameters

color	the color.
alpha	the alpha.

18.95.3 Member Function Documentation

18.95.3.1 virtual bool renderNext (uint8_t & *red*, uint8_t & *green*, uint8_t & *blue*, uint8_t & *alpha*) [protected], [virtual]

Get the color of the next pixel in the scan line.

Parameters

out	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB565.

18.95.3.2 void setColor (colortype *color*, uint8_t *alpha* = 255)

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.

alpha	The alpha.

18.96 PainterRGB565Bitmap Class Reference

A Painter that will paint using a bitmap.

`#include <touchgfx/widgets/canvas/PainterRGB565Bitmap.hpp>`

Public Member Functions

- PainterRGB565Bitmap (const Bitmap &bmp=Bitmap(BITMAP_INVALID), uint8_t alpha=255)

  Constructor.
- void setBitmap (const Bitmap &bmp)

  Sets a bitmap to be used when drawing the CanvasWidget.
- void setAlpha (uint8_t alpha)

  Sets an alpha value for the bitmap.

Protected Member Functions

- virtual bool renderInit ()

  Initialize rendering of a single scan line of pixels for the render.
- virtual bool renderNext (uint8_t &red, uint8_t &green, uint8_t &blue, uint8_t &alpha)

  Get the color of the next pixel in the scan line.

Protected Attributes

- const uint32_t * bitmapARGB8888Pointer

  Pointer to the bitmap (ARGB8888)
- const uint16_t * bitmapRGB565Pointer

  Pointer to the bitmap (RGB565)
- const uint8_t * bitmapAlphaPointer

  Pointer to the bitmap alpha data for RGB565.
- Bitmap bitmap

  The bitmap to be used when painting.
- Rect bitmapRectToFrameBuffer

  Bitmap rectangle translated to frame buffer coordinates.
- uint8_t painterAlpha

  The alpha to use if no alpha data is present in the given bitmap.

18.96.1 Detailed Description

PainterRGB565Bitmap will take the color for a given point in the shape from a bitmap. Please be aware, the the bitmap is used by the CWR (not Shape), so any rotation you might specify for a Canvas Widget (e.g. Shape) is not applied to the bitmap as CWR is not aware of this rotation.

See also

AbstractPainter

18.96.2 Constructor & Destructor Documentation

18.96.2.1 PainterRGB565Bitmap (const Bitmap & *bmp* = Bitmap (BITMAP_INVALID), uint8_t *alpha* = 255)

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

18.96.3 Member Function Documentation

18.96.3.1 virtual bool renderInit ( ) [protected], [virtual]

Initialize rendering of a single scan line of pixels for the render.

Returns

true if it succeeds, false if it fails.

Reimplemented from AbstractPainterRGB565.

18.96.3.2 virtual bool renderNext (uint8_t & *red*, uint8_t & *green*, uint8_t & *blue*, uint8_t & *alpha*) [protected], [virtual]

Get the color of the next pixel in the scan line.

Parameters

out	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB565.

18.96.3.3 void setAlpha (uint8_t *alpha*)

Sets an alpha value for the bitmap. If the image contains an alpha channel, this alpha value is combined with the alpha in the bitmap to produce the final alpha value.

Parameters

alpha	The alpha value to use if there is no alpha channel in the bitmap.

18.96.3.4 void setBitmap (const Bitmap & *bmp*)

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

18.97 PainterRGB888 Class Reference

A Painter that will paint using a color and an alpha value.

```
#include <touchgfx/widgets/canvas/PainterRGB888.hpp>
```

Public Member Functions

- PainterRGB888 (colortype color=0, uint8_t alpha=255)

  Constructor.
- void setColor (colortype color, uint8_t alpha=255)

  Sets color and alpha to use when drawing the CanvasWidget.

Protected Member Functions

- virtual bool renderNext (uint8_t &red, uint8_t &green, uint8_t &blue, uint8_t &alpha)

  Get the color of the next pixel in the scan line.

Protected Attributes

- uint8_t painterRed

  The red part of the color.
- uint8_t painterGreen

  The green part of the color.
- uint8_t painterBlue

  The blue part of the color.
- uint8_t painterAlpha

  The alpha value.

18.97.1 Detailed Description

The PainterRGB888 class allows a shape to be filled with a given color and alpha value. This allows transparent, anti-aliased elements to be drawn.

See also

AbstractPainter

18.97.2 Constructor & Destructor Documentation

18.97.2.1 PainterRGB888 (colortype *color* = 0, uint8_t *alpha* = 255)

Constructor.

Parameters

color	the color.
alpha	the alpha.

18.97.3 Member Function Documentation

18.97.3.1 virtual bool renderNext (uint8_t & *red*, uint8_t & *green*, uint8_t & *blue*, uint8_t & *alpha*) [protected], [virtual]

Get the color of the next pixel in the scan line.

Parameters

out	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB888.

18.97.3.2 void setColor (colortype *color*, uint8_t *alpha* = 255)

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

18.98 PainterRGB888Bitmap Class Reference

A Painter that will paint using a bitmap.

```
#include <touchgfx/widgets/canvas/PainterRGB888Bitmap.hpp>
```

Public Member Functions

- PainterRGB888Bitmap (const Bitmap &bmp=Bitmap(BITMAP_INVALID), uint8_t alpha=255)

  Constructor.
- void setBitmap (const Bitmap &bmp)

  Sets a bitmap to be used when drawing the CanvasWidget.
- void setAlpha (uint8_t alpha)

  Sets an alpha value for the bitmap.

Protected Member Functions

- virtual bool renderInit ()

  Initialize rendering of a single scan line of pixels for the render.
- virtual bool renderNext (uint8_t &red, uint8_t &green, uint8_t &blue, uint8_t &alpha)

  Get the color of the next pixel in the scan line.

Protected Attributes

- const uint32_t * bitmapARGB8888Pointer

  Pointer to the bitmap (ARGB8888)
- const uint8_t * bitmapRGB888Pointer

  Pointer to the bitmap (RGB888)
- Bitmap bitmap

  The bitmap to be used when painting.

- Rect bitmapRectToFrameBuffer

  Bitmap rectangle translated to frame buffer coordinates.
- uint8_t painterAlpha

  The alpha to use if no alpha data is present in the given bitmap.

18.98.1 Detailed Description

PainterRGB888Bitmap will take the color for a given point in the shape from a bitmap. Please be aware, the the bitmap is used by the CWR (not Shape), so any rotation you might specify for a Canvas Widget (e.g. Shape) is not applied to the bitmap as CWR is not aware of this rotation.

See also

AbstractPainter

18.98.2 Constructor & Destructor Documentation

18.98.2.1 PainterRGB888Bitmap (const Bitmap & *bmp* = Bitmap (BITMAP_INVALID) , uint8_t *alpha* = 255)

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

18.98.3 Member Function Documentation

18.98.3.1 virtual bool renderInit ( ) [protected], [virtual]

Initialize rendering of a single scan line of pixels for the render.

Returns

true if it succeeds, false if it fails.

Reimplemented from AbstractPainterRGB888.

18.98.3.2 virtual bool renderNext (uint8_t & *red*, uint8_t & *green*, uint8_t & *blue*, uint8_t & *alpha*) [protected], [virtual]

Get the color of the next pixel in the scan line.

Parameters

out	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB888.

18.98.3.3 void setAlpha (uint8_t *alpha*)

Sets an alpha value for the bitmap. If the image contains an alpha channel, this alpha value is combined with the alpha in the bitmap to produce the final alpha value.

Parameters

alpha	The alpha value to use if there is no alpha channel in the bitmap.

18.98.3.4 void setBitmap (const Bitmap & *bmp*)

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

18.99 Pair< T1, T2 > Struct Template Reference

A simple struct for holding pairs of data.

`#include <touchgfx/hal/Types.hpp>`

Public Member Functions

- Pair ()

  Default constructor.
- Pair (const T1 &x, const T2 &y)

  Constructor initializing the elements it holds using their copy constructor.
- template<class U , class V >
  Pair (const Pair< U, V > &p)

  Copy constructor.

Public Attributes

- T1 first

  The first element.
- T2 second

  The second element.

18.99.1 Detailed Description

template<class T1, class T2>struct touchgfx::Pair< T1, T2 >

A simple struct for holding pairs of data.

Template Parameters

T1	The type of the first element.
T2	The type of the second element.

18.99.2 Constructor & Destructor Documentation

18.99.2.1 Pair ( ) [inline]

Constructor initializing the elements it holds using their default constructors.

18.99.2.2 Pair (const T1 & *x*, const T2 & *y*) [inline]

Constructor initializing the elements it holds using their copy constructor.

Parameters

x	Reference to the first element.
y	Reference to the second element.

18.99.2.3 Pair (const Pair< U, V > & *p*) [inline]

Copy constructor.

Template Parameters

U	Generic type parameter.
V	Generic type parameter.

Parameters

p	The pair to copy from.

18.100 Partition< ListOfTypes, NUMBER_OF_ELEMENTS > Class Template Reference

This type provides a concrete Partition of memory-slots capable of holding any of the specified list of types.

`#include <common/Partition.hpp>`

Public Types

- enum { **INTS_PR_ELEMENT** = (sizeof(typename meta::select_type_maxsize< SupportedTypesList >::type) + sizeof(int) - 1) / sizeof(int), **SIZE_OF_ELEMENT** = INTS_PR_ELEMENT ∗ sizeof(int) }
- typedef ListOfTypes SupportedTypesList

  Provides a generic public type containing the list of supported types.

Public Member Functions

- Partition ()

  Default constructor.
- virtual ∼Partition ()

  Destructor.
- virtual uint16_t capacity () const

  Specialization of AbstractPartition::capacity().
- virtual uint32_t element_size ()

  Specialization of AbstractPartition::element_size().

Protected Member Functions

- virtual void ∗ element (uint16_t index)

  Specialization of AbstractPartition::element()
- virtual const void ∗ element (uint16_t index) const

  Specialization of AbstractPartition::element() const.

18.100.1 Detailed Description

template<typename ListOfTypes, uint16_t NUMBER_OF_ELEMENTS>class touchgfx::Partition< ListOfTypes, NUMBER_OF_E↩ LEMENTS >

The Partition is not aware of the types stored in the Partition memory, hence it provides no mechanism for deleting C++ objects when the Partition is clear()'ed.

This class implements AbstractPartition.

Template Parameters

ListOfTypes	Type of the list of types.
NUMBER_OF_ELEMENTS	Type of the number of elements.

See also

AbstractPartition

18.100.2 Member Typedef Documentation

18.100.2.1 ListOfTypes SupportedTypesList

Provides a generic public type containing the list of supported types.

18.100.3 Member Enumeration Documentation

18.100.3.1 anonymous enum

Compile-time generated constants specifying the "element" or "slot" size used by this partition

18.100.4 Constructor & Destructor Documentation

18.100.4.1 Partition ( ) [inline]

Constructs an empty Partition.

18.100.5 Member Function Documentation

18.100.5.1 uint16_t capacity ( ) const [inline], [virtual]

Specialization of AbstractPartition::capacity().

Returns

An uint16_t.

See also

touchgfx::AbstractPartition::capacity()

Implements AbstractPartition.

18.100.5.2 void * element (uint16_t *index*) [inline], [protected], [virtual]

Parameters

index	Zero-based index of the.

Returns

null if it fails, else a void∗.

See also

touchgfx::AbstractPartition::element()

Implements AbstractPartition.

18.100.5.3 const void ∗ element (uint16_t *index*) const [inline], [protected], [virtual]

Parameters

index	Zero-based index of the.

Returns

null if it fails, else a void∗.

See also

touchgfx::AbstractPartition::element()

Implements AbstractPartition.

18.100.5.4 uint32_t element_size ( ) [inline], [virtual]

Specialization of AbstractPartition::element_size().

Returns

An uint32_t.

See also

touchgfx::AbstractPartition::element_size()

Implements AbstractPartition.

18.101 PixelDataWidget Class Reference

A widget for displaying a buffer of pixel data.

```
#include <touchgfx/widgets/PixelDataWidget.hpp>
```

Public Member Functions

- PixelDataWidget ()

  Default constructor.

- virtual void draw (const touchgfx::Rect &invalidatedArea) const

  Draw the part of the RAM buffer that is inside the invalidated area.
- virtual touchgfx::Rect getSolidRect () const

  Report this widget as being completely solid.
- void setPixelData (uint8_t *const data)

  Set the pixel data to display.
- void setBitmapFormat (Bitmap::BitmapFormat format)

  Set the format of the pixel data.
- void setAlpha (uint8_t alpha)

  Sets the alpha channel for the image.

Protected Attributes

- uint8_t * buffer

  The buffer where the pixels are copied from.
- Bitmap::BitmapFormat format

  The pixel format for the data.
- uint8_t alpha

  The Alpha for this widget.

Additional Inherited Members

18.101.1 Detailed Description

The buffer must be of size as widget. If the LCD is 16 bit the buffer must hold 2 bytes for each pixel. If the LCD is 24 bit the buffer must hold 3 bytes for each pixel.

See also

touchgfx::Widget

18.101.2 Constructor & Destructor Documentation

18.101.2.1 PixelDataWidget ( )

Default constructor.

18.101.3 Member Function Documentation

18.101.3.1 void draw (const touchgfx::Rect & *invalidatedArea*) const [virtual]

Draw the part of the RAM buffer that is inside the invalidated area.

Parameters

invalidatedArea	The region of this drawable that needs to be redrawn.

See also

touchgfx::Drawable

Implements Drawable.

18.101.3.2 touchgfx::Rect getSolidRect ( ) const [virtual]

Report this widget as being completely solid.

Returns

The solid rect.

Implements Drawable.

18.101.3.3 void setAlpha (uint8_t *alpha*) [inline]

Sets the alpha channel for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

18.101.3.4 void setBitmapFormat (Bitmap::BitmapFormat *format*)

Set the format of the pixel data.

Parameters

format	Describes the format to use.

18.101.3.5 void setPixelData (uint8_t *const *const *data*)

Set the pixel data to display.

Parameters

in,out	*data*	If non-null, the data.

18.102 Point Struct Reference

A simple struct containing coordinates.

`#include <touchgfx/hal/Types.hpp>`

Public Member Functions

- unsigned dist_sqr (struct Point &o)

  The squared distance from this Point to another Point.

Public Attributes

- int32_t x

  The x coordinate.
- int32_t y

  The y coordinate.

18.102.1 Member Function Documentation

18.102.1.1 unsigned dist_sqr (struct Point & *o*) [inline]

The squared distance from this Point to another Point.

Parameters

in	*o*	The point to get the squared distance to.

Returns

The squared distance.

18.103 Point3D Struct Reference

A 3D point.

`#include <touchgfx/hal/Types.hpp>`

Public Attributes

- fixed28_4 X

  The X coordinate.
- fixed28_4 Y

  The Y coordinate.
- float Z

  The Z coordinate.
- float U

  The U coordinate.
- float V

  The V coordinate.

18.104 Point4 Class Reference

This class represents a homogeneous 3D point.

`#include <touchgfx/Math3D.hpp>`

Public Member Functions

- Point4 ()

  Default constructor.
- Point4 (float x, float y, float z)

  Constructor.

Additional Inherited Members

18.104.1 Detailed Description

This class represents a homogeneous 3D point.

See also

quadruple

18.104.2 Constructor & Destructor Documentation

18.104.2.1 Point4 ( ) [inline]

Default constructor.

18.104.2.2 Point4 (float *x*, float *y*, float *z*) [inline]

Constructor.

Parameters

x	The x value.
y	The y value.
z	The z value.

18.105 PreRenderable< T > Class Template Reference

This mixin can be used on any Drawable.

```
#include <touchgfx/mixins/PreRenderable.hpp>
```

Public Member Functions

- PreRenderable ()

  Default constructor.
- void draw (const Rect &invalidatedArea) const

  Overrides the draw function.
- virtual void setupDrawChain (const Rect &invalidatedArea, Drawable **nextPreviousElement)

  Add to draw chain.
- bool isPreRendered () const

  Whether or not the snapshot of the widget been taken.
- void preRender ()

  Takes a snapshot of the current visual appearance of this widget.

18.105.1 Detailed Description

template<class T>class touchgfx::PreRenderable< T >

This mixin can be used on any Drawable. It provides a preRender function, which will cache the current visual appearance of the Drawable to be cache in a memory region. Subsequent calls to draw() on this Drawable will result in a simple memcpy of the cached memory instead of the normal draw operation. This mixin can therefore be used on Drawables whose visual appearance is static and the normal draw operation takes a long time to compute.

Note

The actual uses of this mixin are rare, and the class is mainly provided for example purposes.

Template Parameters

T	The type of Drawable to add this functionality to.

18.105.2 Constructor & Destructor Documentation

18.105.2.1 PreRenderable ( ) [inline]

Default constructor. Initializes the PreRenderable.

18.105.3 Member Function Documentation

18.105.3.1 void draw (const Rect & *invalidatedArea*) const [inline]

Overrides the draw function. If preRender has been called, perform a memcpy of the cached version. If not, just call the base class version of draw.

Parameters

invalidatedArea	The subregion of this Drawable which needs to be redrawn.

18.105.3.2 bool isPreRendered ( ) const [inline]

Returns

Is the widget rendered.

18.105.3.3 void preRender ( ) [inline]

Takes a snapshot of the current visual appearance of this widget. All subsequent calls to draw on this mixin will result in the snapshot being draw.

18.105.3.4 void setupDrawChain (const Rect & *invalidatedArea,* Drawable ** *nextPreviousElement*) [inline], [virtual]

Note

For TouchGFX internal use only.

Parameters

	invalidatedArea	Include drawables that intersect with this area only.
in,out	*nextPrevious↩ Element*	Modifiable element in linked list.

18.106 Presenter Class Reference

The Presenter base class that all application-specific presenters should derive from.

```
#include <mvp/Presenter.hpp>
```

Public Member Functions

- virtual void activate ()

Place initialization code for the Presenter here.

- virtual void deactivate ()

  Place cleanup code for the Presenter here.

- virtual ~Presenter ()

  Destructor.

Protected Member Functions

- Presenter ()

  Default constructor.

18.106.1 Detailed Description

The Presenter base class that all application-specific presenters should derive from. Only contains activate and deactivate virtual functions which are called automatically during screen transition.

18.106.2 Constructor & Destructor Documentation

18.106.2.1 ~Presenter ( ) [inline], [virtual]

Destructor.

18.106.2.2 Presenter ( ) [inline], [protected]

Default constructor.

18.106.3 Member Function Documentation

18.106.3.1 void activate ( ) [inline], [virtual]

The activate function is called automatically when a screen transition causes this Presenter to become active. Place initialization code for the Presenter here.

18.106.3.2 void deactivate ( ) [inline], [virtual]

The deactivate function is called automatically when a screen transition causes this Presenter to become inactive. Place cleanup code for the Presenter here.

18.107 CWRUtil::Q10 Class Reference

Defines a number with 10 bits reserved for fraction.

```
#include <touchgfx/widgets/canvas/CWRUtil.hpp>
```

Public Member Functions

- Q10 ()

  Default constructor.

- Q10 (int i)

Constructor from integer.

- operator int () const

  Gets the Q10 as an integer without conversion.
- Q10 operator- () const

  Negation operator.
- Q10 operator+ (const Q10 &q10) const

  Addition operator.
- Q15 operator∗ (const Q5 &q5) const

  Multiplication operator.
- Q5 operator/ (const Q5 &q5) const

  Division operator.

18.107.1 Detailed Description

Defines a number with 10 bits reserved for the fractional part of the decimal number. Q10 implements some simple arithmetic operations, most yielding a Q10 number and some yielding a Q5 number or a Q15 number as a result.

Q5∗Q5=Q10, Q10/Q5=Q5, ...

See also

Q5
Q15
http://en.wikipedia.org/wiki/Q_%28number_format%29
http://en.wikipedia.org/wiki/Fixed-point_arithmetic

18.107.2 Constructor & Destructor Documentation

18.107.2.1 Q10 ( ) [inline]

Default constructor.

18.107.2.2 Q10 (int *i*) [inline], [explicit]

Constructor from integer. No conversion is done - the integer is assumed to already be in Q10 format.

Parameters

i	int pre-formattet in Q10 format.

18.107.3 Member Function Documentation

18.107.3.1 operator int ( ) const [inline]

Gets the Q10 as an integer without conversion.

Returns

The unconverted Q10 value.

18.107.3.2 Q15 operator∗ (const Q5 & *q5*) const [inline]

Multiplication operator. The result is a Q15, not a Q10, for increased precision.

Parameters

q5	The Q5 to multiply this with.

Returns

The result of the operation.

18.107.3.3 Q10 operator+ (const Q10 & *q10*) const [inline]

Addition operator.

Parameters

q10	The Q10 to add to this.

Returns

The result of the operation.

18.107.3.4 Q10 operator- ( ) const [inline]

Negation operator.

Returns

The negative value of thisw.

18.107.3.5 Q5 operator/ (const Q5 & *q5*) const [inline]

Division operator.

Parameters

q5	The Q5 to divide this by.

Returns

The result of the operation.

18.108 CWRUtil::Q15 Class Reference

Defines a number with 15 bits reserved for fraction.

`#include <touchgfx/widgets/canvas/CWRUtil.hpp>`

Public Member Functions

- Q15 (int i)

  Constructor from integer. No conversion is done - the integer is assumed to already be in Q15 format.
- operator int () const

  Gets the Q15 as an integer without conversion.
- Q10 operator/ (const Q5 &q5) const

  Q5 / Q5 which requires the result of a Q15 / Q5 to be calculated.

18.108.1 Detailed Description

Defines a number with 15 bits reserved for the fractional part of the decimal number. Q15 is only used for sine/cosine and for intermediate calculations when multiplying.

Q5∗Q5=Q10, Q10/Q5=Q5, ...

See also

Q5
Q10
http://en.wikipedia.org/wiki/Q_%28number_format%29
http://en.wikipedia.org/wiki/Fixed-point_arithmetic

18.108.2 Constructor & Destructor Documentation

18.108.2.1 Q15 (int *i*) [inline],[explicit]

Constructor from integer. No conversion is done - the integer is assumed to already be in Q15 format.

Parameters

i	int pre-formattet in Q15 format.

18.108.3 Member Function Documentation

18.108.3.1 operator int ( ) const [inline]

Gets the Q15 as an integer without conversion.

Returns

The unconverted Q15 value.

18.108.3.2 Q10 operator/ (const Q5 & *q5*) const [inline]

Q5 / Q5 which requires the result of a Q15 / Q5 to be calculated.

Parameters

q5	The Q5 to divide this by.

Returns

The result of the operation.

18.109 CWRUtil::Q5 Class Reference

Defines a number with 5 bits reserved for fraction.

```
#include <touchgfx/widgets/canvas/CWRUtil.hpp>
```

Public Member Functions

- Q5 ()

  Default constructor.

- Q5 (int i)

  Constructor from integer.
- Q5 (const Q10 q10)

  Constructor from Q10.
- operator int () const

  Gets the Q5 as an integer without conversion.
- Q5 operator- () const

  Negation operator.
- Q5 operator+ (const Q5 &q5) const

  Addition operator.
- Q5 operator- (const Q5 &q5) const

  Subtraction operator.
- Q10 operator∗ (const Q5 &q5) const

  Multiplication operator.
- Q5 operator∗ (const Q15 &q15) const

  Multiplication operator.
- Q5 operator∗ (const int i) const

  Multiplication operator.
- Q5 operator/ (const int i) const

  Division operator.
- Q5 operator/ (const Q5 q5) const

  Division operator.
- template<class T >
  T to () const

  Converts the Q5 value to an int or a float.

18.109.1 Detailed Description

Defines a number with 5 bits reserved for the fractional part of the decimal number. Q5 implements some simple arithmetic operations, most yielding a Q5 number and some yielding a Q10 number as a result. Other operations also work with Q15 numbers.

See also

Q10
Q15
http://en.wikipedia.org/wiki/Q_%28number_format%29
http://en.wikipedia.org/wiki/Fixed-point_arithmetic

18.109.2 Constructor & Destructor Documentation

18.109.2.1 Q5 (int *i*) [inline],[explicit]

Constructor from integer. No conversion is done - the integer is assumed to already be in Q5 format.

Parameters

i	Integer pre-formattet in Q5 format.

18.109.2.2 Q5 (const Q10 *q10*) [inline]

Constructor from Q10. The Q10 is shifted down to convert it to Q5.

Parameters

q10	The Q10 value to convert to a Q5 value.

See also

Q10

18.109.3 Member Function Documentation

18.109.3.1 operator int ( ) const [inline]

Gets the Q5 as an integer without conversion.

Returns

The unconverted Q5 value.

18.109.3.2 Q10 operator∗ (const Q5 & *q5*) const [inline]

Multiplication operator. The result is a Q10, not a Q5, for increased precision.

Parameters

q5	The Q5 to multiply this with.

Returns

The result of the operation.

See also

Q10

18.109.3.3 Q5 operator∗ (const Q15 & *q15*) const [inline]

Multiplication operator. Often used in relation with sine and cosine calculation which are pre-calculated as Q15. As the result is needed as a Q5, this operator multiplies with the given Q15 and converts the result to a Q5.

Parameters

q15	The Q15 to multiply this with.

Returns

The result of the operation.

See also

Q15

18.109.3.4 Q5 operator∗ (const int *i*) const [inline]

Multiplication operator.

Parameters

i	The integer to multiply this with.

Returns

The result of the operation.

18.109.3.5 Q5 operator+ (const Q5 & *q5*) const [inline]

Addition operator.

Parameters

q5	The Q5 to add to this.

Returns

The result of the operation.

18.109.3.6 Q5 operator- ( ) const [inline]

Negation operator.

Returns

The negative value of this.

18.109.3.7 Q5 operator- (const Q5 & *q5*) const [inline]

Subtraction operator.

Parameters

q5	The Q5 to subtract from this.

Returns

The result of the operation.

18.109.3.8 Q5 operator/ (const int *i*) const [inline]

Division operator.

Parameters

i	The integer to divide this by.

Returns

The result of the operation.

18.109.3.9 Q5 operator/ (const Q5 *q5*) const [inline]

Division operator. Internally this Q5 is converted to Q10 before the division to increased precision.

Parameters

q5	The Q5 to divide this by.

Returns

The result of the operation.

See also

Q10

18.109.3.10 template< class T > T to ( ) const [inline]

Convert the Q5 value to an integer by removing the 5 bits used for the fraction, or to a floating point value by dividing by 32, depending on the type specified as T.

Template Parameters

T	Either int or float.

Returns

Q5 value as a type T.

18.110 Quadruple Class Reference

Base class for homogeneous vectors and points.

```
#include <touchgfx/Math3D.hpp>
```

Public Member Functions

- float getElement (int row) const

  Gets an element.
- float getX () const

  Get x coordinate.
- float getY () const

  Get y coordinate.
- float getZ () const

  Get z coordinate.
- float getW () const

  Gets the w.
- void setElement (int row, float value)

  Sets an element.
- void setX (float value)

  Sets an x coordinate.
- void setY (float value)

  Sets a y coordinate.
- void setZ (float value)

  Sets a z coordinate.
- void setW (float value)

  Sets a w.

Protected Member Functions

- Quadruple ()

  Default constructor.
- Quadruple (float x, float y, float z, float w)

  Constructor.

Protected Attributes

- float elements [4]

  The elements[4].

18.110.1 Detailed Description

Base class for homogeneous vectors and points.

18.110.2 Constructor & Destructor Documentation

18.110.2.1 Quadruple ( ) [inline],[protected]

Default constructor. < The elements[3]

18.110.2.2 Quadruple (float *x*, float *y*, float *z*, float *w*) [inline],[protected]

Constructor.

Parameters

x	The x value.
y	The y value.
z	The z value.
w	The w value.

18.110.3 Member Function Documentation

18.110.3.1 float getElement (int *row*) const [inline]

Gets an element.

Parameters

row	The row.

Returns

The element.

18.110.3.2 float getW ( ) const [inline]

Gets the w.

Returns

The w.

18.110.3.3 float getX ( ) const [inline]

Get x coordinate.

Returns

The x coordinate.

18.110.3.4 float getY ( ) const [inline]

Get y coordinate.

Returns

The y coordinate.

18.110.3.5 float getZ ( ) const [inline]

Get z coordinate.

Returns

The z coordinate.

18.110.3.6 void setElement (int *row*, float *value*) [inline]

Sets an element.

Parameters

row	The row.
value	The value.

18.110.3.7 void setW (float *value*) [inline]

Sets a w.

Parameters

value	The value.

18.110.3.8 void setX (float *value*) [inline]

Sets an x coordinate.

Parameters

value	The value.

18.110.3.9 void setY (float *value*) [inline]

Sets a y coordinate.

Parameters

value	The value.

18.110.3.10 void setZ (float *value*) [inline]

Sets a z coordinate.

Parameters

value	The value.

18.111 RadioButton Class Reference

Radio button with two states.

```
#include <touchgfx/widgets/RadioButton.hpp>
```

Public Member Functions

- RadioButton ()

  Default constructor.
- virtual ~RadioButton ()

  Destructor.
- virtual void draw (const Rect &invalidatedArea) const

  Draws the given invalidated area.
- virtual void handleClickEvent (const ClickEvent &event)

  Updates the current state of the radio button.
- virtual void setBitmaps (const Bitmap &bmpUnselected, const Bitmap &bmpUnselectedPressed, const Bitmap &bmpSelected, const Bitmap &bmpSelectedPressed)

  Sets the bitmaps used by this button.
- void setDeselectedAction (GenericCallback< const AbstractButton & > &callback)

  Associates an action to be performed when the AbstractButton is deselected.
- virtual Rect getSolidRect () const

  Gets solid rectangle.
- void setAlpha (uint8_t alpha)

  Sets the alpha channel for the image.
- uint8_t getAlpha () const

  Gets the current alpha value.
- void setDeselectionEnabled (bool state)

  States whether or not it is possible to de-select the RadioButton by clicking it.
- bool getDeselectionEnabled () const

  Gets the current deselectionEnabled state.
- void setSelected (bool newSelected)

  Sets the radio buttons selected state.
- bool getSelected () const

  Gets the current selected state.
- Bitmap getCurrentlyDisplayedBitmap () const

  Gets currently displayed bitmap.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- Bitmap bitmapUnselected

  The image to display when radio button unselected.
- Bitmap bitmapUnselectedPressed

  The image to display when radio button unselected and pressed.
- Bitmap bitmapSelected

  The image to display when radio button selected.
- Bitmap bitmapSelectedPressed

  The image to display when radio button selected and pressed.
- uint8_t alpha

  The current alpha value. 255 denotes solid, 0 denotes completely transparent.
- bool selected

  The current selected state.
- bool deselectionEnabled

  Is de-selecting a selected radio button by clicking it enabled.
- GenericCallback< const AbstractButton & > * deselectedAction

  The callback to be executed when this AbstractButton is unselected.

Additional Inherited Members

18.111.1 Detailed Description

A radio button consists of four images, one for its not selected and one for selected. Each of these have an image for a pressed state. RadioButtons can be added to a RadioButtonGroup which handles the de-selection of radio buttons when a new selection is made.

See also

AbstractButton

18.111.2 Constructor & Destructor Documentation

18.111.2.1 RadioButton ( ) [inline]

Default constructor.

18.111.2.2 ~RadioButton ( ) [inline],[virtual]

Destructor.

18.111.3 Member Function Documentation

18.111.3.1 void draw (const Rect & *invalidatedArea*) const [virtual]

Draws the given invalidated area.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Implements Drawable.

18.111.3.2 uint8_t getAlpha ( ) const [inline]

Gets the current alpha value.

Returns

The current alpha value.

18.111.3.3 Bitmap getCurrentlyDisplayedBitmap ( ) const [inline]

Function to obtain the currently displayed bitmap, which depends on the radio button's pressed and selected state.

Returns

The bitmap currently displayed.

18.111.3.4 bool getDeselectionEnabled ( ) const [inline]

Gets the current deselectionEnabled state.

Returns

The current deselectionEnabled state.

18.111.3.5 bool getSelected ( ) const [inline]

Gets the current selected state.

Returns

The current selected state.

18.111.3.6 Rect getSolidRect ( ) const [virtual]

Gets solid rectangle.

Returns

largest possible solid rect. Delegated to the largest solid rect of the radio button bitmap(s).

Implements Drawable.

18.111.3.7 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_RADIOBUTTON.

Reimplemented from AbstractButton.

18.111.3.8 void handleClickEvent (const ClickEvent & *event*) [virtual]

Updates the current state of the radio button - pressed or released, selected or not selected - and invalidates it.

If a transition from the not selected to selected was made, the associated action is executed and then the Widget is invalidated.

Parameters

event	Information about the click.

See also

Drawable::handleClickEvent()

Reimplemented from AbstractButton.

18.111.3.9 void setAlpha (uint8_t *alpha*) [inline]

Sets the alpha channel for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

18.111.3.10 void setBitmaps (const Bitmap & *bmpUnselected*, const Bitmap & *bmpUnselectedPressed*, const Bitmap & *bmpSelected*, const Bitmap & *bmpSelectedPressed*) [virtual]

Sets the bitmaps used by this button. If no special pressed states are needed just specify the same bitmap for both pressed and non-pressed bitmaps.

Parameters

bmpUnselected	Bitmap to use when button is unselected.
bmpUnselectedPressed	Bitmap to use when button is unselected and pressed.
bmpSelected	Bitmap to use when button is selected.
bmpSelectedPressed	Bitmap to use when button is selected and pressed.

18.111.3.11 void setDeselectedAction (GenericCallback< const AbstractButton & > & *callback*) [inline]

Associates an action to be performed when the AbstractButton is deselected.

Parameters

callback	The callback to be executed. The callback will be given a reference to the AbstractButton.

See also

GenericCallback

18.111.3.12 void setDeselectionEnabled (bool *state*) [inline]

States whether or not it is possible to de-select the RadioButton by clicking it.

Parameters

state	true if it should be possible to de-select by click.

18.111.3.13 void setSelected (bool *newSelected*)

Sets the radio buttons selected state.

Parameters

newSelected	the new selected state.

18.112 RadioButtonGroup< CAPACITY > Class Template Reference

Class for handling a collection of RadioButtons.

```
#include <touchgfx/widgets/RadioButtonGroup.hpp>
```

Public Member Functions

- RadioButtonGroup ()

  Default constructor.
- virtual ~RadioButtonGroup ()

  Destructor.
- virtual void add (RadioButton &radioButton)

  Add the RadioButton to the RadioButtonGroup.
- virtual RadioButton * getRadioButton (uint16_t index) const

  Gets the RadioButton at the specified index.
- virtual int32_t getSelectedRadioButtonIndex () const

  Gets the index of the selected RadioButton.
- virtual RadioButton * getSelectedRadioButton () const

  Gets the selected RadioButton.
- virtual void setSelected (RadioButton &radioButton)

  Sets the specified RadioButton to be selected.
- virtual void setDeselectionEnabled (bool deselectionEnabled)

  Sets whether or not it is possible to deselect RadioButtons by clicking them when they are selected.
- virtual bool getDeselectionEnabled () const

  Gets the current deselectionEnabled state.
- void setRadioButtonSelectedHandler (GenericCallback< const AbstractButton & > &callback)

  Associate an action with a radio button.
- void setRadioButtonDeselectedHandler (GenericCallback< const AbstractButton & > &callback)

  Associate an action with a radio button.

Protected Member Functions

- virtual void radioButtonClickedHandler (const AbstractButton &radioButton)

  Handles the event that a RadioButton has been selected.
- virtual void radioButtonDeselectedHandler (const AbstractButton &radioButton)

  Handles the event that a RadioButton has been deselected.

Protected Attributes

- RadioButton * radioButtons [CAPACITY]

  The list of added RadioButtons.
- uint16_t size

  The current number of added RadioButtons.
- Callback< RadioButtonGroup, const AbstractButton & > radioButtonClicked

  Callback that is attached to the RadioButtons.
- Callback< RadioButtonGroup, const AbstractButton & > radioButtonUnselected

  Callback that is attached to the RadioButtons.
- GenericCallback< const AbstractButton & > * radioButtonSelectedCallback

  The callback to be executed when a radio button belonging to this group is selected.
- GenericCallback< const AbstractButton & > * radioButtonDeselectedCallback

  The callback to be executed when a radio button belonging to this group is deselected.

18.112.1 Detailed Description

template<uint16_t CAPACITY>class touchgfx::RadioButtonGroup< CAPACITY >

Class for handling a collection of RadioButtons. The RadioButtonGroup handles the de-selection of radio buttons when a new selection occurs. A callback is executed when a new selection occurs reporting the newly selected RadioButton.

Template class: specify a CAPACITY, that is the number of RadioButtons to store.

Template Parameters

CAPACITY	Type of the capacity.

See also

RadioButton

18.112.2 Constructor & Destructor Documentation

18.112.2.1 RadioButtonGroup ( ) [inline]

Default constructor.

18.112.2.2 ~RadioButtonGroup ( ) [inline],[virtual]

Destructor.

18.112.3 Member Function Documentation

18.112.3.1 void add (RadioButton & *radioButton*) [inline], [virtual]

Add the RadioButton to the RadioButtonGroup. Only add as many RadioButtons as the stated CAPACITY. Checked by an assert.

Parameters

in	*radioButton*	the RadioButton that is to be added.

18.112.3.2 bool getDeselectionEnabled ( ) const [inline], [virtual]

Gets the current deselectionEnabled state.

Returns

The current deselectionEnabled state.

18.112.3.3 RadioButton * getRadioButton (uint16_t *index*) const [inline], [virtual]

Gets the RadioButton at the specified index.

Parameters

index	the index of the RadioButton to return.

Returns

the RadioButton at the specified index. Returns 0 if illegal index.

18.112.3.4 RadioButton * getSelectedRadioButton ( ) const [inline], [virtual]

Gets the selected RadioButton.

Returns

a pointer to the selected RadioButton. Returns 0 if no RadioButton is selected.

18.112.3.5 int32_t getSelectedRadioButtonIndex ( ) const [inline], [virtual]

Gets the index of the selected RadioButton.

Returns

the index of the selected RadioButton. Returns -1 if no RadioButton is selected.

18.112.3.6 void radioButtonClickedHandler (const AbstractButton & *radioButton*) [inline], [protected], [virtual]

Handles the event that a RadioButton has been selected. deselects all other RadioButtons.

Parameters

radioButton	the RadioButton that has been selected.

18.112.3.7 void radioButtonDeselectedHandler (const AbstractButton & *radioButton*) [inline], [protected], [virtual]

Handles the event that a RadioButton has been deselected.

Parameters

radioButton	the RadioButton that has been deselected.

18.112.3.8 void setDeselectionEnabled (bool *deselectionEnabled*) [inline], [virtual]

Sets whether or not it is possible to deselect RadioButtons by clicking them when they are selected.

Parameters

deselection↩ Enabled	true if it should be possible to deselect by click.

18.112.3.9 void setRadioButtonDeselectedHandler (GenericCallback< const AbstractButton & > & *callback*) [inline]

Associates an action to be performed when a radio button belonging to this group transition from selected to unselected.

Parameters

callback	The callback to be executed. The callback will be given a reference to the RadioButton that was selected.

See also

GenericCallback

18.112.3.10 void setRadioButtonSelectedHandler (GenericCallback< const AbstractButton & > & *callback*) [inline]

Associates an action to be performed when a radio button belonging to this group is selected.

Parameters

callback	The callback to be executed. The callback will be given a reference to the RadioButton that was selected.

See also

GenericCallback

18.112.3.11 void setSelected (RadioButton & *radioButton*) [inline], [virtual]

Sets the specified RadioButton to be selected and deselects all other. Do not call before all RadioButtons have been added to the RadioButtonGroup. Will call the radioButtonSelected callback.

Parameters

in	*radioButton*	the RadioButton to be selected.

18.113 Rasterizer Class Reference

Polygon Rasterizer that is used to render filled polygons with high-quality Anti- Aliasing.

`#include <touchgfx/canvas_widget_renderer/Rasterizer.hpp>`

Public Types

- enum { POLY_BASE_SHIFT = 5, POLY_BASE_SIZE = 1 << POLY_BASE_SHIFT, POLY_BASE_MASK = POLY_BASE_SIZE - 1 }

  Determine the sub pixel accuracy, to be more precise, the number of bits of the fractional part of the coordinates.
- enum { AA_SHIFT = 8, AA_NUM = 1 << AA_SHIFT, AA_MASK = AA_NUM - 1, AA_2NUM = AA_NUM ∗ 2, AA_2MASK = AA_2NUM - 1 }

  Determine the area accuracy, to be more precise, the number of bits of the fractional part of the areas when calculating scanlines.
- enum FillingRule { FILL_NON_ZERO, FILL_EVEN_ODD }

  Values that represent filling rules.

Public Member Functions

- Rasterizer ()

  Default constructor.
- void reset ()

  Resets this object.
- void setFillingRule (FillingRule fillingRule)

  Sets the filling rule to be used when rendering the outline.
- void moveTo (int x, int y)

  Move to.
- void lineTo (int x, int y)

  Line to.
- unsigned calculateAlpha (int area) const

  Calculates the alpha.
- template<class Renderer >
  bool render (Renderer &r)

  Renders this object.
- void setMaxRenderY (int y)

  Sets maximum render y coordinate.

18.113.1 Detailed Description

Polygon Rasterizer that is used to render filled polygons with high-quality Anti- Aliasing. Internally, by default, the class uses integer coordinates in format 24.8, i.e. 24 bits for integer part and 8 bits for fractional - see POLY_BA↩SE_SHIFT. This class can be used in the following way:

1. setFillingRule(FillingRule fr) - optional.
2. reset()

3. moveTo(x, y) / lineTo(x, y) - make the polygon. One can create more than one contour, but each contour must consist of at least 3 vertices, i.e. moveTo(x1, y1); lineTo(x2, y2); lineTo(x3, y3); is the absolute minimum of vertices that define a triangle. The algorithm does not check either the number of vertices nor coincidence of their coordinates, but in the worst case it just won't draw anything. The order of the vertices (clockwise or counterclockwise) is important when using the non-zero filling rule (fill_non_zero). In this case the vertex order of all the contours must be the same if you want your intersecting polygons to be without "holes". You actually can use different vertices order. If the contours do not intersect each other the order is not important anyway. If they do, contours with the same vertex order will be rendered without "holes" while the intersecting contours with different orders will have "holes".

setFillingRule() can be called anytime before "sweeping".

18.113.2 Member Enumeration Documentation

18.113.2.1 anonymous enum

Enumerator

POLY_BASE_SHIFT Number of bits reserved for fraction part.

POLY_BASE_SIZE The value to divide or multiply with to convert to / from this format.

POLY_BASE_MASK The value used to mask the fraction.

18.113.2.2 anonymous enum

Enumerator

AA_SHIFT Number of bits reserved for fraction part when calculating the area.

AA_NUM The value to divide or multiply with to convert to / from this format.

AA_MASK The value used to mask the fraction.

AA_2NUM Number of fraction bits when multiplying two area numbers.

AA_2MASK Mask for fraction bits when multiplying two area numbers.

18.113.2.3 enum FillingRule

Values that represent filling rules.

Enumerator

FILL_NON_ZERO Filling rule to fill anything inside the outmost border of the outline.

FILL_EVEN_ODD Filling rule to fill using xor rule inside the outline.

18.113.3 Constructor & Destructor Documentation

18.113.3.1 Rasterizer ( ) [inline]

Default constructor.

18.113.4 Member Function Documentation

18.113.4.1 unsigned calculateAlpha (int *area*) const [inline]

Calculates the alpha.

Parameters

area	The area.

Returns

The calculated alpha.

18.113.4.2 void lineTo (int *x*, int *y*) [inline]

Line to.

Parameters

x	The x coordinate.
y	The y coordinate.

18.113.4.3 void moveTo (int *x*, int *y*) [inline]

Move to.

Parameters

x	The x coordinate.
y	The y coordinate.

18.113.4.4 template< class Renderer > bool render (Renderer & *r*) [inline]

Renders this object.

Template Parameters

Renderer	Type of the renderer.

Parameters

in	*r*	The Renderer to process.

Returns

true there was enough memory available to draw the outline and render the graphics, false if there was insufficient memory and nothing was drawn.

18.113.4.5 void reset ( ) [inline]

Resets this object. Basically this is done by resetting the the Outline.

18.113.4.6 void setFillingRule (FillingRule *fillingRule*) [inline]

Sets the filling rule to be used when rendering the outline.

Parameters

fillingRule	The filling rule.

18.113.4.7 void setMaxRenderY (int *y*) [inline]

Sets maximum render y coordinate. This is passed to the Outline to avoid registering any Cell that has a y coordinate less than zero of higher than the given y.

Parameters

y	The max y coordinate to render for the Outline.

18.114 Rect Class Reference

Class representing a Rectangle with a few convenient methods.

`#include <touchgfx/hal/Types.hpp>`

Public Member Functions

- Rect ()

  Default constructor.
- Rect (int16_t x, int16_t y, int16_t width, int16_t height)

  Constructor.
- int16_t right () const

  Gets the x coordinate of the right edge of the Rect.
- int16_t bottom () const

  Gets the y coordinate of the bottom edge of the Rect.
- bool intersect (int16_t otherX, int16_t otherY) const

  Determines whether specified point lies inside this rectangle.
- bool intersect (const Rect &other) const

  Determines whether specified rectangle intersects with this rectangle.
- bool includes (const Rect &other) const

  Determines whether the specified rectangle is completely included in this rectangle.
- Rect operator& (const Rect &other) const

  Gets a rectangle describing the intersecting area between this rectangle and the supplied rectangle.
- void operator&= (const Rect &other)

  Assigns this Rect to the intersection of the current Rect and the assigned Rect.
- void expandToFit (const Rect &other)

  Increases the area covered by this rectangle to encompass the area covered by supplied rectangle.
- bool operator== (const Rect &other) const

  Compares equality of two Rect by the dimensions and position of these.
- bool operator!= (const Rect &other) const

  Opposite of the == operator.
- bool isEmpty () const

  Query if this object is empty.
- uint32_t area () const

  Calculate the area of the rectangle.

Public Attributes

- int16_t x

  The x coordinate.
- int16_t y

  The y coordinate.
- int16_t width

  The width.
- int16_t height

  The height.

18.114.1 Detailed Description

Class representing a Rectangle with a few convenient methods. Size: 8 bytes.

18.114.2 Constructor & Destructor Documentation

18.114.2.1 Rect ( ) [inline]

Default constructor. Resulting in an empty Rect with coordinates 0,0.

18.114.2.2 Rect (int16_t *x*, int16_t *y*, int16_t *width*, int16_t *height*) [inline]

Constructor.

Parameters

x	The x coordinate.
y	The y coordinate.
width	The width.
height	The height.

18.114.3 Member Function Documentation

18.114.3.1 uint32_t area ( ) const [inline]

Calculate the area of the rectangle.

Returns

area of the rectangle.

18.114.3.2 int16_t bottom ( ) const [inline]

Gets the y coordinate of the bottom edge of the Rect.

Returns

y coordinate of the buttom edge.

18.114.3.3 void expandToFit (const Rect & *other*) [inline]

Increases the area covered by this rectangle to encompass the area covered by supplied rectangle.

Parameters

other	The other rectangle.

18.114.3.4 bool includes (const Rect & *other*) const [inline]

Determines whether the specified rectangle is completely included in this rectangle.

Parameters

other	The other rectangle.

Returns

true if the specified rectangle is completely included.

18.114.3.5 bool intersect (int16_t *otherX*, int16_t *otherY*) const [inline]

Determines whether specified point lies inside this rectangle.

Parameters

otherX	The x coordinate of the point.
otherY	The y coordinate of the point.

Returns

true if point lies inside rectangle.

18.114.3.6 bool intersect (const Rect & *other*) const [inline]

Determines whether specified rectangle intersects with this rectangle.

Parameters

other	The other rectangle.

Returns

true if the two rectangles intersect.

18.114.3.7 bool isEmpty ( ) const [inline]

Query if this object is empty.

Returns

true if any of the dimensions are 0.

18.114.3.8 bool operator!= (const Rect & *other*) const [inline]

Opposite of the == operator.

Parameters

other	The Rect to compare with.

Returns

true if the compared Rect differ in dimensions or coordinates.

18.114.3.9 Rect operator& (const Rect & *other*) const [inline]

Gets a rectangle describing the intersecting area between this rectangle and the supplied rectangle.

Parameters

other	The other rectangle.

Returns

Intersecting rectangle or Rect(0, 0, 0, 0) in case of no intersection.

18.114.3.10 void operator&= (const Rect & *other*) [inline]

Assigns this Rect to the intersection of the current Rect and the assigned Rect. The assignment will result in a Rect(0, 0, 0, 0) if they do not intersect.

Parameters

other	The rect to intersect with.

18.114.3.11 bool operator== (const Rect & *other*) const [inline]

Compares equality of two Rect by the dimensions and position of these.

Parameters

other	The Rect to compare with.

Returns

true if the compared Rect have the same dimensions and coordinates.

18.114.3.12 int16_t right ( ) const [inline]

Gets the x coordinate of the right edge of the Rect.

Returns

x coordinate of the right edge.

18.115 Renderer Class Reference

This class template is used basically for rendering scan lines.

```
#include <touchgfx/canvas_widget_renderer/Renderer.hpp>
```

Public Member Functions

- Renderer ()

  Default constructor.
- Renderer (RenderingBuffer &renderingBuffer, AbstractPainter &painter)

  Constructor.
- void setRenderingBuffer (RenderingBuffer &renderingBuffer)

  Sets rendering buffer.
- void render (const Scanline &scanline)

  Render the given Scanline in the given color.
- RenderingBuffer & getRenderingBuffer ()

  Gets the getRenderingBuffer.

18.115.1 Detailed Description

This class template is used basically for rendering scanlines. The 'Span' argument is one of the span renderers, such as SpanRGB565 and others.

18.115.2 Constructor & Destructor Documentation

18.115.2.1 Renderer ( ) [inline]

Default constructor. Function setRenderingBuffer() should be called to specify where the polygon should be rendered.

18.115.2.2 Renderer (RenderingBuffer & *renderingBuffer,* AbstractPainter & *painter*) [inline]

Constructor.

Parameters

in	*renderingBuffer*	The screen buffer to render the polygon in.
in	*painter*	The painter to use for drawing individual pixels in a scanline.

18.115.3 Member Function Documentation

18.115.3.1 RenderingBuffer & getRenderingBuffer ( ) [inline]

Gets the getRenderingBuffer.

Returns

A RenderingBuffer&

18.115.3.2 void render (const Scanline & *scanline*) [inline]

Render the given Scanline in the given color.

Parameters

scanline	The Scanline.

18.115.3.3 void setRenderingBuffer (RenderingBuffer & *renderingBuffer*) [inline]

Sets rendering buffer.

Parameters

in	*renderingBuffer*	The screen buffer to render the polygon in.

18.116 RenderingBuffer Class Reference

Rendering buffer wrapper.

`#include <touchgfx/canvas_widget_renderer/RenderingBuffer.hpp>`

Public Member Functions

- ~RenderingBuffer ()

  Destructor.
- RenderingBuffer ()

  Default constructor.
- RenderingBuffer (unsigned char *buf_, unsigned char xAdjust_, unsigned width_, unsigned height_, int stride_)

  Constructor.
- void attach (unsigned char *buf_, unsigned char xAdjust_, unsigned width_, unsigned height_, int stride_)

  Attaches a buffer.
- unsigned char getXAdjust () const

  Gets x coordinate adjust.
- unsigned getWidth () const

  Gets the width.
- unsigned getHeight () const

  Gets the height.
- bool inbox (int x, int y) const

  Tests if a given coordinate is inside the RenderingBuffer.
- unsigned char * row (unsigned y)

  Gets a pointer to the given row in the RenderingBuffer.
- const unsigned char * row (unsigned y) const

  Gets a pointer to the given row in the RenderingBuffer.

18.116.1 Detailed Description

Rendering buffer wrapper. This class does not know anything about memory organizations, all it does it keeps an array of pointers to each pixel row. The general rules of rendering are as follows.

1. Allocate or create somehow a rendering buffer itself. Since the library does not depend on any particular platform or architecture it was decided that it's your responsibility to create and destroy rendering buffers properly. You can use any available mechanism to create it - you can use a system API function, simple memory allocation, or even statically defined array. You also should know the memory organization (or possible variants) in your system. For example, there's an R,G,B or B,G,R organizations with one byte per component (three

bytes per pixel) is used very often. So, if you intend to use class render_bgr24, for example, you should allocate at least width∗height∗3 bytes of memory.

2. Create a RenderingBuffer object and then call method attach(). It requires a pointer to the buffer itself, width and height of the buffer in pixels, and the length of the row in bytes. All these values must properly correspond to the memory organization. The argument stride is used because in reality the row length in bytes does not obligatory correspond with the width of the image in pixels, i.e. it cannot be simply calculated as width_in_↩ pixels ∗ bytes_per_pixel. For example, it must be aligned to 4 bytes in Windows bitmaps. Method attach() can be called more than once. The execution time of it is very little, still it allocates memory of heigh ∗ sizeof(char∗) bytes and has a loop while (height–) {...}, so it's unreasonable to call it every time before drawing any single pixel :-)

3. Create an object (or a number of objects) of a rendering class, such as renderer_bgr24_solid, renderer↩ _bgr24_image and so on. These classes require a pointer to the RenderingBuffer object, but they do not perform any considerable operations except storing this pointer. So, rendering objects can be created on demand almost any time. These objects know about concrete memory organization (this knowledge is hard coded), so actually, the memory you allocated or created in clause 1 should actually be in correspondence to the needs of the rendering class.

4. Render your image using rendering classes, for example, Rasterizer

5. Display the result, or store it, or whatever. It's also your responsibility and depends on the platform.

18.116.2 Constructor & Destructor Documentation

18.116.2.1 ∼RenderingBuffer ( )

Destructor.

18.116.2.2 RenderingBuffer ( ) [inline]

Default constructor.

18.116.2.3 RenderingBuffer (unsigned char ∗ *buf_*, unsigned char *xAdjust_*, unsigned *width_*, unsigned *height_*, int *stride_*)

Constructor.

Parameters

in	*buf_*	Pointer to the frame buffer where the image is rendered.
	*xAdjust_*	Horizontal adjustment of the x coordinate, used when bits per pixel is less than eight which implies that a uint8_t pointer cannot precisely address the start of the frame buffer.
	*width_*	The width of the frame buffer to write.
	*height_*	The height of the frame buffer to write.
	*stride_*	How much to add the a pointer inside the frame buffer to advance to the next line in the frame buffer.

18.116.3 Member Function Documentation

18.116.3.1 void attach (unsigned char ∗ *buf_*, unsigned char *xAdjust_*, unsigned *width_*, unsigned *height_*, int *stride_*)

Attaches a buffer. Can be used if the buffer is not ready when the Rendering buffer is created initially.

Parameters

in	*buf_*	Pointer to the frame buffer where the image is rendered.
	*xAdjust_*	Horizontal adjustment of the x coordinate, used when bits per pixel is less than eight which implies that a uint8_t pointer cannot precisely address the start of the frame buffer.
	*width_*	The width of the frame buffer to write.
	*height_*	The height of the frame buffer to write.
	*stride_*	How much to add the a pointer inside the frame buffer to advance to the next line in the frame buffer.

18.116.3.2 unsigned getHeight ( ) const [inline]

Gets the height.

Returns

The height.

18.116.3.3 unsigned getWidth ( ) const [inline]

Gets the width.

Returns

The width.

18.116.3.4 unsigned char getXAdjust ( ) const [inline]

Gets x coordinate adjust.

Returns

The x coordinate adjust.

18.116.3.5 bool inbox (int *x*, int *y*) const [inline]

Tests if a given coordinate is inside the RenderingBuffer.

Parameters

x	The x coordinate.
y	The y coordinate.

Returns

true if (x,y) is inside the RenderingBuffer, false otherwise.

18.116.3.6 unsigned char * row (unsigned *y*) [inline]

Gets a pointer to the given row in the RenderingBuffer.

Parameters

y	The line number, ie the row.

Returns

The pointer to the start of the given line in the RenderingBuffer.

18.116.3.7 const unsigned char * row (unsigned *y*) const [inline]

Gets a pointer to the given row in the RenderingBuffer.

Parameters

y	The line number, ie the row.

Returns

The pointer to the start of the given line in the RenderingBuffer.

18.117 RGBA8 Struct Reference

Deprecated, use colortype instead.

```
#include <touchgfx/canvas_widget_renderer/RGBA8.hpp>
```

Public Member Functions

- RGBA8 ()

  Default constructor. Deprecated, use colortype instead.
- RGBA8 (uint16_t rgb565, unsigned alpha_=255)

  Constructor. Deprecated, use colortype instead.
- RGBA8 (unsigned red_, unsigned green_, unsigned blue_, unsigned alpha_=255)

  Constructor. Deprecated, use colortype instead.

Public Attributes

- uint8_t red

  The red part of the color.
- uint8_t green

  The green part of the color.
- uint8_t blue

  The blue part of the color.
- uint8_t alpha

  The alpha value of the color, specifying how transparent the color is (0 = invisible, 255 = solid).

18.117.1 Detailed Description

Note

Deprecated, use colortype instead.

See also

colortype

18.117.2 Constructor & Destructor Documentation

18.117.2.1 RGBA8 ( ) [inline]

Deprecated, use colortype instead.

Note

Deprecated, use colortype instead.

See also

colortype

18.117.2.2 RGBA8 (uint16_t *rgb565*, unsigned *alpha_* = 255) [inline]

Deprecated, use colortype instead.

Note

Deprecated, use colortype instead.

Parameters

rgb565	RGB565 encoded color.
*alpha_*	alpha value, specifying the transparency of the color (0 = invisible, 255 = solid).

See also

colortype

18.117.2.3 RGBA8 (unsigned *red_*, unsigned *green_*, unsigned *blue_*, unsigned *alpha_* = 255) [inline]

Deprecated, use colortype instead.

Note

Deprecated, use colortype instead.

Parameters

*red_*	The red part of the color.
*green_*	The green part of the color.
*blue_*	The blue part of the color.
*alpha_*	alpha value, specifying the transparency of the color (0 = invisible, 255 = solid).

See also

colortype

18.118 ScalableImage Class Reference

Widget for representing a scaled version of a bitmap.

```
#include <touchgfx/widgets/ScalableImage.hpp>
```

Public Types

- enum ScalingAlgorithm { **NEAREST_NEIGHBOR**, **BILINEAR_INTERPOLATION** }

  Rendering algorithms of the scaled bitmap.

Public Member Functions

- ScalableImage ()

  Default constructor.
- virtual ~ScalableImage ()

  Destructor.
- virtual void setBitmap (const Bitmap &bmp)

  Sets the bitmap for the image.
- Bitmap getBitmap () const

  Gets the bitmap for the image.
- virtual void setAlpha (uint8_t alpha)

  Sets the alpha channel for the image.
- virtual uint8_t getAlpha () const

  Gets the current alpha value.
- virtual void setScalingAlgorithm (ScalingAlgorithm algorithm)

  Sets the algorithm to be used.
- virtual ScalingAlgorithm getScalingAlgorithm ()

  Gets the algorithm used when rendering.
- virtual void draw (const Rect &invalidatedArea) const

  Draws the given invalidated area.
- virtual Rect getSolidRect () const

  Gets solid rectangle.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Member Functions

- void drawTriangle (const Rect &invalidatedArea, uint16_t *fb, const float *triangleXs, const float *triangleYs, const float *triangleZs, const float *triangleUs, const float *triangleVs) const

  Draw a triangle part of the bitmap.
- RenderingVariant lookupRenderVariant () const

  Looks up the appropriate render variant based on the bitmap format and scaling algorithm.

Protected Attributes

- ScalingAlgorithm currentScalingAlgorithm

  The current scaling algorithm.
- Bitmap bitmap

  The bitmap to render.
- uint8_t alpha

  An alpha value that is applied to the entire image.

Additional Inherited Members

18.118.1 Detailed Description

Widget for representing a scaled version of a bitmap. Simply change the width/height of the widget to resize the image. The quality of the scaled image depends of the rendering algorithm used. The rendering algorithm can be changed dynamically. Please note that scaling images is done at runtime and requires a lot of calculations. Therefore use it with some care.

Note that this widget does not support 1 bit per pixel color depth.

See also

Widget

18.118.2 Member Enumeration Documentation

18.118.2.1 enum ScalingAlgorithm

Rendering algorithms of the scaled bitmap.

NEAREST_NEIGHBOR: Fast but not a very good image quality. Good for fast animations.

BILINEAR_INTERPOLATION: Slow but good image quality. Good for static representation of a scaled image.

18.118.3 Constructor & Destructor Documentation

18.118.3.1 ScalableImage ( )

Default constructor.

18.118.3.2 ~ScalableImage ( ) [virtual]

Destructor.

18.118.4 Member Function Documentation

18.118.4.1 void draw (const Rect & *invalidatedArea*) const [virtual]

Draws the given invalidated area.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Implements Drawable.

18.118.4.2 void drawTriangle (const Rect & *invalidatedArea*, uint16_t * *fb*, const float * *triangleXs*, const float * *triangleYs*, const float * *triangleZs*, const float * *triangleUs*, const float * *triangleVs*) const [protected]

Draw a triangle part of the bitmap.

Parameters

	invalidatedArea	The invalidated area.
in,out	*fb*	If non-null, the fb.
	triangleXs	The triangle xs.
	triangleYs	The triangle ys.
	triangleZs	The triangle zs.
	triangleUs	The triangle us.
	triangleVs	The triangle vs.

18.118.4.3 uint8_t getAlpha ( ) const [inline],[virtual]

Gets the current alpha value.

Returns

The current alpha value.

18.118.4.4 Bitmap getBitmap ( ) const [inline]

Gets the bitmap for the image.

Returns

the small bitmap.

18.118.4.5 ScalingAlgorithm getScalingAlgorithm ( ) [virtual]

Gets the algorithm used when rendering.

Returns

The algorithm used when rendering.

18.118.4.6 Rect getSolidRect ( ) const [virtual]

Gets solid rectangle.

Returns

largest possible solid rect.

See also

Drawable::getSolidRect()

Implements Drawable.

18.118.4.7 uint16_t getType ( ) const [inline],[virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_SCALABLEIMAGE.

Reimplemented from Widget.

18.118.4.8 RenderingVariant lookupRenderVariant ( ) const [protected]

Looks up the appropriate render variant based on the bitmap format and scaling algorithm.

Returns

A RenderingVariant.

18.118.4.9 void setAlpha (uint8_t *alpha*) [inline], [virtual]

Sets the alpha channel for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

18.118.4.10 void setBitmap (const Bitmap & *bmp*) [virtual]

Sets the bitmap for the image.

Parameters

bmp	The bitmap to be used by the widget.

18.118.4.11 void setScalingAlgorithm (ScalingAlgorithm *algorithm*) [virtual]

Sets the algorithm to be used.

Parameters

algorithm	The algorithm to use when rendering.

18.119 Scanline Class Reference

This class is used to transfer data from class Outline (or a similar one) to the rendering buffer.

`#include <touchgfx/canvas_widget_renderer/Scanline.hpp>`

Classes

- class iterator

  An iterator to help go through all the elements that make up a Scanline.

Public Types

- enum { **AA_SHIFT** = 8 }

Public Member Functions

- ~Scanline ()

  Destructor.
- Scanline ()

Default constructor.

- void reset ()

  Resets the Scanline object in preparation for the handling the next Scanline.
- void resetSpans ()

  Resets the spans in preparation for the next Scanline.
- void addCell (int x, int y, unsigned cover)

  Adds a single cell to the current Scanline.
- void addSpan (int x, int y, unsigned len, unsigned cover)

  Adds a span of cells to the current Scanline.
- int isReady (int y) const

  Checks if a Scanline is ready for rendering.
- int getY () const

  Gets y coordinate, i.e. the vertical offset of the Scanline.
- unsigned getNumSpans () const

  Gets number spans in the Scanline.

Friends

- class **iterator**

18.119.1 Detailed Description

This class is used to transfer data from class Outline (or a similar one) to the rendering buffer. It's organized very simple. The class stores information of horizontal spans to render it into a pixel-map buffer. Each span has initial X, length, and an array of bytes that determine the alpha values for each pixel. So, the restriction of using this class is 256 levels of Anti-Aliasing, which is quite enough for any practical purpose. Before using this class you should know the minimal and maximal pixel coordinates of your scanline. The protocol of using is: 1. reset()

1. addCell() / addSpan() - accumulate scanline. You pass y coordinate into these functions in order to make scanline know the last Y. Before calling addCell() / addSpan() you should check with method isReady(y) if the last Y has changed. It also checks if the scanline is not empty. When forming one scanline the next x coordinate must be always greater than the last stored one, i.e. it works only with ordered coordinates.

2. If the current scanline isReady() you should render it and then call resetSpans() before adding new cells/spans.

3. Rendering:

Scanline provides an iterator class that allows you to extract the spans and the cover values for each pixel. Be aware that clipping has not been done yet, so you

should perform it yourself. Use Scanline::iterator to render spans:

int baseX = scanline.getBaseX(); // base X. Should be added to the span's X // "scanline" is a const reference to the // scanline passed in.

int y = scanline.y(); // y coordinate of the scanline

...Perform vertical clipping here...

Scanline::iterator span(scanline);

unsigned char∗ row = renderingBuffer->row(y); // The the address of the beginning // of the current row

unsigned num_spans = scanline.getNumSpans(); // Number of spans. It's guaranteed that // numSpans is always greater than 0.

do { int x = span.next() + baseX; // The beginning X of the span

const int8u covers∗ = span.getCovers(); // The array of the cover values

int numPix = span.getNumPix(); // Number of pixels of the span. // Always greater than 0, still we // should use "int" instead of // "unsigned" because it's more // convenient for clipping

...Perform horizontal clipping here... ...you have x, covers, and pix_Fromcount...

unsigned char∗ dst = row + x; // Calculate the start address of the row. // In this case we assume a simple // grayscale image 1-byte per pixel. do { ∗dst++ = ∗covers++; // Hypotetical rendering. } while (–numPix); } while (–numSpans); // numSpans cannot be 0, so this loop is quite safe

The question is: why should we accumulate the whole scanline when we could render just separate spans when they're ready? That's because using the scanline is in general faster. When is consists of more than one span the conditions for the processor cash system are better, because switching between two different areas of memory (that can be large ones) occurs less frequently.

18.119.2 Constructor & Destructor Documentation

18.119.2.1 ~Scanline ( ) [inline]

Destructor.

18.119.2.2 Scanline ( )

Default constructor. Initiate a Scanline by setting up pointers to store covers, and counts.

18.119.3 Member Function Documentation

18.119.3.1 void addCell (int *x*, int *y*, unsigned *cover*) [inline]

Adds a single cell to the current Scanline. Works just like invoking addSpan() with a len=1.

Parameters

x	The x coordinate.
y	The y coordinate.
cover	The cover.

18.119.3.2 void addSpan (int *x*, int *y*, unsigned *len*, unsigned *cover*)

Adds a span of cells to the current Scanline. Works like calling addCell() len times.

Parameters

x	The x coordinate.
y	The y coordinate.
len	The length.
cover	The cover.

18.119.3.3 unsigned getNumSpans ( ) const [inline]

Gets number spans in the Scanline.

Returns

The number spans.

18.119.3.4 int getY ( ) const [inline]

Gets y coordinate, i.e. the vertical offset of the Scanline. This allows easy positioning of the Outline. The y coordinate is setup through function reset().

Returns

The y coordinate.

18.119.3.5 int isReady (int *y*) const [inline]

Checks if a Scanline is ready for rendering. A Scanline is ready for rendering when the y coordinate has changed. Since all the cells are sorted, a change in the y coordinate means that we have moved to the next Scanline and thus the collected data for the Scanline must be rendered before we register cells for the next Scanline.

Parameters

y	The y coordinate.

Returns

True if the given y coordinate differs from the y coordinate for the cells in the current Scanline.

18.119.3.6 void reset ( )

Resets the Scanline object in preparation for the handling the next Scanline.

18.119.3.7 void resetSpans ( ) [inline]

Resets the spans in preparation for the next Scanline. Identical to calling reset() without changing the dx_ and dy_ parameters from the previous call to reset().

18.120 Screen Class Reference

A Screen represents a full-screen drawable area. Applications create specific screens by subclassing this class.

`#include <touchgfx/Screen.hpp>`

Public Member Functions

- Screen ()

  Default constructor.
- virtual ~Screen ()

  Destructor.
- void draw ()

  Tells the screen to draw its entire area.
- void startSMOC (Rect &invalidatedArea)

  Starts a JSMOC run, analyzing what parts of what widgets should be redrawn.

- void JSMOC (const Rect &invalidatedArea, Drawable *widgetToDraw)

  Recursive JSMOC function. This is the actual occlusion culling implementation.
- virtual void draw (Rect &rect)

  Tell the screen to draw the specified area.
- virtual void setupScreen ()

  Called by Application::switchScreen() when this screen is going to be displayed.
- virtual void afterTransition ()

  Called by Application::handleTick() when the transition to the screen is done.
- virtual void tearDownScreen ()

  Called by Application::switchScreen() when this screen will no longer be displayed.
- virtual void handleClickEvent (const ClickEvent &evt)

  Traverse the drawables in reverse z-order and notify them of a click event.
- virtual void handleDragEvent (const DragEvent &evt)

  Traverse the drawables in reverse z-order and notify them of a drag event.
- virtual void handleGestureEvent (const GestureEvent &evt)

  Handle gestures. Traverses drawables in reverse-z and notifies them of the gesture.
- virtual void handleTickEvent ()

  Called by the Application on the current screen with a frequency of Application::TICK_INTERVAL_MS.
- virtual void handleKeyEvent (uint8_t key)

  Called by the Application on the reception of a "key", the meaning of which is platform/application specific.
- bool usingSMOC () const

  Determines if using JSMOC.
- void bindTransition (Transition &trans)

  Enables the transition to access the containers.
- Container & getRootContainer ()

  Obtain a reference to the root container of this screen.

Protected Member Functions

- void useSMOCDrawing (bool enabled)

  Determines whether to use JSMOC or painter's algorithm for drawing.
- void add (Drawable &d)

  Add a drawable to the content container.
- void remove (Drawable &d)

  Removes a drawable from the content container.

Protected Attributes

- Container container

  The container contains the contents of the screen.
- Drawable * focus

  The drawable currently in focus (set when DOWN_PRESSED is received).

18.120.1 Detailed Description

A Screen represents a full-screen drawable area. Applications create specific screens by subclassing this class.

Each screen has a root container to which drawables can be added.

This class makes sure to delegate draw requests and various events to the appropriate drawables in correct order.

18.120.2 Constructor & Destructor Documentation

18.120.2.1 Screen ( )

Default constructor.

18.120.2.2 ~Screen ( ) [inline],[virtual]

Destructor.

18.120.3 Member Function Documentation

18.120.3.1 void add (Drawable & *d*) [inline],[protected]

Add a drawable to the content container.

Note

Must not be called with a Drawable that was already added to the screen. If in doubt, call remove() first.

Parameters

in	*d*	The Drawable to add.

18.120.3.2 void afterTransition ( ) [inline],[virtual]

Called by Application::handleTick() when the transition to the screen is done. Base version does nothing, but override to do screen specific initialization code that has to be done after the transition to the screen.

See also

touchgfx::Application::handleTick()

18.120.3.3 void bindTransition (Transition & *trans*)

Enables the transition to access the containers.

Parameters

in	*trans*	The transition to bind.

18.120.3.4 void draw ( )

Tells the screen to draw its entire area.

Note

The more specific draw(Rect&) version is preferred when possible.

18.120.3.5 void draw (Rect & *rect*) [virtual]

Tell the screen to draw the specified area. Will traverse the drawables tree in z- order and delegate draw to them.

Note

The given rect must be in absolute coordinates.

Parameters

in	*rect*	The area in absolute coordinates.

18.120.3.6 Container & getRootContainer ( ) [inline]

Obtain a reference to the root container of this screen.

Returns

The root container.

18.120.3.7 void handleClickEvent (const ClickEvent & *evt*) [virtual]

Traverse the drawables in reverse z-order and notify them of a click event.

Parameters

evt	The event to handle.

18.120.3.8 void handleDragEvent (const DragEvent & *evt*) [virtual]

Traverse the drawables in reverse z-order and notify them of a drag event.

Parameters

evt	The event to handle.

18.120.3.9 void handleGestureEvent (const GestureEvent & *evt*) [virtual]

Handle gestures. Traverses drawables in reverse-z and notifies them of the gesture.

Parameters

evt	The event to handle.

18.120.3.10 void handleKeyEvent (uint8_t *key*) [inline], [virtual]

Called by the Application on the reception of a "key", the meaning of which is platform/application specific. Default implementation does nothing.

Parameters

key	The key to handle.

18.120.3.11 void handleTickEvent ( ) [inline], [virtual]

Called by the Application on the current screen with a frequency of Application::TICK_INTERVAL_MS.

18.120.3.12 void JSMOC (const Rect & *invalidatedArea,* Drawable * *widgetToDraw*)

Recursive JSMOC function. This is the actual occlusion culling implementation.

Parameters

in	*invalidatedArea*	The area to redraw, expressed in absolute coordinates.
in	*widgetToDraw*	Widget currently being drawn.

18.120.3.13 void remove (Drawable & *d*) [inline], [protected]

Removes a drawable from the content container. Safe to call even if the drawable was never added (in which case nothing happens).

Parameters

in	*d*	The Drawable to remove.

18.120.3.14 void setupScreen ( ) [inline], [virtual]

Called by Application::switchScreen() when this screen is going to be displayed. Base version does nothing, but place any screen specific initialization code in an overridden version.

See also

touchgfx::Application::switchScreen()

18.120.3.15 void startSMOC (Rect & *invalidatedArea*)

Starts a JSMOC run, analyzing what parts of what widgets should be redrawn.

Parameters

in	*invalidatedArea*	The area to redraw, expressed in absolute coordinates.

18.120.3.16 void tearDownScreen ( ) [inline], [virtual]

Called by Application::switchScreen() when this screen will no longer be displayed. Base version does nothing, but place any screen specific cleanup code in an overridden version.

See also

touchgfx::Application::switchScreen()

18.120.3.17 void useSMOCDrawing (bool *enabled*) [protected]

Determines whether to use JSMOC or painter's algorithm for drawing.

Parameters

enabled	true if JSMOC should be enabled, false if disabled (meaning painter's algorithm is employed instead).

18.120.3.18 bool usingSMOC ( ) const [inline]

Returns

true if this screen uses the JSMOC drawing algorithm.

18.121 ScrollableContainer Class Reference

A ScrollableContainer is a container that allows its contents to be scrolled.

```
#include <touchgfx/containers/ScrollableContainer.hpp>
```

Public Member Functions

- ScrollableContainer ()

  Default constructor.
- virtual ~ScrollableContainer ()

  Destructor.
- void enableHorizontalScroll (bool enable)

  Enables horizontal scrolling.
- void enableVerticalScroll (bool enable)

  Enables the vertical scroll.
- void setScrollbarsVisible (bool newVisible)

  Sets scrollbars visible.
- virtual void add (Drawable &d)

  Adds a Drawable instance as child to this ScrollableContainer.
- virtual void getLastChild (int16_t x, int16_t y, Drawable **last)

  Gets the last child in the container.
- virtual void handleGestureEvent (const GestureEvent &evt)

  Gestures generate a scroll animation so these are intercepted in the same manner as drag events.
- virtual void handleDragEvent (const DragEvent &evt)

  Handle the drag event.
- virtual void handleClickEvent (const ClickEvent &evt)

  Handle the click event.
- virtual void handleTickEvent ()

  Handle tick events.
- virtual Rect getContainedArea () const

  Gets contained area.
- virtual void childGeometryChanged ()

  Used to signal that the size of one or more children have changed.
- void reset ()

  Resets the x/y coordinates of children.
- virtual void moveChildrenRelative (int16_t deltaX, int16_t deltaY)

  Moves the scrollable contents relatively.
- void setMaxVelocity (uint16_t max)

  Sets the maximum velocity of a scroll due to a swipe.
- void setScrollThreshold (int16_t t)

  Change the threshold which the first drag event received must exceed before initiating a scroll.
- void setScrollbarsColor (colortype color)

  Sets the color of the scroll bars.
- void setScrollbarsAlpha (uint8_t alpha)

  Sets the alpha value for the scroll bars.
- void setScrollbarPadding (uint8_t padding)

  Sets the amount of space the scrollbar has to its borders.
- void setScrollbarWidth (uint8_t width)

  Sets the width of the scrollbar.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Member Functions

- Rect getXScrollbar () const

  Gets x coordinate of the scrollbar.
- Rect getYScrollbar () const

  Gets y coordinate of the scrollbar.
- Rect getXBorder (const Rect &xBar, const Rect &yBar) const

  Gets the area where the horizontal scrollbar can move.
- Rect getYBorder (const Rect &xBar, const Rect &yBar) const

  Gets the area where the vertical scrollbar can move.
- void invalidateScrollbars ()

  Invalidate the scrollbars.
- virtual bool doScroll (int16_t deltaX, int16_t deltaY)

  Method to actually scroll the container.

Protected Attributes

- uint8_t scrollbarPadding

  The amount of padding. The scrollbar will have a bit of space to the borders of the container.
- uint8_t scrollbarWidth

  The width of the scrollbar.
- uint8_t scrollbarAlpha

  The scrollbar is semitransparent.
- colortype scrollbarColor

  The color of the scrollbar.
- uint16_t maxVelocity

  The maximum velocity of a scroll (due to a swipe)
- GestureEvent::GestureType accelDirection

  The current direction (horizontal or vertical) of scroll.
- Box xSlider

  The horizontal scrollbar drawable.
- Box ySlider

  The vertical scrollbar drawable.
- Drawable * pressedDrawable

  The drawable child of this container which received the last ClickEvent::PRESSED notification. When scrolling, send this drawable a CANCEL event if the new x/y coords no longer matches this drawable.
- Drawable * lastDraggableChild

  The drawable child of this container which should receive drag events. Note that only drag events in directions which cannot be scrolled by this ScrollableContainer will be forwarded to children.
- int16_t scrolledXDistance

  The scrolled horizontal distance.
- int16_t scrolledYDistance

  The scrolled vertical distance.
- int16_t scrollThreshold

  The threshold which the first drag event received must exceed before scrolling. Default is 5.
- int16_t pressedX

  The x coordinate where the last ClickEvent::PRESSED was received.
- int16_t pressedY

  The y coordinate where the last ClickEvent::PRESSED was received.
- bool isPressed

  Is the container currently pressed (maybe show scrollbars)

- bool isScrolling

  Is the container scrolling (i.e. has overcome the initial larger drag that is required to initiate a scroll).
- bool scrollableX

  Is the container scrollable in the horizontal direction.
- bool scrollableY

  Is the container scrollable in the vertical direction.
- bool scrollbarsVisible

  Are scrollbars visible.
- uint16_t scrollDuration

  Number of ticks the scroll animation should use.
- int16_t beginningValue

  Initial X or Y for calculated values in scroll animation.
- int16_t targetValue

  Target X or Y value for scroll animation.
- uint16_t animationCounter

  Current step/tick in scroll animation.
- bool animate

  Is scroll animation currently active.
- int16_t fingerAdjustmentX

  How much should the finger be adjusted horizontally.
- int16_t fingerAdjustmentY

  and how much vertically
- bool hasIssuedCancelEvent

  true if the pressed drawable has received cancel event

Static Protected Attributes

- static const uint8_t SCROLLBAR_LINE = 1

  The scrollbar line.
- static const uint16_t SCROLLBAR_MIN_VELOCITY = 5

  The minimum velocity of a scroll due to a swipe.
- static const uint16_t SCROLLBAR_MAX_VELOCITY = 17

  The (default) maximum velocity of a scroll due to a swipe.

Additional Inherited Members

18.121.1 Detailed Description

A ScrollableContainer is a container that allows its contents to be scrolled. It will intercept drag operations and move child nodes accordingly.

The size of the ScrollableContainer should be the visible view port area. If the container contains drawables that are larger than the ScrollableContainer itself, scrolling is enabled.

Note

The ScrollableContainer will consume all DragEvents in the area covered by the container, and use.

See also

Container

18.121.2 Constructor & Destructor Documentation

18.121.2.1 ScrollableContainer ( )

Default constructor.

18.121.2.2 ~ScrollableContainer ( ) [inline],[virtual]

Destructor.

18.121.3 Member Function Documentation

18.121.3.1 void add (Drawable & *d*) [virtual]

Adds a Drawable instance as child to this ScrollableContainer.

Parameters

in	*d*	The drawable.

Reimplemented from Container.

18.121.3.2 void childGeometryChanged ( ) [virtual]

This function can be called on parent nodes to signal that the size of one or more of its children have changed. Currently only used in ScrollableContainer to redraw scrollbars when the size of the scrolling contents changes.

See also

Drawable::childGeometryChanged

Reimplemented from Drawable.

18.121.3.3 bool doScroll (int16_t *deltaX*, int16_t *deltaY*) [protected],[virtual]

Method to actually scroll the container.

Parameters

deltaX	The horizontal amount to scroll.
deltaY	The vertical amount to scroll.

Returns

did the container actually scroll.

18.121.3.4 void enableHorizontalScroll (bool *enable*) [inline]

By default, scrolling in either direction is enabled, provided that the content is larger than the size of the scrollable container. This function can be used to explicitly (dis)allow scrolling in the horizontal direction, even if the content is larger than the container.

Parameters

enable	If true (default), horizontal scrolling is enabled. If false, scrolling is disabled.

18.121.3.5 void enableVerticalScroll (bool *enable*) [inline]

Enables the vertical scroll. By default, scrolling in either direction is enabled, provided that the content is larger than the size of the scrollable container. This function can be used to explicitly (dis)allow scrolling in the vertical direction, even if the content is larger than the container.

Parameters

enable	If true (default), vertical scrolling is enabled. If false, scrolling is disabled.

18.121.3.6 Rect getContainedArea ( ) const [virtual]

Gets contained area.

Returns

The contained area.

Reimplemented from Container.

18.121.3.7 void getLastChild (int16_t *x*, int16_t *y*, Drawable ** *last*) [inline], [virtual]

Gets the last child in the container. The ScrollableContainer needs to intercept click events, since the scrollbars are displayed upon reception of a PRESSED ClickEvent. The ScrollableContainer will automatically re-delegate the event to the appropriate child.

Parameters

	x	The x coordinate of the (click) event.
	y	The y coordinate of the (click) event.
out	*last*	The last child intersecting x,y. ScrollableContainer intercepts these, so returns it self.

Reimplemented from Container.

18.121.3.8 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_SCROLLABLECONTAINER.

Reimplemented from Container.

18.121.3.9 Rect getXBorder (const Rect & *xBar*, const Rect & *yBar*) const [protected]

Gets the area where the horizontal scrollbar can move.

Parameters

xBar	The current horizontal scrollbar, supplied for caching reasons.
yBar	The current vertical scrollbar, supplied for caching reasons.

Returns

The area.

18.121.3.10 Rect getXScrollbar ( ) const [protected]

Gets x coordinate of the scrollbar.

Returns

The horizontal scrollbar area.

18.121.3.11 Rect getYBorder (const Rect & *xBar,* const Rect & *yBar*) const [protected]

Gets the area where the vertical scrollbar can move.

Parameters

xBar	The current horizontal scrollbar, supplied for caching reasons.
yBar	The current vertical scrollbar, supplied for caching reasons.

Returns

The area.

18.121.3.12 Rect getYScrollbar ( ) const [protected]

Gets y coordinate of the scrollbar.

Returns

The vertical scrollbar area.

18.121.3.13 void handleClickEvent (const ClickEvent & *evt*) [virtual]

Handle the click event. Get ready for scrolling, display scrollbars, etc. Send the click to appropriate child widget.

Parameters

evt	The ClickEvent.

Reimplemented from Drawable.

18.121.3.14 void handleDragEvent (const DragEvent & *evt*) [virtual]

Handle the drag event. Initiate a scrolling of the container. Update scrollbars.

Parameters

evt	The DragEvent.

Reimplemented from Drawable.

18.121.3.15 void handleGestureEvent (const GestureEvent & *evt*) [virtual]

Gestures generate a scroll animation so these are intercepted in the same manner as drag events.

Parameters

evt	The GestureEvent.

Reimplemented from Drawable.

18.121.3.16 void handleTickEvent ( ) [virtual]

Handle tick events. Used in updating the animation of the scroll.

Reimplemented from Drawable.

18.121.3.17 void invalidateScrollbars ( ) [protected]

Invalidate the scrollbars.

18.121.3.18 void moveChildrenRelative (int16_t *deltaX*, int16_t *deltaY*) [virtual]

Moves the scrollable contents relatively.

Parameters

deltaX	Horizontal displacement.
deltaY	Vertical displacement.

Reimplemented from Container.

18.121.3.19 void reset ( )

Resets the x/y coordinates of childrento the position they were in before the first drag event was received or to the position they were in the last time reset() was invoked.

18.121.3.20 void setMaxVelocity (uint16_t *max*) [inline]

Sets the maximum velocity of a scroll due to a swipe.

Parameters

max	The maximum velocity of the scroll.

18.121.3.21 void setScrollbarPadding (uint8_t *padding*)

Sets the amount of space the scrollbar has to its borders.

Parameters

padding	The padding.

18.121.3.22 void setScrollbarsAlpha (uint8_t *alpha*)

Sets the alpha value for the scroll bars.

Parameters

alpha	The alpha value. 255 = completely solid.

18.121.3.23 void setScrollbarsColor (colortype *color*)

Sets the color of the scroll bars.

Parameters

color	The color of the box.

18.121.3.24 void setScrollbarsVisible (bool *newVisible*) [inline]

Sets scrollbars visible.

Parameters

newVisible	If true (default), the scrollbars are visible. If false, scrollbars are hidden.

18.121.3.25 void setScrollbarWidth (uint8_t *width*)

Sets the width of the scrollbar.

Parameters

width	The width of the scrollbar.

18.121.3.26 void setScrollThreshold (int16_t *t*) [inline]

Change the threshold which the first drag event received must exceed before initiating a scroll.

Note

All subsequent scrolls will be processed regardless of threshold value until a ClickEvent::RELEASED is received.

Parameters

t	The new threshold value.

18.122 SDLTouchController Class Reference

TouchController for the simulator.

```
#include <platform/driver/touch/SDLTouchController.hpp>
```

Public Member Functions

- virtual void init ()

  Initializes touch controller.
- virtual bool sampleTouch (int32_t &x, int32_t &y)

  Checks whether the touch screen is being touched, and if so, what coordinates.

18.122.1 Detailed Description

TouchController for the simulator.

See also

TouchController

18.122.2 Member Function Documentation

18.122.2.1 void init ( ) [virtual]

Initializes touch controller.

Implements TouchController.

18.122.2.2 bool sampleTouch (int32_t & *x*, int32_t & *y*) [virtual]

Checks whether the touch screen is being touched, and if so, what coordinates.

Parameters

out	*x*	The x position of the touch
out	*y*	The y position of the touch

Returns

True if a touch has been detected, otherwise false.

Implements TouchController.

18.123 Shape< POINTS > Class Template Reference

Simple widget capable of drawing a shape.

`#include <touchgfx/widgets/canvas/Shape.hpp>`

Public Member Functions

- virtual ~Shape ()

  Virtual Destructor.
- virtual int getNumPoints () const

  Gets number points used to make up the shape.
- virtual void setCorner (int i, CWRUtil::Q5 x, CWRUtil::Q5 y)

  Sets a corner of the shape in Q5 format.
- virtual CWRUtil::Q5 getCornerX (int i) const

  Gets the x coordinate of a corner.

- virtual CWRUtil::Q5 getCornerY (int i) const

  Gets the y coordinate of a corner.
- virtual void setCache (int i, CWRUtil::Q5 x, CWRUtil::Q5 y)

  Sets the cached coordinates of a given corner.
- virtual CWRUtil::Q5 getCacheX (int i) const

  Gets cached x coordinate of a corner.
- virtual CWRUtil::Q5 getCacheY (int i) const

  Gets cached y coordinate of a corner.

Additional Inherited Members

18.123.1 Detailed Description

template<uint16_t POINTS>class touchgfx::Shape< POINTS >

Simple widget capable of drawing a shape. The shape can be scaled and rotated around 0,0. Note that the y axis goes down, so a shape that goes up must be given negative coordinates.

The Shape class allows the user to draw any shape and allows the defined shape to be scaled, rotated and moved freely. Example uses could be the hands of a clock (see the touchgfx_demo2014 for an actual implementation).

Template Parameters

POINTS	The number of points in the given shape.
T	The type of the points used for the shape. Must be int or float.

See also

CanvasWidget

18.123.2 Constructor & Destructor Documentation

18.123.2.1 ~Shape ( ) [inline],[virtual]

Virtual Destructor.

18.123.3 Member Function Documentation

18.123.3.1 CWRUtil::Q5 getCacheX (int *i*) const [inline],[virtual]

Gets cached x coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The cached x coordinate.

Implements AbstractShape.

18.123.3.2 CWRUtil::Q5 getCacheY (int *i*) const [inline],[virtual]

Gets cached y coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The cached y coordinate.

Implements AbstractShape.

18.123.3.3 CWRUtil::Q5 getCornerX (int *i*) const [inline], [virtual]

Gets the x coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The corner x coordinate.

Implements AbstractShape.

18.123.3.4 CWRUtil::Q5 getCornerY (int *i*) const [inline], [virtual]

Gets the y coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The corner y coordinate.

Implements AbstractShape.

18.123.3.5 int getNumPoints ( ) const [inline], [virtual]

Gets number points used to make up the shape.

Returns

The number points.

Implements AbstractShape.

18.123.3.6 void setCache (int *i*, CWRUtil::Q5 *x*, CWRUtil::Q5 *y*) [inline], [virtual]

Sets the cached coordinates of a given corner. The coordinates in the cache are the coordinates from the corners after rotating and scaling the coordinate.

Parameters

i	Zero-based index of the corner.
x	The x coordinate.
y	The y coordinate.

Implements AbstractShape.

18.123.3.7 void setCorner (int *i*, CWRUtil::Q5 *x*, CWRUtil::Q5 *y*) [inline],[virtual]

Sets a corner of the shape in Q5 format.

Parameters

i	Zero-based index of the corner.
x	The x coordinate in Q5 format.
y	The y coordinate in Q5 format.

Implements AbstractShape.

18.124 AbstractShape::ShapePoint< T > Struct Template Reference

Defines an alias representing the array of points making up the abstract shape.

```
#include <touchgfx/widgets/canvas/AbstractShape.hpp>
```

Public Attributes

- T x

  The x coordinate of the points.
- T y

  The y coordinate of the points.

18.124.1 Detailed Description

template<class T>struct touchgfx::AbstractShape::ShapePoint< T >

Defines an alias representing the array of points making up the abstract shape. This will help setting up the abstractShape very easily using setAbstractShape().

Template Parameters

T	Generic type parameter.

See also

setAbstractShape()

18.125 Slider Class Reference

A slider is a graphical element with which the user may set a value by moving an indicator or by clicking the slider.

```
#include <touchgfx/containers/Slider.hpp>
```

Public Member Functions

- Slider ()

  Default constructor.
- virtual ~Slider ()

  Destructor.
- void setBitmaps (const Bitmap &sliderBackground, const Bitmap &sliderBackgroundSelected, const Bitmap &indicator)

  Sets all the bitmaps for the Slider.
- void setBitmaps (const BitmapId sliderBackground, const BitmapId sliderBackgroundSelected, const BitmapId indicator)

  Sets all the bitmaps for the Slider.
- void setStartValueCallback (GenericCallback< const Slider &, int > &callback)

  Associates an action to be performed when an interaction (drag or click) with the slider is initiated.
- void setStopValueCallback (GenericCallback< const Slider &, int > &callback)

  Associates an action to be performed when an interaction with the slider ends (i.e. drag/click).
- void setNewValueCallback (GenericCallback< const Slider &, int > &callback)

  Associates an action to be performed when the slider changes its value.
- void setupHorizontalSlider (uint16_t backgroundX, uint16_t backgroundY, uint16_t indicatorY, uint16_↩ t indicatorMinX, uint16_t indicatorMaxX)

  Sets up the slider in horizontal mode.
- void setupVerticalSlider (uint16_t backgroundX, uint16_t backgroundY, uint16_t indicatorX, uint16_↩ t indicatorMinY, uint16_t indicatorMaxY)

  Sets up the slider in vertical mode.
- void setValueRange (int minValue, int maxValue, int newValue)

  Sets the value range of the slider.
- void setValueRange (int minValue, int maxValue)

  Sets the value range of the slider.
- void setValue (int value)

  Places the indicator at the specified value.
- int getValue ()

  Gets the current value represented by the indicator.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Types

- enum SliderOrientation { **HORIZONTAL**, **VERTICAL** }

  Values that represent slider orientations.

Protected Member Functions

- virtual void handleClickEvent (const ClickEvent &evt)

  Updates the indicators position.
- virtual void handleDragEvent (const DragEvent &evt)

  Updates the indicators position.
- void updateIndicatorPosition (int16_t position)

  Updates the indicator position described by position.
- int16_t valueToPosition (int value)

  Translate a value in the value range to a corresponding indicator position.
- int positionToValue (int16_t position)

Translate a position in the indicator position range to the corresponding value.

- uint16_t getIndicatorRadius ()

  Gets the indicator radius.
- int getIndicatorPositionRangeSize ()

  Gets the indicator position range size.
- int getValueRangeSize ()

  Gets the value range size.

Protected Attributes

- SliderOrientation sliderOrientation

  The slider orientation.
- int currentValue

  The current value represented of the slider.
- int valueRangeMin

  The value range min.
- int valueRangeMax

  The value range max.
- Image background

  The background image.
- Image backgroundSelected

  The backgroundSelected image.
- Image indicator

  The indicator image.
- Container backgroundSelectedViewPort

  The backgroundSelected view port. Controls the visible part of the backgroundSelected image.
- int16_t indicatorMinPosition

  The minimum position of the indicator (either x coordinate in horizontal mode or y coordinate in vertical mode)
- int16_t indicatorMaxPosition

  The maximum position of the indicator (either x coordinate in horizontal mode or y coordinate in vertical mode)
- GenericCallback< const Slider &, int > * startValueCallback

  The start value callback (called when an interaction with the indicator is initiated)
- GenericCallback< const Slider &, int > * stopValueCallback

  The stop value callback (called when an interaction with the indicator ends)
- GenericCallback< const Slider &, int > * newValueCallback

  The new value callback (called when the indicator is moved)

Additional Inherited Members

18.125.1 Detailed Description

A slider is a graphical element with which the user may set a value by moving an indicator or by clicking the slider. The slider can operate in horizontal or vertical mode.

The slider has two bitmaps. One bitmap is used on one side of the indicator. The other is used on the other side. They can be used in indicating the part of the slider value range that is currently selected.

The slider operates on an integer value range that can be set by the user.

18.125.2 Constructor & Destructor Documentation

18.125.2.1 Slider ( )

Default constructor. Set the value range to default 0-100.

18.125.3 Member Function Documentation

18.125.3.1 int getIndicatorPositionRangeSize ( ) [protected]

Returns

The indicator position range size.

18.125.3.2 uint16_t getIndicatorRadius ( ) [protected]

Returns

The the indicator radius.

18.125.3.3 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_BUTTON.

Reimplemented from Container.

18.125.3.4 int getValue ( ) [inline]

Gets the current value represented by the indicator.

Returns

The current value.

18.125.3.5 int getValueRangeSize ( ) [protected]

Returns

The value range size.

18.125.3.6 void handleClickEvent (const ClickEvent & *evt*) [protected], [virtual]

Parameters

evt	The event.

Reimplemented from Drawable.

18.125.3.7 void handleDragEvent (const DragEvent & *evt*) [protected], [virtual]

Parameters

evt	The event.

Reimplemented from Drawable.

18.125.3.8 int positionToValue (int16_t *position*) [protected]

Translate a position (x coordinate in horizontal mode and y in vertical mode) in the indicator position range to the corresponding value in the value range.

Parameters

position	The position.

Returns

The value that corresponds to the coordinate.

18.125.3.9 void setBitmaps (const Bitmap & *sliderBackground,* const Bitmap & *sliderBackgroundSelected,* const Bitmap & *indicator*)

Sets all the bitmaps for the Slider. The slider show the sliderBackgroundSelected bitmap in the region of the slider that is selected, that is the area to the left of the indicator for a horizontal slider and below the indicator for a vertical slider. To ignore this effect just add the same bitmap for both the sliderBackground and the sliderBackground↩ Selected.

Parameters

slider↩ Background	The slider background with the slider range unselected.
slider↩ Background↩ Selected	The slider background with the slider range selected.
indicator	The indicator.

18.125.3.10 void setBitmaps (const BitmapId *sliderBackground,* const BitmapId *sliderBackgroundSelected,* const BitmapId *indicator*)

Sets all the bitmaps for the Slider. The slider show the sliderBackgroundSelected bitmap in the region of the slider that is selected, that is the area to the left of the indicator for a horizontal slider and below the indicator for a vertical slider. To ignore this effect just add the same bitmap for both the sliderBackground and the sliderBackground↩ Selected.

Parameters

slider↩ Background	The slider background.
slider↩ Background↩ Selected	The slider background selected.
indicator	The indicator.

18.125.3.11 void setNewValueCallback (GenericCallback< const Slider &, int > & *callback*) [inline]

Associates an action to be performed when the slider changes its value.

Parameters

callback	The callback to be executed. The callback will be given a reference to the Slider and the current value of the slider.

See also

GenericCallback

18.125.3.12 void setStartValueCallback (GenericCallback< const Slider &, int > & *callback*) [inline]

Associates an action to be performed when an interaction (drag or click) with the slider is initiated.

Parameters

callback	The callback to be executed. The callback will be given a reference to the Slider and the current value of the slider at interaction start.

See also

GenericCallback

18.125.3.13 void setStopValueCallback (GenericCallback< const Slider &, int > & *callback*) [inline]

Associates an action to be performed when an interaction with the slider ends (i.e. drag/click).

Parameters

callback	The callback to be executed. The callback will be given a reference to the Slider and the current value of the slider at interaction end.

See also

GenericCallback

18.125.3.14 void setupHorizontalSlider (uint16_t *backgroundX*, uint16_t *backgroundY*, uint16_t *indicatorY*, uint16_t *indicatorMinX*, uint16_t *indicatorMaxX*)

Sets up the slider in horizontal mode with the range going from the left to right.

Places the backgrounds and the indicator inside the Slider container. It is possible to place the end points of the indicator outside the background image if it needs to go beyond the boundaries of the background. The width and height of the Slider will be adjusted appropriately so that both the background and the indicator will be fully visible in both the minimum and maximum indicator positions.

Note that the x and y position of the Slider will either be the left/top of the background or the left/top of the indicator in its minimum x coordinate.

Calls setValue with the current value (default 0) and triggers the newSliderValue callback.

Parameters

backgroundX	The background x coordinate inside the slider.
backgroundY	The background y coordinate inside the slider.

indicatorY	The indicator y coordinate inside the slider.
indicatorMinX	The indicator minimum x coordinate inside the slider. This is the position used when the slider is at its minimum value. Must be less than indicatorMaxX.
indicatorMaxX	The indicator maximum x coordinate inside the slider. This is the position used when the slider is at its maximum value. Must be greater than indicatorMinX.

18.125.3.15 void setupVerticalSlider (uint16_t *backgroundX,* uint16_t *backgroundY,* uint16_t *indicatorX,* uint16_t *indicatorMinY,* uint16_t *indicatorMaxY*)

Sets up the slider in vertical mode with the range going from the bottom to top.

Places the backgrounds and the indicator inside the Slider container. It is possible to place the end points of the indicator outside the background image if it needs to go beyond the boundaries of the background. The width and height of the Slider will be adjusted appropriately so that both the background and the indicator will be fully visible in both the minimum and maximum indicator positions.

Note that the x and y position of the Slider will either be the left/top of the background or the left/top of the indicator in its minimum y coordinate.

Calls setValue with the current value (default 0) and triggers the newSliderValue callback.

Parameters

backgroundX	The background x coordinate inside the slider.
backgroundY	The background y coordinate inside the slider.
indicatorX	The indicator x coordinate inside the slider.
indicatorMinY	The indicator minimum y coordinate inside the slider. This is the position used when the slider is at its maximum value. Must be less than indicatorMaxX.
indicatorMaxY	The indicator maximum y coordinate inside the slider. This is the position used when the slider is at its minimum value. Must be greater than indicatorMinX.

18.125.3.16 void setValue (int *value*)

Places the indicator at the specified value relative to the specified value range. Values beyond the value range will be rounded to the min/max value in the value range.

Note that the value update triggers a newSliderValue callback just as a drag or click does.

Note that if the value range is larger than the number of pixels specified for the indicator min and max some values will not be represented by the slider and thus is not possible to set with this method. In this case the value will be rounded to the nearest value that is represented in the current setting.

Parameters

value	The value.

18.125.3.17 void setValueRange (int *minValue,* int *maxValue,* int *newValue*)

Sets the value range of the slider. Values accepted and returned by the slider will be in this range.

The slider will set its value to the specified new value.

Note that if the range is larger than the number of pixels specified for the indicator min and max some values will not be represented by the slider.

Parameters

minValue	The minimum value. Must be less than maxValue.
maxValue	The maximum value. Must be greater than minValue.
newValue	The new value.

18.125.3.18 void setValueRange (int *minValue*, int *maxValue*)

Sets the value range of the slider. Values accepted and returned by the slider will be in this range.

The slider will set its value to the current value or round to minValue or maxValue if the current value is outside the new range.

Note that if the range is larger than the number of pixels specified for the indicator min and max some values will not be represented by the slider.

Parameters

minValue	The minimum value. Must be less than maxValue.
maxValue	The maximum value. Must be greater than minValue.

18.125.3.19 void updateIndicatorPosition (int16_t *position*) [protected]

Updates the indicator position described by position. Calls the newSliderValueCallback with the new value.

Parameters

position	The position (x coordinate in horizontal mode and y coordinate in vertical mode).

18.125.3.20 int16_t valueToPosition (int *value*) [protected]

Translate a value in the value range to a position in the indicator position range (x coordinate in horizontal mode and y in vertical mode).

Parameters

value	The value.

Returns

The coordinate that corresponds to the value.

18.126 SlideTransition< templateDirection > Class Template Reference

A Transition that slides from one screen to the next.

```
#include <touchgfx/transitions/SlideTransition.hpp>
```

Public Member Functions

- SlideTransition (const uint8_t transitionSteps=20)

  Constructor.
- virtual ~SlideTransition ()

  Destructor.
- virtual void handleTickEvent ()

Handles the tick event when transitioning.

- virtual void tearDown ()

  Tear down.

- virtual void init ()

  Initializes this object.

Protected Member Functions

- virtual void initMoveDrawable (Drawable &d)

  Moves the Drawable to its initial position.

- virtual void tickMoveDrawable (Drawable &d)

  Moves the Drawable.

Protected Attributes

- SnapshotWidget snapshot

  The SnapshotWidget that is moved when transitioning.

- SnapshotWidget * snapshotPtr

  Pointer pointing to the snapshot used in this transition.The snapshot pointer.

18.126.1 Detailed Description

template<Direction templateDirection>class touchgfx::SlideTransition< templateDirection >

A Transition that slides from one screen to the next. It does so by moving a SnapShotWidget with a snapshot of the Screen transitioning away from, and by moving the contents of Screen transitioning to.

Template Parameters

templateDirection	Type of the template direction.

See also

Transition

18.126.2 Constructor & Destructor Documentation

18.126.2.1 SlideTransition (const uint8_t *transitionSteps* = `20`) `[inline]`

Constructor.

Parameters

transitionSteps	Number of steps in the transition animation.

18.126.2.2 ~SlideTransition ( ) `[inline],[virtual]`

Destructor.

18.126.3 Member Function Documentation

18.126.3.1 void handleTickEvent ( ) [inline],[virtual]

Handles the tick event when transitioning. It moves the contents of the Screen's container and a SnapshotWidget with a snapshot of the previous Screen. The direction of the transition determines the direction the contents of the container and the SnapshotWidget moves.

Reimplemented from Transition.

18.126.3.2 void init ( ) [inline],[virtual]

Initializes this object.

See also

Transition::init()

Reimplemented from Transition.

18.126.3.3 void initMoveDrawable (Drawable & *d*) [inline],[protected],[virtual]

Moves the Drawable to its initial position.

Parameters

in	*d*	The Drawable to move.

18.126.3.4 void tearDown ( ) [inline],[virtual]

Tear down.

See also

Transition::teadDown()

Reimplemented from Transition.

18.126.3.5 void tickMoveDrawable (Drawable & *d*) [inline],[protected],[virtual]

Moves the Drawable.

Parameters

in	*d*	The Drawable to move.

18.127 Snapper< T > Class Template Reference

A mix-in that will make class T draggable and able to snap to a position.

```
#include <touchgfx/mixins/Snapper.hpp>
```

Public Member Functions

- Snapper ()

  Default constructor.
- virtual ~Snapper ()

  Destructor.
- virtual void handleDragEvent (const DragEvent &evt)

  Called when dragging the Snapper.
- virtual void handleClickEvent (const ClickEvent &evt)

  Handles the click events when the Snapper is clicked.
- void setSnapPosition (int16_t x, int16_t y)

  Sets the position the Snapper should snap to.
- void setDragAction (GenericCallback< const DragEvent & > &callback)

  Associates an action to be performed when the Snapper is dragged.
- void setSnappedAction (GenericCallback<> &callback)

  Associates an action to be performed when the Snapper is snapped.

18.127.1 Detailed Description

template<class T>class touchgfx::Snapper< T >

A mix-in that will make class T draggable and able to snap to a position when a drag operation has ended. The mix-in is able to perform callbacks when the snapper gets dragged and when the Snapper snaps to its snap position.

Template Parameters

T	specifies the type to enable the Snap behavior to.

See also

Draggable<T>

18.127.2 Constructor & Destructor Documentation

18.127.2.1 Snapper ( ) [inline]

Default constructor.

18.127.2.2 ~Snapper ( ) [inline],[virtual]

Destructor.

18.127.3 Member Function Documentation

18.127.3.1 void handleClickEvent (const ClickEvent & *evt*) [inline],[virtual]

Handles the click events when the Snapper is clicked. It saves its current position as the snap position if the Snapper is pressed. This happens when the drag operation starts.

The snapper will then move to the snap position when the click is released. This happens when the drag operation ends.

Parameters

evt	The click event.

18.127.3.2 void handleDragEvent (const DragEvent & *evt*) [inline], [virtual]

Called when dragging the Snapper. It will delegate the event if a GenericCallback is set with setDragAction.

Parameters

evt	The drag event.

Reimplemented from Draggable< T >.

18.127.3.3 void setDragAction (GenericCallback< const DragEvent & > & *callback*) [inline]

Associates an action to be performed when the Snapper is dragged.

Parameters

callback	The callback will be executed with the DragEvent.

See also

GenericCallback

18.127.3.4 void setSnappedAction (GenericCallback<> & *callback*) [inline]

Associates an action to be performed when the Snapper is snapped.

Parameters

in	*callback*	The callback to be executed on snap.

See also

GenericCallback

18.127.3.5 void setSnapPosition (int16_t *x*, int16_t *y*) [inline]

Sets the position the Snapper should snap to. This position will be overridden with the Snappers current position when the Snapper is pressed.

Parameters

x	The x coordinate.
y	The y coordinate.

18.128 SnapshotWidget Class Reference

A widget that is able to make a snapshot of the area the SnapshotWidget covers.

```
#include <touchgfx/widgets/SnapshotWidget.hpp>
```

Public Member Functions

- SnapshotWidget ()

  Default constructor.
- virtual ~SnapshotWidget ()

  Destructor.
- virtual void draw (const Rect &invalidatedArea) const

  Draws the SnapshotWidget.
- virtual Rect getSolidRect () const

  Gets solid rectangle.
- virtual void makeSnapshot ()

  Makes a snapshot of the area the SnapshotWidget currently covers.
- void setAlpha (const uint8_t &alpha)

  Sets the alpha value.
- uint8_t getAlpha () const

  Gets the current alpha value.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- uint16_t * fbCopy

  Pointer to the animation storage in which the pixel data for this snapshot is stored. Zero if no snapshot has been made.
- uint8_t alpha

  The alpha with which to draw this snapshot.

Additional Inherited Members

18.128.1 Detailed Description

A widget that is able to make a snapshot of the area the SnapshotWidget covers. The SnapshotWidget will show the snapshot captured when it is drawn.

See also

Widget

18.128.2 Constructor & Destructor Documentation

18.128.2.1 SnapshotWidget ( )

Default constructor.

18.128.2.2 ~SnapshotWidget ( ) [virtual]

Destructor.

18.128.3 Member Function Documentation

18.128.3.1 void draw (const Rect & *invalidatedArea*) const [virtual]

Draws the SnapshotWidget. It supports partial drawing, so it only redraws the area described by invalidatedArea.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Implements Drawable.

18.128.3.2 uint8_t getAlpha ( ) const [inline]

Gets the current alpha value.

Returns

The alpha value.

18.128.3.3 Rect getSolidRect ( ) const [virtual]

Gets solid rectangle.

Returns

The solid rectangle.

Implements Drawable.

18.128.3.4 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_SNAPSHOTWIDGET.

Reimplemented from Widget.

18.128.3.5 void makeSnapshot ( ) [virtual]

Makes a snapshot of the area the SnapshotWidget currently covers. This area is defined by setting the dimensions and the position of the SnapshotWidget.

18.128.3.6 void setAlpha (const uint8_t & *alpha*) [inline]

Sets the alpha value.

Parameters

alpha	The alpha value.

18.129 LCD::StringVisuals Struct Reference

The visual elements when writing a string.

`#include <touchgfx/lcd/LCD.hpp>`

Public Member Functions

- StringVisuals ()

  Construct an empty StringVisuals object.
- StringVisuals (const Font *font, colortype color=0, uint8_t alpha=255, Alignment alignment=LEFT, uint16_t linespace=0, TextRotation rotation=TEXT_ROTATE_0, TextDirection textDirection=TEXT_DIRECTION_LTR, uint8_t indentation=0)

  Construct a StringVisual object for rendering text.

Public Attributes

- const Font * font

  The font to use.
- Alignment alignment

  The alignment to use. Default is LEFT.
- TextDirection textDirection

  The direction to use. Default is LTR.
- TextRotation rotation

  Orientation (rotation) of the text. Default is TEXT_ROTATE_0.
- colortype color

  RGB color value. Default is 0 (black).
- uint8_t alpha

  8-bit alpha value. Default is 255 (solid).
- uint16_t linespace

  Line space in pixels for multiline strings. Default is 0.
- uint8_t indentation

  Indentation of text inside rectangle. Text will start this far from the left/right edge.

18.129.1 Detailed Description

The visual elements when writing a string.

18.129.2 Constructor & Destructor Documentation

18.129.2.1 StringVisuals ( ) [inline]

Construct an empty StringVisuals object.

18.129.2.2 StringVisuals (const Font * *font*, colortype *color* = 0, uint8_t *alpha* = 255, Alignment *alignment* = LEFT, uint16_t *linespace* = 0, TextRotation *rotation* = TEXT_ROTATE_0, TextDirection *textDirection* = TEXT_DIRECTION_LTR, uint8_t *indentation* = 0) [inline]

Parameters

font	The Font with which to draw the text.
color	The color with which to draw the text.
alpha	Alpha blending. Default value is 255 (solid)

alignment	How to align the text.
linespace	Line space in pixels between each line, in case the text contains newline characters.
rotation	How to rotate the text.
textDirection	The text direction.
indentation	The indentation of the text from the left and right of the text area rectangle.

18.130 TextArea Class Reference

This widget is capable of showing a text area on the screen.

```
#include <touchgfx/widgets/TextArea.hpp>
```

Public Member Functions

- TextArea ()

  Default constructor.
- virtual Rect getSolidRect () const

  Gets solid rectangle.
- void setColor (colortype color)

  Sets the color of the text.
- colortype getColor () const

  Gets the color of the text.
- void setAlpha (uint8_t alpha)

  Sets the alpha value of the text.
- uint8_t getAlpha () const

  Gets the alpha value of the text.
- virtual void setBaselineY (int16_t baselineY)

  Adjusts the TextArea y coordinate to place the text at the specified baseline.
- virtual void setXBaselineY (int16_t x, int16_t baselineY)

  Adjusts the TextArea y coordinate to place the text at the specified baseline.
- void setLinespacing (uint16_t space)

  Sets the linespacing of the textarea.
- uint16_t getLinespacing () const

  Gets the linespacing of the textarea.
- void setIndentation (uint8_t indent)

  Sets the indentation for the text.
- uint8_t getIndentation ()

  Gets the indentation.
- virtual int16_t getTextHeight ()

  Gets the total height needed by the text.
- virtual uint16_t getTextWidth () const

  Gets the width in pixels of the current associated text in the current selected language.
- virtual void draw (const Rect &area) const

  Draws the text.
- void setTypedText (TypedText t)

  Sets the TypedText of the text area.
- TypedText getTypedText () const

  Gets the TypedText of the text area.
- void setRotation (const TextRotation rotation=TEXT_ROTATE_0)

  Sets rotation of the text in the TextArea.

- TextRotation getRotation () const

  Gets rotation of the text in the TextArea.
- void resizeToCurrentText ()

  Sets the dimensions of the TextArea.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- TypedText typedText

  The TypedText to display.
- TextRotation rotation

  The text rotation to use.
- colortype color

  The color to use.
- uint16_t linespace

  The line spacing to use, in pixels, in case the text contains newlines.
- uint8_t alpha

  The alpha to use.
- uint8_t indentation

  The indentation of the text inside the text area.

Additional Inherited Members

18.130.1 Detailed Description

This widget is capable of showing a text area on the screen. A TextArea can display a TypedText. Optional configuration include text color.

Example text_example shows how to use a TextArea.

Note

A TextArea just holds a pointer to the text displayed. The developer must ensure that the pointer remains valid when drawing.

See also

TypedText for information about text
TextAreaWithOneWildcard,
TextAreaWithTwoWildcards for displaying texts containing wildcards.

18.130.2 Constructor & Destructor Documentation

18.130.2.1 TextArea ( ) [inline]

Create an empty text area. Default color is black.

18.130.3 Member Function Documentation

18.130.3.1 void draw (const Rect & *area*) const [virtual]

Draws the text. Called automatically.

Parameters

area	The invalidated area.

Implements Drawable.

Reimplemented in TextAreaWithTwoWildcards, and TextAreaWithOneWildcard.

18.130.3.2 uint8_t getAlpha ( ) const [inline]

Gets the alpha value of the text.

Returns

The alpha value. 255 = completely solid. 0 = invisible.

18.130.3.3 colortype getColor ( ) const [inline]

Gets the color of the text.

Returns

The color to used for drawing the text.

18.130.3.4 uint8_t getIndentation ( ) [inline]

Gets the indentation.

Returns

The indentation.

See also

setIndetation

18.130.3.5 uint16_t getLinespacing ( ) const [inline]

Gets the linespacing of the textarea.

Returns

The linespacing.

18.130.3.6 TextRotation getRotation ( ) const [inline]

Gets rotation of the text in the TextArea.

Returns

The rotation of the text.

18.130.3.7 Rect getSolidRect ( ) const [inline],[virtual]

Gets solid rectangle.

Returns

the largest solid rectangle for this widget. For textareas, this is an empty area.

Implements Drawable.

18.130.3.8 int16_t getTextHeight ( ) [virtual]

Gets the total height needed by the text, taking number of lines and line spacing into consideration.

Returns

the total height needed by the text.

Reimplemented in TextAreaWithTwoWildcards, and TextAreaWithOneWildcard.

18.130.3.9 uint16_t getTextWidth ( ) const [virtual]

Gets the width in pixels of the current associated text in the current selected language. In case of multi-lined text the width of the widest line is returned.

Returns

The width in pixels of the current text.

Reimplemented in TextAreaWithTwoWildcards, and TextAreaWithOneWildcard.

18.130.3.10 uint16_t getType ( ) const [inline],[virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TEXTAREA.

Reimplemented from Widget.

Reimplemented in TextAreaWithTwoWildcards, and TextAreaWithOneWildcard.

18.130.3.11 TypedText getTypedText ( ) const [inline]

Gets the TypedText of the text area.

Returns

The currently used TypedText.

18.130.3.12 void resizeToCurrentText ( )

Sets the dimensions of the TextArea to match the width and height of the current associated text for the current selected language.

18.130.3.13 void setAlpha (uint8_t *alpha*) [inline]

Sets the alpha value of the text.

Parameters

alpha	The alpha value. 255 = completely solid. 0 = invisible.

18.130.3.14 void setBaselineY (int16_t *baselineY*) [inline], [virtual]

Adjusts the text areas y coordinate so the text will have its baseline at the specified value. The placements is relative to the specified TypedText so if this changes you have to set the baseline again. Note that setTypedText must be called prior to setting the baseline.

Parameters

baselineY	The y coordinate of the baseline.

18.130.3.15 void setColor (colortype *color*) [inline]

Sets the color of the text.

Parameters

color	The color to use.

18.130.3.16 void setIndentation (uint8_t *indent*) [inline]

Sets the indentation for the text. This is very useful when a font is an italic font where letters such as "j" and "g" extend a lot to the left under the previous characters. if a line starts with a "j" or "g" this letter would either have to be pushed to the right to be able to see all of it, e.g. using spaces which would ruin a multi line text which is left aligned. This could be solved by changing a textarea.setPosition(50,50,100,100) to textarea.setPosition(45,50,110, 100) followed by a textarea.setIndentation(5). Characters that do not extend to the left under the previous characters will be drawn in the same position in either case, but "j" and "g" will be aligned with other lines.

The function getMaxPixelsLeft() will give you the maximum number of pixels any glyph in the font extends to the left.

Parameters

indent	The indentation from left (when left aligned text) and right (when right aligned text).

See also

getMaxPixelsLeft

18.130.3.17 void setLinespacing (uint16_t *space*) [inline]

Sets the linespacing of the textarea.

Parameters

space	The linespacing of use in the textarea.

18.130.3.18 void setRotation (const TextRotation *rotation* = TEXT_ROTATE_0) [inline]

Sets rotation of the text in the TextArea. The value TEXT_ROTATE_0 is the default for normal text. The value TEXT_ROTATE_90 will rotate the text clockwise, thus writing from the top of the display and down. Similarly TE↩XT_ROTATE_180 and TEXT_ROTATE_270 is further rotate 90 degrees clockwise.

Parameters

rotation	The rotation of the text.

18.130.3.19 void setTypedText (TypedText *t*)

Sets the TypedText of the text area. If no prior size has been set the TextArea will be resized to fit the new TypedText.

Parameters

t	The TypedText for this widget to display.

18.130.3.20 void setXBaselineY (int16_t *x*, int16_t *baselineY*) [inline], [virtual]

Adjusts the text areas y coordinate so the text will have its baseline at the specified value. The placements is relative to the specified TypedText so if this changes you have to set the baseline again. Note that setTypedText must be called prior to setting the baseline. The specified x coordinate will be used as the x coordinate of the TextArea.

Parameters

x	The x coordinate of the TextArea.
baselineY	The y coordinate of the baseline.

18.131 TextAreaWithOneWildcard Class Reference

TextArea with one wildcard.

```
#include <touchgfx/widgets/TextAreaWithWildcard.hpp>
```

Public Member Functions

- TextAreaWithOneWildcard ()

  Default constructor.
- virtual int16_t getTextHeight ()

  Gets text height.
- virtual void draw (const Rect &area) const

  Draws TextArea and its text.
- void setWildcard (const Unicode::UnicodeChar *value)

  Sets the wildcard in the text.
- const Unicode::UnicodeChar * getWildcard () const

  Gets the wildcard in the text.
- virtual uint16_t getTextWidth () const

  Gets the width in pixels of the current associated text.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- const Unicode::UnicodeChar * wildcard

  Pointer to the wildcard string. Must be zero-terminated.

Additional Inherited Members

18.131.1 Detailed Description

TextArea with one wildcard. The format string (i.e. the text pointer set in TextArea::setText) is expected to contain a wildcard s.

See also

TextAreaWithWildcardBase

18.131.2 Constructor & Destructor Documentation

18.131.2.1 TextAreaWithOneWildcard ( ) [inline]

Create an empty text area.

Note

No text can be displayed until a font is set. Default color is black.

18.131.3 Member Function Documentation

18.131.3.1 void draw (const Rect & *area*) const [inline], [virtual]

Draws TextArea and its text if a Font is set and the TypedText associated with the TextArea is valid.

Parameters

area	The invalidated area.

Reimplemented from TextArea.

18.131.3.2 int16_t getTextHeight ( ) [inline], [virtual]

Gets text height.

Returns

The text height.

Reimplemented from TextArea.

18.131.3.3 uint16_t getTextWidth ( ) const [inline], [virtual]

Gets the width in pixels of the current associated text in the current selected language. In case of multi-lined text the width of the widest line is returned.

Returns

The width in pixels of the current text.

Reimplemented from TextArea.

18.131.3.4 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TEXTAREAWITHONEWILDCARD.

Reimplemented from TextArea.

18.131.3.5 const Unicode::UnicodeChar * getWildcard ( ) const [inline]

Gets the wildcard in the text.

Returns

The wildcard used in the text.

18.131.3.6 void setWildcard (const Unicode::UnicodeChar * *value*) [inline]

Sets the wildcard in the text.Must be a zero-terminated UnicodeChar array.

Parameters

value	A pointer to the UnicodeChar to set the wildcard to.

18.132 TextAreaWithTwoWildcards Class Reference

TextArea with two wildcards.

```
#include <touchgfx/widgets/TextAreaWithWildcard.hpp>
```

Public Member Functions

- TextAreaWithTwoWildcards ()

  Default constructor.
- virtual int16_t getTextHeight ()

  Gets text height.
- virtual void draw (const Rect &area) const

  Draws TextArea and its text.
- void setWildcard1 (const Unicode::UnicodeChar *value)

  Sets the first wildcard in the text.
- const Unicode::UnicodeChar * getWildcard1 () const

  Gets the first wildcard in the text.
- void setWildcard2 (const Unicode::UnicodeChar *value)

  Sets the second wildcard in the text.
- const Unicode::UnicodeChar * getWildcard2 () const

  Gets the second wildcard in the text.
- virtual uint16_t getTextWidth () const

  Gets the width in pixels of the current associated text.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- const Unicode::UnicodeChar * wc1

  Pointer to the first wildcard string. Must be zero-terminated.
- const Unicode::UnicodeChar * wc2

  Pointer to the second wildcard string. Must be zero-terminated.

Additional Inherited Members

18.132.1 Detailed Description

TextArea with two wildcards. The format string (i.e. the text pointer set in TextArea::setText) is expected to contain two wildcards s.

See also

TextAreaWithWildcardBase

18.132.2 Constructor & Destructor Documentation

18.132.2.1 TextAreaWithTwoWildcards ( ) [inline]

Create an empty text area.

Note

No text can be displayed until a font is set. Default color is black.

18.132.3 Member Function Documentation

18.132.3.1 void draw (const Rect & *area*) const [inline], [virtual]

Draws TextArea and its text if a Font is set and the TypedText associated with the TextArea is valid.

Parameters

area	The invalidated area.

Reimplemented from TextArea.

18.132.3.2 int16_t getTextHeight ( ) [inline], [virtual]

Gets text height.

Returns

The text height.

Reimplemented from TextArea.

18.132.3.3 uint16_t getTextWidth ( ) const [inline], [virtual]

Gets the width in pixels of the current associated text in the current selected language. In case of multi-lined text the width of the widest line is returned.

Returns

The width in pixels of the current text.

Reimplemented from TextArea.

18.132.3.4 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TEXTAREAWITHTWOWILDCARDS.

Reimplemented from TextArea.

18.132.3.5 const Unicode::UnicodeChar * getWildcard1 ( ) const [inline]

Gets the first wildcard in the text.

Returns

The first wildcard from a TextArea with two wildcards.

18.132.3.6 const Unicode::UnicodeChar * getWildcard2 ( ) const [inline]

Gets the second wildcard in the text.

Returns

The second wildcard from a TextArea with two wildcards.

18.132.3.7 void setWildcard1 (const Unicode::UnicodeChar * *value*) [inline]

Sets the first wildcard in the text. Must be a zero-terminated UnicodeChar array.

Parameters

value	A pointer to the UnicodeChar to set the wildcard to.

18.132.3.8 void setWildcard2 (const Unicode::UnicodeChar * *value*) [inline]

Sets the second wildcard in the text. Must be a zero-terminated UnicodeChar array.

Parameters

value	A pointer to the UnicodeChar to set the wildcard to.

18.133 TextAreaWithWildcardBase Class Reference

Base class for TextAreas displaying texts with one or more wildcards.

```
#include <touchgfx/widgets/TextAreaWithWildcard.hpp>
```

Public Member Functions

- TextAreaWithWildcardBase ()

  Create an empty text area.
- int16_t calculateTextHeight (const Unicode::UnicodeChar *format,...)

  Gets the total height needed by the text.

Additional Inherited Members

18.133.1 Detailed Description

Base class for TextAreas displaying texts with one or more wildcards.

See also

TextAreaWithOneWildcard
TextAreaWithTwoWildcards

18.133.2 Constructor & Destructor Documentation

18.133.2.1 TextAreaWithWildcardBase ( ) [inline]

Create an empty text area.

Note

No text can be displayed until a font is set. Default color is black.

18.133.3 Member Function Documentation

18.133.3.1 int16_t calculateTextHeight (const Unicode::UnicodeChar * *format*, ...)

Gets the total height needed by the text. Determined by number of lines and linespace. The number of wildcards in the text should match the number of values for the wildcards.

Parameters

format	The text containing s wildcards.
...	Variable arguments providing additional information.

Returns

the total height needed by the text.

18.134 TextProvider Class Reference

The TextProvider is used in drawing basic and wildcard strings.

```
#include <touchgfx/TextProvider.hpp>
```

Public Member Functions

- TextProvider ()

  Default constructor.

- void initialize (const Unicode::UnicodeChar *format, va_list pArg)

  Initializes the TextProvider.
- Unicode::UnicodeChar getNextChar (TextDirection direction=TEXT_DIRECTION_LTR)

  Gets the next character.

Static Public Attributes

- static const uint32_t MAX_32BIT_INTEGER_DIGITS = 33U

  Max number of digits used for the text representation of a 32 bit integer.

18.134.1 Detailed Description

The TextProvider is used in drawing basic and wildcard strings. The TextProvider enables wildcard expansion of the string at the time it is written to the LCD.

It provides printf formatted text strings one character at the time, without the need for a user provided buffer to store the text string.

18.134.2 Constructor & Destructor Documentation

18.134.2.1 TextProvider ( )

Empty constructor. The user must call initialize() before characters can be provided.

18.134.3 Member Function Documentation

18.134.3.1 Unicode::UnicodeChar getNextChar (TextDirection *direction* = TEXT_DIRECTION_LTR)

Gets the next character.

Parameters

direction	The direction.

Returns

The next character of the expanded string or 0 if end of string i reached.

18.134.3.2 void initialize (const Unicode::UnicodeChar * *format*, va_list *pArg*)

Initializes the TextProvider.

Note

Support formats: c, s, %%, u, i, d, o, x.

Parameters

format	The string to format.
pArg	Format arguments in the form of a va_list.

18.135 Texts Class Reference

Class for setting language and getting texts.

`#include <touchgfx/Texts.hpp>`

Public Member Functions

- const Unicode::UnicodeChar * getText (TypedTextId id) const

  Get text in the set language.

Static Public Member Functions

- static void setLanguage (LanguageId id)

  Sets the current language for texts.
- static LanguageId getLanguage ()

  Gets the current language.

18.135.1 Detailed Description

Class for setting language and getting texts. The language set will determine which texts will be used.

18.135.2 Member Function Documentation

18.135.2.1 static LanguageId getLanguage ( ) [inline],[static]

Gets the current language.

Returns

The id of the language.

18.135.2.2 const Unicode::UnicodeChar * getText (TypedTextId *id*) const [inline]

Get text in the set language.

Parameters

id	The id of the text to lookup.

Returns

The text.

18.135.2.3 static void setLanguage (LanguageId *id*) [static]

Sets the current language for texts.

Parameters

id	The id of the language.

18.136 TextureMapper Class Reference

The TextureMapper class is a widget capable of drawing a transformed image.

```
#include <touchgfx/widgets/TextureMapper.hpp>
```

Public Types

- enum RenderingAlgorithm { **NEAREST_NEIGHBOR**, **BILINEAR_INTERPOLATION** }

  Rendering algorithms of the image.

Public Member Functions

- TextureMapper ()

  Default constructor.
- virtual ~TextureMapper ()

  Destructor.
- virtual void setBitmap (const Bitmap &bmp)

  Sets the bitmap for the image.
- Bitmap getBitmap () const

  Gets the bitmap for the image.
- virtual void draw (const Rect &invalidatedArea) const

  Draws the given invalidated area.
- virtual Rect getSolidRect () const

  Gets solid rectangle.
- virtual void setRenderingAlgorithm (RenderingAlgorithm algorithm)

  Sets the algorithm to be used.
- virtual RenderingAlgorithm getRenderingAlgorithm () const

  Gets the algorithm used when rendering.
- void setAlpha (uint8_t a)

  Sets the global alpha blending value.
- virtual void updateAngles (float xAngle, float yAngle, float zAngle)

  Updates the angles of the image.
- virtual void updateXAngle (float xAngle)

  Updates the x coordinate angle described by xAngle.
- virtual void updateYAngle (float yAngle)

  Updates the y coordinate angle described by yAngle.
- virtual void updateZAngle (float zAngle)

  Updates the z coordinate angle described by zAngle.
- virtual float getXAngle () const

  Get x angle.
- virtual float getYAngle () const

  Get y angle.
- virtual float getZAngle () const

  Get z angle.
- virtual void setScale (float scale)

Sets the scale of the image.

- virtual float getScale () const

  Gets the scale.
- virtual void setOrigo (float x, float y, float z)

  Sets the transformation origo.
- virtual void setOrigo (float x, float y)

  Sets the transformation origo.
- virtual float getOrigoX () const

  Gets transformation origo x coordinate.
- virtual float getOrigoY () const

  Gets transformation origo y coordinate.
- virtual float getOrigoZ () const

  Gets transformation origo z coordinate.
- virtual void setCamera (float x, float y)

  Sets the camera coordinate.
- virtual float getCameraX () const

  Gets camera x coordinate.
- virtual float getCameraY () const

  Gets camera y coordinate.
- virtual void setCameraDistance (float d)

  Sets camera distance.
- virtual float getCameraDistance () const

  Gets camera distance.
- virtual void setBitmapPosition (float x, float y)

  Sets bitmap position.
- virtual void setBitmapPosition (int x, int y)

  Sets bitmap position.
- virtual float getBitmapPositionX () const

  Gets bitmap position x coordinate.
- virtual float getBitmapPositionY () const

  Gets bitmap position y coordinate.
- virtual float getX0 () const

  Get X0 coordinate.
- virtual float getX1 () const

  Get X1 coordinate.
- virtual float getX2 () const

  Get X2 coordinate.
- virtual float getX3 () const

  Get X3 coordinate.
- virtual float getY0 () const

  Get Y0 coordinate.
- virtual float getY1 () const

  Get Y1 coordinate.
- virtual float getY2 () const

  Get Y2 coordinate.
- virtual float getY3 () const

  Get Y3 coordinate.
- virtual float getZ0 () const

  Get Z0 coordinate.
- virtual float getZ1 () const

  Get Z1 coordinate.

- virtual float getZ2 () const

  Get Z2 coordinate.
- virtual float getZ3 () const

  Get Z3 coordinate.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Member Functions

- void applyTransformation ()

  Applies the transformation.
- Rect getBoundingRect ()

  Gets bounding rectangle.
- void drawTriangle (const Rect &invalidatedArea, uint16_t *fb, const float *triangleXs, const float *triangleYs, const float *triangleZs, const float *triangleUs, const float *triangleVs) const

  Draw triangle.
- RenderingVariant lookupRenderVariant () const

  Returns the rendering variant based on the bitmap format, alpha value and rendering algorithm.

Protected Attributes

- RenderingAlgorithm currentRenderingAlgorithm

  The current rendering algorithm.
- Bitmap bitmap

  The bitmap to render.
- uint8_t alpha

  An alpha value that is applied to the entire image.
- float xBitmapPosition

  The bitmap position x.
- float yBitmapPosition

  The bitmap position y.
- float xAngle

  The angle x.
- float yAngle

  The angle y.
- float zAngle

  The angle z.
- float scale

  The scale.
- float xOrigo

  The origo x coordinate.
- float yOrigo

  The origo y coordinate.
- float zOrigo

  The origo z coordinate.
- float xCamera

  The camera x coordinate.
- float yCamera

  The camera y coordinate.
- float cameraDistance

The camera distance.

- float imageX0

  The coordinate for the image points.
- float imageY0

  The coordinate for the image points.
- float imageZ0

  The coordinate for the image points.
- float imageX1

  The coordinate for the image points.
- float imageY1

  The coordinate for the image points.
- float imageZ1

  The coordinate for the image points.
- float imageX2

  The coordinate for the image points.
- float imageY2

  The coordinate for the image points.
- float imageZ2

  The coordinate for the image points.
- float imageX3

  The coordinate for the image points.
- float imageY3

  The coordinate for the image points.
- float imageZ3

  The coordinate for the image points.
- uint16_t subDivisionSize

  The size of the affine sub divisions.

Static Protected Attributes

- static const int MINIMAL_CAMERA_DISTANCE = 1

  The minimal camera distance.

18.136.1 Detailed Description

The TextureMapper displays a transformed image. The TextureMapper can be used in effects where an image should be rotated in two or three dimensions.

The image can be freely scaled and rotated in three dimensions. The scaling and rotation is done around the adjustable origin. A virtual camera is applied to the rendered image yielding a perspective impression. The amount of perspective impression can be adjusted. The transformed image is clipped according to the dimensions of the TextureMapper. In order to make the image fully visible the TextureMapper should be large enough to accommodate the transformed image.

Note that the drawing of this widget is not trivial and typically has a significant effect on the mcu load. The number of pixels drawn, the presence of global alpha or per pixel alpha inflicts the computation and should be considered.

Note that this widget does not support 1 bit per pixel color depth.

See also

Widget

18.136.2 Member Enumeration Documentation

18.136.2.1 enum RenderingAlgorithm

Rendering algorithms of the image.

NEAREST_NEIGHBOR: Fast algorithm with medium image quality. Good for fast animations. (Default)

BILINEAR_INTERPOLATION: Slower algorithm but better image quality.

18.136.3 Constructor & Destructor Documentation

18.136.3.1 ~TextureMapper ( ) [virtual]

Destructor.

18.136.4 Member Function Documentation

18.136.4.1 void applyTransformation ( ) [protected]

Transform the bitmap using the supplied origo, scale, rotation and camera. This method is called by all the methods that manipulate origo, scale, rotation and camera.

18.136.4.2 void draw (const Rect & *invalidatedArea*) const [virtual]

Draws the given invalidated area. The part of the transformed image inside the invalidatedArea will be drawn.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Implements Drawable.

18.136.4.3 void drawTriangle (const Rect & *invalidatedArea*, uint16_t * *fb*, const float * *triangleXs*, const float * *triangleYs*, const float * *triangleZs*, const float * *triangleUs*, const float * *triangleVs*) const [protected]

The TextureMapper will draw the transformed bitmap by drawing two triangles. One triangle is created from the points 0,1,2 and the other triangle from the points 1,2,3. The triangle is drawn using the x,y,z values from each point along with the u,v coordinates in the bitmap associated with each point.

Parameters

	invalidatedArea	The invalidated area.
in,out	*fb*	If non-null, the fb.
	triangleXs	The triangle xs.
	triangleYs	The triangle ys.
	triangleZs	The triangle zs.
	triangleUs	The triangle us.
	triangleVs	The triangle vs.

18.136.4.4 Bitmap getBitmap ( ) const [inline]

Gets the bitmap for the image.

Returns

the bitmap.

18.136.4.5 float getBitmapPositionX ( ) const [inline],[virtual]

Gets bitmap position x coordinate.

Returns

The bitmap position x coordinate.

18.136.4.6 float getBitmapPositionY ( ) const [inline],[virtual]

Gets bitmap position y coordinate.

Returns

The bitmap position y coordinate.

18.136.4.7 Rect getBoundingRect ( ) [protected]

Gets bounding rectangle of the transformed bitmap.

Returns

The bounding rectangle.

18.136.4.8 float getCameraDistance ( ) const [inline],[virtual]

Gets camera distance.

Returns

The camera distance.

18.136.4.9 float getCameraX ( ) const [inline],[virtual]

Gets camera x coordinate.

Returns

The camera x coordinate.

18.136.4.10 float getCameraY ( ) const [inline],[virtual]

Gets camera y coordinate.

Returns

The camera y coordinate.

18.136.4.11 float getOrigoX ( ) const [inline], [virtual]

Gets transformation origo x coordinate.

Returns

The transformation origo x coordinate.

18.136.4.12 float getOrigoY ( ) const [inline], [virtual]

Gets transformation origo y coordinate.

Returns

The transformation origo y coordinate.

18.136.4.13 float getOrigoZ ( ) const [inline], [virtual]

Gets transformation origo z coordinate.

Returns

The transformation origo z coordinate.

18.136.4.14 RenderingAlgorithm getRenderingAlgorithm ( ) const [inline], [virtual]

Gets the algorithm used when rendering.

Returns

The algorithm used when rendering.

18.136.4.15 float getScale ( ) const [inline], [virtual]

Gets the scale.

Returns

The scale.

18.136.4.16 Rect getSolidRect ( ) const [virtual]

Gets solid rectangle.

Returns

largest possible solid rect.

See also

Drawable::getSolidRect()

Implements Drawable.

18.136.4.17 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TEXTUREMAPPER.

Reimplemented from Widget.

18.136.4.18 float getX0 ( ) const [inline], [virtual]

Get the x coordinate of the top left corner of the transformed bitmap.

Returns

The X0 coordinate.

18.136.4.19 float getX1 ( ) const [inline], [virtual]

Get the x coordinate of the top right corner of the transformed bitmap.

Returns

The X1 coordinate.

18.136.4.20 float getX2 ( ) const [inline], [virtual]

Get the x coordinate of the bottom right of the transformed bitmap.

Returns

The X2 coordinate.

18.136.4.21 float getX3 ( ) const [inline], [virtual]

Get the x coordinate of the bottom left corner of the transformed bitmap.

Returns

The X3 coordinate.

18.136.4.22 float getXAngle ( ) const [inline], [virtual]

Get x angle.

Returns

The x angle.

18.136.4.23 float getY0 ( ) const [inline],[virtual]

Get the y coordinate of the top left corner of the transformed bitmap.

Returns

The Y0 coordinate.

18.136.4.24 float getY1 ( ) const [inline],[virtual]

Get the y coordinate of the top right corner of the transformed bitmap.

Returns

The Y1 coordinate.

18.136.4.25 float getY2 ( ) const [inline],[virtual]

Get the y coordinate of the bottom right corner of the transformed bitmap.

Returns

The Y2 coordinate.

18.136.4.26 float getY3 ( ) const [inline],[virtual]

Get the y coordinate of the bottom left corner of the transformed bitmap.

Returns

The Y3 coordinate.

18.136.4.27 float getYAngle ( ) const [inline],[virtual]

Get y angle.

Returns

The y angle.

18.136.4.28 float getZ0 ( ) const [inline],[virtual]

Get the z coordinate of the top left corner of the transformed bitmap.

Returns

The Z0 coordinate.

18.136.4.29 float getZ1 ( ) const [inline],[virtual]

Get the z coordinate of the top right corner of the transformed bitmap.

Returns

The Z1 coordinate.

18.136.4.30 float getZ2 ( ) const [inline], [virtual]

Get the z coordinate of the bottom right corner of the transformed bitmap.

Returns

The Z2 coordinate.

18.136.4.31 float getZ3 ( ) const [inline], [virtual]

Get the z coordinate of the bottom left corner of the transformed bitmap.

Returns

The Z3 coordinate.

18.136.4.32 float getZAngle ( ) const [inline], [virtual]

Get z angle.

Returns

The z angle.

18.136.4.33 RenderingVariant lookupRenderVariant ( ) const [protected]

Returns the rendering variant based on the bitmap format, alpha value and rendering algorithm.

Returns

The RenderingVariant.

18.136.4.34 void setAlpha (uint8_t *a*) [inline]

Sets the global alpha blending value.

Parameters

a	new alpha.

18.136.4.35 void setBitmap (const Bitmap & *bmp*) [virtual]

Sets the bitmap for the image. Note that the width and height of the TextureMapper is set to the size of the image.

Parameters

bmp	The bitmap to be used by the widget.

18.136.4.36 void setBitmapPosition (float *x*, float *y*) [inline], [virtual]

Sets the position of the bitmap within the TextureMapper. The bitmap is clipped against the dimensions of the TextureMapper.

Parameters

x	The x coordinate.
y	The y coordinate.

18.136.4.37 void setBitmapPosition (int *x*, int *y*) [inline], [virtual]

Sets the position of the bitmap within the TextureMapper. The bitmap is clipped against the dimensions of the TextureMapper.

Parameters

x	The x coordinate.
y	The y coordinate.

18.136.4.38 void setCamera (float *x*, float *y*) [inline], [virtual]

Sets the camera coordinate.

Parameters

x	The x coordinate for the camera.
y	The y coordinate for the camera.

18.136.4.39 void setCameraDistance (float *d*) [inline], [virtual]

Sets camera distance. Minimal allowed distance is MINIMAL_CAMERA_DISTANCE. Values below will be set to MINIMAL_CAMERA_DISTANCE.

Parameters

d	The new camera distance.

18.136.4.40 void setOrigo (float *x*, float *y*, float *z*) [inline], [virtual]

Sets the transformation origo.

Parameters

x	The x coordinate.
y	The y coordinate.
z	The z coordinate.

18.136.4.41 void setOrigo (float *x*, float *y*) [inline], [virtual]

Sets the transformation origo.

Parameters

x	The x coordinate.
y	The y coordinate.

18.136.4.42 void setRenderingAlgorithm (RenderingAlgorithm *algorithm*) [inline], [virtual]

Sets the algorithm to be used. Default setting is NEAREST_NEIGHBOR.

Parameters

algorithm	The algorithm to use when rendering.

18.136.4.43 void setScale (float *scale*) [virtual]

Sets the scale of the image.

Parameters

scale	The new scale value.

18.136.4.44 void updateAngles (float *xAngle*, float *yAngle*, float *zAngle*) [virtual]

Updates the angles of the image.

Parameters

xAngle	The new x Angle.
yAngle	The new y Angle.
zAngle	The new x Angle.

18.136.4.45 void updateXAngle (float *xAngle*) [inline], [virtual]

Updates the x coordinate angle described by xAngle.

Parameters

xAngle	The new x angle.

18.136.4.46 void updateYAngle (float *yAngle*) [inline], [virtual]

Updates the y coordinate angle described by yAngle.

Parameters

yAngle	The new y angle.

18.136.4.47 void updateZAngle (float *zAngle*) [inline], [virtual]

Updates the z coordinate angle described by zAngle.

Parameters

zAngle	The new z angle.

18.137 TextureSurface Struct Reference

A texture source. Contains a pointer to the data and the width and height of the texture. The alpha channel is used in 565 rendering with alpha. The stride is the width used when moving to the next line of the texture.

```
#include <touchgfx/hal/Types.hpp>
```

Public Attributes

- const uint16_t * data

  The bits.
- const uint8_t * alphaChannel

  The alpha channel.
- int32_t width

  The width.
- int32_t height

  The height.
- int32_t stride

  The stride.

18.138 ToggleButton Class Reference

A ToggleButton is a Button specialization that swaps the two bitmaps when clicked.

```
#include <touchgfx/widgets/ToggleButton.hpp>
```

Public Member Functions

- ToggleButton ()

  Default constructor.
- virtual void setBitmaps (const Bitmap &bmpReleased, const Bitmap &bmpPressed)

  Sets the bitmaps.
- void forceState (bool activeState)

  Force the button into a specific state.
- bool getState () const

  Gets the state.
- virtual void handleClickEvent (const ClickEvent &event)

  Overrides handleClickEvent.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- Bitmap originalPressed

  Contains the bitmap that was originally being displayed when button is pressed.

Additional Inherited Members

18.138.1 Detailed Description

A ToggleButton is a Button specialization that swaps the two bitmaps when clicked, such that the previous "pressed" bitmap, now becomes the one displayed when button is not pressed.

See also

Button

18.138.2 Constructor & Destructor Documentation

18.138.2.1 ToggleButton ( )

Default constructor.

18.138.3 Member Function Documentation

18.138.3.1 void forceState (bool *activeState*)

Use this function to force the button in one of the two possible states. If button is forced to the active state, then the pressed bitmap from the last call to setBitmaps becomes the one displayed when button is not pressed.

Parameters

activeState	If true, display the bmpPressed bitmap when not pressed. If false display the bmpReleased bitmap.

18.138.3.2 bool getState ( ) const [inline]

Gets the state.

Returns

true if state is currently active.

18.138.3.3 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TOGGLEBUTTON.

Reimplemented from Button.

18.138.3.4 void handleClickEvent (const ClickEvent & *event*) [virtual]

Overrides handleClickEvent in order to swap the bitmaps after being clicked.

Parameters

event	The event to handle.

Reimplemented from AbstractButton.

18.138.3.5 void setBitmaps (const Bitmap & *bmpReleased,* const Bitmap & *bmpPressed*) [inline], [virtual]

Sets the bitmaps.

Note

This specific implementation remembers what bitmap was used as pressed, in order to support the ability to force the state.

Parameters

bmpReleased	The bitmap to show in the "normal" state, ie when button is not pressed.
bmpPressed	The bitmap to show when the button is pressed.

See also

Button::setBitmaps

Reimplemented from Button.

18.139 TouchArea Class Reference

Invisible widget used to capture touch events.

`#include <touchgfx/widgets/TouchArea.hpp>`

Public Member Functions

- TouchArea ()

  Default constructor.
- virtual void draw (const Rect &invalidatedArea) const

  A TouchArea will not draw anything.
- virtual void handleDragEvent (const DragEvent &evt)

  A TouchArea will not move when dragged.
- virtual void handleClickEvent (const ClickEvent &event)

  A TouchArea will refine the handling of click events.
- virtual Rect getSolidRect () const

  A TouchArea has no solid rectangle.
- void setPressedAction (GenericCallback< const AbstractButton & > &callback)

  Associates an action to be performed when the TouchArea is pressed.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Attributes

- GenericCallback< const AbstractButton & > * pressedAction

  The action to perform when the TouchArea is clicked.

Additional Inherited Members

18.139.1 Detailed Description

Invisible widget used to capture touch events. The TouchArea consumes drag events without the widget it self moving.

See also

AbstractButton

18.139.2 Constructor & Destructor Documentation

18.139.2.1 TouchArea ( ) [inline]

Default constructor.

18.139.3 Member Function Documentation

18.139.3.1 void draw (const Rect & *invalidatedArea*) const [inline], [virtual]

A TouchArea will not draw anything.

Parameters

invalidatedArea	The region of the toucharea to draw. Ignored.

Implements Drawable.

18.139.3.2 Rect getSolidRect ( ) const [inline], [virtual]

A TouchArea has no solid rectangle.

Returns

an empty rect.

Implements Drawable.

18.139.3.3 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TOUCHAREA.

Reimplemented from AbstractButton.

18.139.3.4 void handleClickEvent (const ClickEvent & *event*) [virtual]

A TouchArea will refine the handling of click events in order to enable the callback to the pressedAction.

Parameters

event	The event to handle.

Reimplemented from AbstractButton.

18.139.3.5 void handleDragEvent (const DragEvent & *evt*) [inline], [virtual]

A TouchArea will not move when dragged.

Parameters

evt	The event to handle. Ignored.

Reimplemented from Drawable.

18.139.3.6 void setPressedAction (GenericCallback< const AbstractButton & > & *callback*) [inline]

Associates an action to be performed when the TouchArea is pressed.

Parameters

callback	The callback is given a reference to this touch area.

18.140 TouchCalibration Class Reference

Calibrates a touch coordinate.

`#include <touchgfx/transforms/TouchCalibration.hpp>`

Static Public Member Functions

- static void setCalibrationMatrix (const Point *ref, const Point *scr)

  Initializes the calibration matrix based on reference and measured values.
- static void translatePoint (Point &p)

  Translates the specified point using the matrix.

18.140.1 Detailed Description

Class TouchCalibraiton is responsible for translating coordinates (Point) based on matrix of calibration values.

18.140.2 Member Function Documentation

18.140.2.1 static void setCalibrationMatrix (const Point * *ref*, const Point * *scr*) [static]

Initializes the calibration matrix based on reference and measured values.

Parameters

ref	Pointer to array of three reference points.
scr	Pointer to array of three measured points.

18.140.2.2 static void translatePoint (Point & *p*) [static]

Translates the specified point using the matrix. If matrix has not been initialized, p is not modified.

Parameters

in,out	*p*	The point to translate.

18.141 TouchController Class Reference

Basic Touch Controller interface.

`#include <platform/driver/touch/TouchController.hpp>`

Public Member Functions

- virtual ~TouchController ()

  Destructor.
- virtual void init ()=0

Initializes touch controller.

- virtual bool sampleTouch (int32_t &x, int32_t &y)=0

  Checks whether the touch screen is being touched, and if so, what coordinates.

18.141.1 Detailed Description

Basic Touch Controller interface.

18.141.2 Constructor & Destructor Documentation

18.141.2.1 ~TouchController ( ) [inline],[virtual]

Destructor.

18.141.3 Member Function Documentation

18.141.3.1 void init ( ) [pure virtual]

Initializes touch controller.

Implemented in I2CTouchController, SDLTouchController, and NoTouchController.

18.141.3.2 bool sampleTouch (int32_t & *x*, int32_t & *y*) [pure virtual]

Checks whether the touch screen is being touched, and if so, what coordinates.

Parameters

out	*x*	The x position of the touch
out	*y*	The y position of the touch

Returns

True if a touch has been detected, otherwise false.

Implemented in I2CTouchController, NoTouchController, and SDLTouchController.

18.142 Transition Class Reference

The Transition class is the base class for Transitions.

```
#include <touchgfx/transitions/Transition.hpp>
```

Public Member Functions

- Transition ()

  Default constructor.
- virtual ~Transition ()

  Destructor.
- virtual void handleTickEvent ()

  Called for every tick when transitioning.
- bool isDone () const

Query if the transition is done transitioning.

- virtual void tearDown ()

  Tears down the Animation.

- virtual void init ()

  Initializes the transition.

- virtual void setScreenContainer (Container &cont)

  Sets the screen container.

Protected Attributes

- Container * screenContainer

  The screen Container of the Screen transitioning to.

- bool done

  Flag that indicates when the transition is done. This should be set by implementing classes.

18.142.1 Detailed Description

The Transition class is the base class for Transitions. Implementations of Transition defines what happens when transitioning between Screens, which typically involves visual effects. An example of a transition implementation can be seen in example custom_transition_example. The most basic transition is the NoTransition class that does a transition without any visual effects.

See also

NoTransition
SlideTransition

18.142.2 Constructor & Destructor Documentation

18.142.2.1 Transition ( ) [inline]

Constructs the Transition.

18.142.2.2 ~Transition ( ) [inline], [virtual]

Destructor.

18.142.3 Member Function Documentation

18.142.3.1 void handleTickEvent ( ) [inline], [virtual]

Called for every tick when transitioning. Base does nothing.

Reimplemented in SlideTransition< templateDirection >, and NoTransition.

18.142.3.2 void init ( ) [inline], [virtual]

Initializes the transition. Called after the c.tor. when the application changes the transition. Base version does nothing.

Reimplemented in SlideTransition< templateDirection >.

18.142.3.3 bool isDone ( ) const [inline]

Query if the transition is done transitioning. It is the responsibility of the inheriting class to set the underlying done flag.

Returns

True if the transition is done, false otherwise.

18.142.3.4 void setScreenContainer (Container & *cont*) [inline], [virtual]

Sets the screen container. Is used by Screen to enable the transition to access the container.

Parameters

in	*cont*	The container the transition should have access to.

18.142.3.5 void tearDown ( ) [inline], [virtual]

Tears down the Animation. Called before the d.tor. when the application changes the transition. Base version does nothing.

Reimplemented in SlideTransition< templateDirection >.

18.143 TypedText Class Reference

TypedText represents text (as in characters) and typography (as in font and alignment).

```
#include <touchgfx/TypedText.hpp>
```

Classes

- struct TypedTextData

  The data structure for typed texts.

Public Member Functions

- TypedText (const TypedTextId id=TYPED_TEXT_INVALID)

  Construct a typed text.
- TypedTextId getId () const

  Gets the id of the typed text.
- bool hasValidId () const

  Has the TypedText been set to a proper value.
- const Unicode::UnicodeChar * getText () const

  Gets the text associated with this TypedText.
- const Font * getFont () const

  Gets the font associated with this TypedText.
- Alignment getAlignment () const

  Gets the alignment associated with this TypedText.
- TextDirection getTextDirection () const

  Gets the text direction associated with this TypedText.

Static Public Member Functions

- static void registerTypedTextDatabase (const TypedTextData *data, const Font *const *f, const uint16_t n)

  Registers an array of typed texts.
- static void registerTexts (const Texts *t)

  Registers an array of texts.

18.143.1 Detailed Description

TypedText represents text (as in characters) and typography (as in font and alignment). TypedText provides methods for interacting with the text, font and alignment.

Example text_example shows how to use TypedText.

See also

TextArea

18.143.2 Constructor & Destructor Documentation

18.143.2.1 TypedText (const TypedTextId *id* = TYPED_TEXT_INVALID) [inline],[explicit]

Construct a typed text.

Parameters

id	The id of the TypedText.

18.143.3 Member Function Documentation

18.143.3.1 Alignment getAlignment ( ) const [inline]

Gets the alignment associated with this TypedText.

Returns

The alignment.

18.143.3.2 const Font * getFont ( ) const [inline]

Gets the font associated with this TypedText.

Returns

The font.

18.143.3.3 TypedTextId getId ( ) const [inline]

Gets the id of the typed text.

Returns

The id.

18.143.3.4 const Unicode::UnicodeChar * getText ( ) const [inline]

Gets the text associated with this TypedText.

Returns

The text.

18.143.3.5 TextDirection getTextDirection ( ) const [inline]

Gets the text direction associated with this TypedText.

Returns

The alignment.

18.143.3.6 bool hasValidId ( ) const [inline]

Has the TypedText been set to a proper value.

Returns

Is the id valid.

18.143.3.7 static void registerTexts (const Texts * *t*) [inline],[static]

Registers an array of texts.

Parameters

t	The array of texts.

18.143.3.8 static void registerTypedTextDatabase (const TypedTextData * *data*, const Font *const * *f*, const uint16_t *n*) [inline],[static]

Registers an array of typed texts. All typed text instances are bound to this database.

Parameters

data	A reference to the TypedTextData storage array.
f	The fonts associated with the array.
n	The number of typed texts in the array.

18.144 TypedText::TypedTextData Struct Reference

The data structure for typed texts.

```
#include <touchgfx/TypedText.hpp>
```

Public Attributes

- const unsigned char fontIdx

  The font associated with the typed text.
- const Alignment alignment: 2

  The alignment of the typed text.
- const TextDirection direction: 2

  The text direction (LTR,RTL,...) of the typed text.

18.144.1 Detailed Description

The data structure for typed texts.

18.145 UIEventListener Class Reference

This class declares a handler interface for user interface events.

```
#include <touchgfx/UIEventListener.hpp>
```

Public Member Functions

- virtual void handleClickEvent (const ClickEvent &event)

  This handler is invoked when a mouse click or display touch event has been detected by the system.
- virtual void handleDragEvent (const DragEvent &event)

  This handler is invoked when a drag event has been detected by the system.
- virtual void handleGestureEvent (const GestureEvent &event)

  This handler is invoked when a gesture event has been detected by the system.
- virtual void handleKeyEvent (uint8_t c)

  This handler is invoked when a key (or button) event has been detected by the system.
- virtual void handleTickEvent ()

  This handler is invoked when a system tick event has been generated.
- virtual void handlePendingScreenTransition ()

  This handler is invoked when a change screen event is pending.
- virtual ~UIEventListener ()

  Destructor.

18.145.1 Detailed Description

This class declares a handler interface for user interface events, i.e. events generated by the users interaction with the device. With the exception of the system timer tick, all other system events, which are not related to the user interface device peripherals (display, keys etc.) are not part of this interface.

18.145.2 Constructor & Destructor Documentation

18.145.2.1 ~UIEventListener ( ) [inline],[virtual]

Destructor.

18.145.3 Member Function Documentation

18.145.3.1 void handleClickEvent (const ClickEvent & *event*) [inline], [virtual]

This handler is invoked when a mouse click or display touch event has been detected by the system.

Parameters

event	The event data.

Reimplemented in Application.

18.145.3.2 void handleDragEvent (const DragEvent & *event*) [inline],[virtual]

This handler is invoked when a drag event has been detected by the system.

Parameters

event	The event data.

Reimplemented in Application.

18.145.3.3 void handleGestureEvent (const GestureEvent & *event*) [inline],[virtual]

This handler is invoked when a gesture event has been detected by the system.

Parameters

event	The event data.

Reimplemented in Application.

18.145.3.4 void handleKeyEvent (uint8_t *c*) [inline],[virtual]

This handler is invoked when a key (or button) event has been detected by the system.

Parameters

c	The key or button pressed.

Reimplemented in Application.

18.145.3.5 void handlePendingScreenTransition ( ) [inline],[virtual]

This handler is invoked when a change screen event is pending.

Reimplemented in Application, and MVPApplication.

18.145.3.6 void handleTickEvent ( ) [inline],[virtual]

This handler is invoked when a system tick event has been generated. The system tick period is configured in the HAL.

Reimplemented in Application.

18.146 Unicode Class Reference

This class provides simple helper functions for working with 16-bit strings.

```
#include <touchgfx/Unicode.hpp>
```

Public Types

- typedef uint16_t UnicodeChar

Use the UnicodeChar typename when referring to strings.

Static Public Member Functions

- static uint16_t strlen (const UnicodeChar ∗str)

  Gets the length of a 0-terminated unicode string.
- static uint16_t strlen (const char ∗str)

  Gets the length of a 0-terminated string.
- static uint16_t strncpy (UnicodeChar ∗RESTRICT dst, const UnicodeChar ∗RESTRICT src, uint16_t maxchars)

  Copy a string to a destination buffer, UnicodeChar version.
- static uint16_t strncpy (UnicodeChar ∗RESTRICT dst, const char ∗RESTRICT src, uint16_t maxchars)

  Copy a string to a destination buffer, char version.
- static void itoa (int32_t value, UnicodeChar ∗buffer, uint16_t bufferSize, int radix)

  Integer to ASCII conversion.
- static void utoa (uint32_t value, UnicodeChar ∗buffer, uint16_t bufferSize, int radix)

  Integer to ASCII conversion.
- static int atoi (const UnicodeChar ∗s)

  String to integer conversion.
- static UnicodeChar ∗ snprintf (UnicodeChar ∗dst, uint16_t dstSize, const UnicodeChar ∗format,...)

  Formats a string and adds null termination.
- static UnicodeChar ∗ snprintf (UnicodeChar ∗dst, uint16_t dstSize, const char ∗format,...)

  Variant of snprintf.
- static UnicodeChar ∗ snprintfFloats (UnicodeChar ∗dst, uint16_t dstSize, const UnicodeChar ∗format, const float ∗values)

  Variant of snprintf for floats only.
- static uint16_t ∗ snprintfFloat (UnicodeChar ∗dst, uint16_t dstSize, const UnicodeChar ∗format, const float value)

  Variant of snprintf.
- static UnicodeChar ∗ snprintfFloats (UnicodeChar ∗dst, uint16_t dstSize, const char ∗format, const float ∗values)

  Variant of snprintf for floats only.
- static UnicodeChar ∗ snprintfFloat (UnicodeChar ∗dst, uint16_t dstSize, const char ∗format, const float value)

  Variant of snprintf.
- static int strncmp (const UnicodeChar ∗RESTRICT str1, const UnicodeChar ∗RESTRICT str2, uint16_t maxchars)

  Compares up to maxchars characters of the string str1 to those of the string str2.
- static int strncmp_ignore_white_spaces (const UnicodeChar ∗RESTRICT str1, const UnicodeChar ∗RESTRICT str2, uint16_t maxchars)

  Like strncmp except that ignore any spaces in the two strings.

Static Public Attributes

- static const UnicodeChar EMPTY [1]

  An empty string, which should be used instead of a null-pointer to indicate that the a string has no value.

18.146.1 Member Typedef Documentation

18.146.1.1 uint16_t UnicodeChar

Use the UnicodeChar typename when referring to strings.

18.146.2 Member Function Documentation

18.146.2.1 static int atoi (const UnicodeChar * s) [static]

String to integer conversion. Starts conversion at the start of the string. Running digits from here are converted.

Parameters

s	DECIMAL zero-terminated unicode string to convert.

Returns

The converted integer value of the string, 0 if the string does not start with a digit.

18.146.2.2 static void itoa (int32_t *value*, UnicodeChar * *buffer*, uint16_t *bufferSize*, int *radix*) [static]

Integer to ASCII conversion.

Parameters

	value	to convert.
out	*buffer*	to place result in.
	bufferSize	Size of buffer (number of 16-bit values).
	radix	to use (8 for octal, 10 for decimal, 16 for hex)

18.146.2.3 static UnicodeChar * snprintf (UnicodeChar * *dst*, uint16_t *dstSize*, const UnicodeChar * *format*, ...) [static]

Formats a string and adds null termination. The string is formatted like when printf is used.

Support formats: c (element type: char), s (element type: zero-terminated UnicodeChar list), %% (The " character), u, i, d, o, x (all these are integers formatted in radix 10, 10, 10, 8, 16 respectively).

The number formats (u, i, d, o and x) all support %[0][length]X to specify the size of the generated field (length) and whether the number should be prefixed with zeros (or blanks).

Parameters

out	*dst*	Buffer to place the result in.
	dstSize	Size of the buffer.
	format	The string to format.
	...	Variable arguments providing additional information.

Returns

pointer to the first element in the buffer.

See also

snprintfFloat, snprintfFloats

18.146.2.4 static UnicodeChar * snprintf (UnicodeChar * *dst*, uint16_t *dstSize*, const char * *format*, ...) [static]

Variant of snprintf.

Support formats: c (element type: char), s (element type: zero-terminated UnicodeChar list), %% (The " character), u, i, d, o, x (all these are integers formatted in radix 10, 10, 10, 8, 16 respectively).

The number formats (u, i, d, o and x) all support %[0][length]X to specify the size of the generated field (length) and whether the number should be prefixed with zeros (or blanks).

Parameters

out	*dst*	Buffer to place the result in.
	dstSize	Size of the buffer.
in	*format*	The string to format.
	...	Variable arguments providing additional information.

Returns

pointer to the first element in the buffer.

See also

snprintfFloat, snprintfFloats

18.146.2.5 static uint16_t * snprintfFloat (UnicodeChar * *dst*, uint16_t *dstSize*, const UnicodeChar * *format*, const float *value*) [inline], [static]

Variant of snprintf.

Support formats: f (element type: float), %% (The ” character).

The float format (f) support %[0][length]f to specify the size of the generated field (length) and whether the number should be prefixed with zeros (or blanks).

Parameters

out	*dst*	Buffer to place the result in.
	dstSize	Size of the buffer.
in	*format*	The string to format.
	value	Variable arguments providing additional information.

Returns

pointer to the first element in the buffer.

See also

snprintf, snprintfFloats

18.146.2.6 static UnicodeChar * snprintfFloat (UnicodeChar * *dst*, uint16_t *dstSize*, const char * *format*, const float *value*) [inline], [static]

Variant of snprintf.

Support formats: c (element type: char), s (element type: zero-terminated UnicodeChar list), %% (The ” character), u, i, d, o, x (all these are integers formatted in radix 10, 10, 10, 8, 16 respectively).

The number formats (u, i, d, o and x) all support %[0][length]X to specify the size of the generated field (length) and whether the number should be prefixed with zeros (or blanks).

Parameters

out	*dst*	Buffer to place the result in.
	dstSize	Size of the buffer.
in	*format*	The string to format.
	value	Variable arguments providing additional information.

Returns

pointer to the first element in the buffer.

See also

snprintf, snprintfFloats

18.146.2.7 static UnicodeChar * snprintfFloats (UnicodeChar * *dst*, uint16_t *dstSize*, const UnicodeChar * *format*, const float * *values*) [static]

Variant of snprintf for floats only.

Support formats: f (element type: float), %% (The " character).

The float format (f) support %[0][length]f to specify the size of the generated field (length) and whether the number should be prefixed with zeros (or blanks).

Technical note: The reason for having this function is that if the snprintf should support f, it would require using doubles which would include extra code to be linked into the final application. To print a string with mixed e.g. d, s and s, use the following:

UnicodeChar tmpBuf[50]; snprintfFloat(tmpBuf, 50, "%d %5.3f %s", 3.14f); // tmpBuf="%d 3.140 %s" UnicodeChar buf[50]; snprintf(buf, 50, tmpBuf, 42, "Hello World"); // buf="42 3.140 Hello World".

Parameters

out	*dst*	Buffer to place the result in.
	dstSize	Size of the buffer.
in	*format*	The string to format.
in	*values*	Variable arguments providing additional information.

Returns

pointer to the first element in the buffer.

See also

snprintf, snprintfFloat

18.146.2.8 static UnicodeChar * snprintfFloats (UnicodeChar * *dst*, uint16_t *dstSize*, const char * *format*, const float * *values*) [static]

Variant of snprintf for floats only.

Support formats: f (element type: float), %% (The " character).

The float format (f) support %[0][length]f to specify the size of the generated field (length) and whether the number should be prefixed with zeros (or blanks).

Technical note: The reason for having this function is that if the snprintf should support f, it would require using doubles which would include extra code to be linked into the final application. To print a string with mixed e.g. d, s and s, use the following:

UnicodeChar tmpBuf[50]; snprintfFloat(tmpBuf, 50, "%d %5.3f %s", 3.14f); // tmpBuf="%d 3.140 %s" UnicodeChar buf[50]; snprintf(buf, 50, tmpBuf, 42, "Hello World"); // buf="42 3.140 Hello World".

Parameters

out	*dst*	Buffer to place the result in.
	dstSize	Size of the buffer.
in	*format*	The string to format.
in	*values*	Variable arguments providing additional information.

Returns

pointer to the first element in the buffer.

See also

snprintf, snprintfFloat

18.146.2.9 static uint16_t strlen (const UnicodeChar * *str*) [static]

Gets the length of a 0-terminated unicode string.

Parameters

str	The string in question.

Returns

Length of string.

18.146.2.10 static uint16_t strlen (const char * *str*) [static]

Gets the length of a 0-terminated string.

Parameters

str	The string.

Returns

Length of string.

18.146.2.11 static int strncmp (const UnicodeChar *RESTRICT *str1*, const UnicodeChar *RESTRICT *str2*, uint16_t *maxchars*) [static]

Compares up to maxchars characters of the string str1 to those of the string str2. This function starts comparing the first character of each string. If they are equal to each other, it continues with the following pairs until the characters differ, until a terminating null-character is reached, or until maxchars characters match in both strings, whichever happens first.

Parameters

str1	The first string.
str2	The second string.
maxchars	The maximum number of chars to compare.

Returns

Returns an integral value indicating the relationship between the strings: A zero value indicates that the characters compared in both strings are all equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2; And a value less than zero indicates the opposite.

18.146.2.12 static int strncmp_ignore_white_spaces (const UnicodeChar *RESTRICT *str1*, const UnicodeChar *RESTRICT *str2*, uint16_t *maxchars*) [static]

Like strncmp except that ignore any spaces in the two strings.

Parameters

str1	The first string.
str2	The second string.
maxchars	The maximum number of chars to compare.

Returns

Returns an integral value indicating the relationship between the strings: A zero value indicates that the characters compared in both strings are all equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2; And a value less than zero indicates the opposite.

18.146.2.13 static uint16_t strncpy (UnicodeChar *RESTRICT *dst*, const UnicodeChar *RESTRICT *src*, uint16_t *maxchars*) [static]

Copy a string to a destination buffer, UnicodeChar version. Stops if it encounters a zero-termination, in which case the zero-termination is included in the dest string. Otherwise copies maxchars.

Warning

If there is no null-termination among the first n UnicodeChars of src, the string placed in dest will NOT be zero-terminated!

Parameters

out	*dst*	The destination buffer. Must have a size of at least maxchars.
in	*src*	The source string (UnicodeChars)
	maxchars	Maximum number of characters to copy.

Returns

The number of characters copied (excluding zero-termination if encountered)

18.146.2.14 static uint16_t strncpy (UnicodeChar *RESTRICT *dst*, const char *RESTRICT *src*, uint16_t *maxchars*) [static]

Copy a string to a destination buffer, char version. Stops if it encounters a zero-termination, in which case the zero-termination is included in the dest string. Otherwise copies maxchars.

Parameters

out	*dst*	The destination buffer. Must have a size of at least maxchars.
in	*src*	The source string as an array of chars.
	maxchars	Maximum number of characters to copy.

Returns

The number of characters copied (excluding zero-termination if encountered)

```
@warning If there is no null-termination among the first n bytes of src, the
 string placed in dest will NOT be zero-terminated!
```

18.146.2.15 static void utoa (uint32_t *value*, UnicodeChar * *buffer*, uint16_t *bufferSize*, int *radix*) [static]

Integer to ASCII conversion.

Parameters

	value	to convert.
out	*buffer*	to place result in.
	bufferSize	Size of buffer (number of 16-bit values).
	radix	to use (8 for octal, 10 for decimal, 16 for hex)

18.147 Vector< T, capacity > Class Template Reference

A very simple container class using pre-allocated memory.

`#include <touchgfx/hal/Types.hpp>`

Public Member Functions

- Vector ()

  Default constructor.
- T & operator[] (uint16_t idx)

  Index operator.
- const T & operator[] (uint16_t idx) const

  Const version of the index operator.
- void add (T e)

  Adds an element to the Vector if the Vector is not full.
- void remove (T e)

  Removes an element from the Vector if found in the Vector.
- T removeAt (uint16_t index)

  Removes an element at the specified index of the Vector.
- void reverse ()

  Reverses the ordering of the elements in the Vector.
- bool contains (T elem)

  Checks if the Vector contains an element.
- uint16_t size () const

  Gets the current size of the Vector which is the number of elements contained in the Vector.
- bool isEmpty () const

  Query if this object is empty.
- uint16_t maxCapacity () const

  Query the maximum capacity of the vector.
- void clear ()

  Clears the contents of the container.

18.147.1 Detailed Description

template<class T, uint16_t capacity>class touchgfx::Vector< T, capacity >

A very simple container class using pre-allocated memory.

Template Parameters

T	The type of objects this container works on.
capacity	The maximum number of objects this container can store.

18.147.2 Constructor & Destructor Documentation

18.147.2.1 Vector ( ) [inline]

Default constructor. Constructs an empty vector.

18.147.3 Member Function Documentation

18.147.3.1 void add (T *e*) [inline]

Adds an element to the Vector if the Vector is not full. Does nothing if the Vector is full.

Parameters

e	The element to add to the Vector.

18.147.3.2 void clear ( ) [inline]

Clears the contents of the container. It does not destruct any of the elements in the Vector.

18.147.3.3 bool contains (T *elem*) [inline]

Checks if the Vector contains an element. The == operator of the element is used when comparing it with the elements in the Vector.

Parameters

elem	The element.

Returns

true if the Vector contains the element, false otherwise.

18.147.3.4 bool isEmpty ( ) const [inline]

Query if this object is empty.

Returns

true if the Vector contains no elements.

18.147.3.5 uint16_t maxCapacity ( ) const [inline]

Query the maximum capacity of the vector.

Returns

The capacity the Vector was initialized with.

18.147.3.6 T & operator[] (uint16_t *idx*) [inline]

Index operator.

Parameters

idx	The index of the element to obtain.

Returns

A reference to the element placed at index idx.

18.147.3.7 const T & operator[] (uint16_t *idx*) const [inline]

Const version of the index operator.

Parameters

idx	The index of the element to obtain.

Returns

A const reference to the element placed at index idx.

18.147.3.8 void remove (T *e*) [inline]

Removes an element from the Vector if found in the Vector. Does nothing if the element is not found in the Vector. The == operator of the element is used when comparing it with the elements in the Vector.

Parameters

e	The element to remove from the Vector.

18.147.3.9 T removeAt (uint16_t *index*) [inline]

Removes an element at the specified index of the Vector. Will "bubble-down" any remaining elements after the specified index.

Parameters

index	The index to remove.

Returns

The value of the removed element.

18.147.3.10 void reverse ( ) [inline]

Reverses the ordering of the elements in the Vector.

18.147.3.11 uint16_t size ( ) const [inline]

Gets the current size of the Vector which is the number of elements contained in the Vector.

Returns

The size of the Vector.

18.148 Vector4 Class Reference

This class represents a homogeneous 3D vector.

`#include <touchgfx/Math3D.hpp>`

Public Member Functions

- Vector4 ()

  Default constructor.
- Vector4 (float x, float y, float z)

  Constructor.
- Vector4 crossProduct (const Vector4 &operand)

  Cross product.

Additional Inherited Members

18.148.1 Detailed Description

This class represents a homogeneous 3D vector.

See also

quadruple

18.148.2 Constructor & Destructor Documentation

18.148.2.1 Vector4 ( ) [inline]

Default constructor.

18.148.2.2 Vector4 (float *x*, float *y*, float *z*) [inline]

Constructor.

Parameters

x	The x value.
y	The y value.
z	The z value.

18.148.3 Member Function Documentation

18.148.3.1 Vector4 crossProduct (const Vector4 & *operand*) [inline]

Cross product.

Parameters

operand	The second operand.

Returns

The result of the operation.

18.149 View< T > Class Template Reference

This is a generic touchgfx::Screen specialization for normal applications.

`#include <mvp/View.hpp>`

Public Member Functions

- View ()

  Default constructor.
- void bind (T &presenter)

  Binds an instance of a specific Presenter type (subclass) to the View instance.

Protected Attributes

- T * presenter

  Pointer to the Presenter associated with this view.

Additional Inherited Members

18.149.1 Detailed Description

template<class T>class touchgfx::View< T >

This is a generic touchgfx::Screen specialization for normal applications. It provides a link to the Presenter class.

Note

All views in the application must be a subclass of this type.

Template Parameters

T	The type of Presenter associated with this view.

See also

Screen

18.149.2 Constructor & Destructor Documentation

18.149.2.1 View ( ) [inline]

Default constructor.

18.149.3 Member Function Documentation

18.149.3.1 void bind (T & *presenter*) [inline]

Binds an instance of a specific Presenter type (subclass) to the View instance. This function is called automatically when a new presenter/view pair is activated.

Parameters

in	*presenter*	The specific Presenter to be associated with the View.

18.150 Widget Class Reference

A Widget is a Drawable leaf (i.e. not a container).

`#include <touchgfx/widgets/Widget.hpp>`

Public Member Functions

- Widget ()

  Default constructor.
- virtual ~Widget ()

  Destructor.
- virtual void getLastChild (int16_t x, int16_t y, Drawable **last)

  Function for obtaining the the last child of this widget that intersects with the specified point.
- virtual uint16_t getType () const

  For GUI testing only.

Additional Inherited Members

18.150.1 Detailed Description

A Widget is a Drawable leaf (i.e. not a container). It does not currently contain any implementation code, since the Drawable base class handles everything related to leaf nodes. Extend this when implementing custom widgets.

See also

Drawable

18.150.2 Constructor & Destructor Documentation

18.150.2.1 Widget ( ) [inline]

Default constructor.

18.150.2.2 ~Widget ( ) [inline], [virtual]

Destructor.

18.150.3 Member Function Documentation

18.150.3.1 void getLastChild (int16_t *x*, int16_t *y*, Drawable ** *last*) [inline], [virtual]

Function for obtaining the the last child of this widget that intersects with the specified point. Used in input event handling for obtaining the appropriate drawable that should receive the event. Note that input events must be delegated to the last drawable of the tree (meaning highest z-order / front-most drawable).

Only containers can have children, so this implementation simply yields itself as result. The container implementation will filter children that do not intersect with the point or are not visible/enabled, so performing those checks are unnecessary.

Parameters

	x	The point of intersection expressed in coordinates relative to the parent.
	y	The y coordinate.
out	*last*	Result will be placed here.

Implements Drawable.

18.150.3.2 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_WIDGET.

Reimplemented from Drawable.

Reimplemented in TextureMapper, TextArea, TextAreaWithTwoWildcards, RadioButton, AnimatedImage, ScalableImage, TextAreaWithOneWildcard, ButtonWithLabel, ButtonWithIcon, Box, Image, Button, SnapshotWidget, TouchArea, ToggleButton, and AbstractButton.

18.151 ZoomAnimationImage Class Reference

Class for optimizing and wrapping move and zoom operations on ScalableImages.

`#include <touchgfx/containers/ZoomAnimationImage.hpp>`

Public Types

- enum ZoomMode {
  FIXED_CENTER = 0, **FIXED_LEFT**, **FIXED_RIGHT**, **FIXED_TOP**, **FIXED_BOTTOM**, **FIXED_LEFT_AND_TOP**, **FIXED_RIGHT_AND_TOP**,
  FIXED_LEFT_AND_BOTTOM, **FIXED_RIGHT_AND_BOTTOM** }

  A ZoomMode describes in which direction the image will grow/shrink when do a zoom animation. A FIXED direction means that the image will not grow/shrink in that direction.

Public Member Functions

- ZoomAnimationImage ()

  Default constructor.
- virtual ~ZoomAnimationImage ()

  Destructor.
- void startZoomAnimation (int16_t endWidth, int16_t endHeight, uint16_t duration, ZoomMode zoomMode=FIXED_LEFT_AND_TOP, touchgfx::EasingEquation widthProgressionEquation=&touchgfx::EasingEquations::linearEaseNone, EasingEquation heightProgressionEquation=&touchgfx::EasingEquations::linearEaseNone)

  Setup and starts the zoom animation.
- void startZoomAndMoveAnimation (int16_t endX, int16_t endY, int16_t endWidth, int16_t endHeight, uint16_t duration, ZoomMode zoomMode=FIXED_LEFT_AND_TOP, EasingEquation xProgressionEquation=&touchgfx::EasingEquations::linearEaseNone, EasingEquation yProgressionEquation=&touchgfx::EasingEquations::linearEaseNone, EasingEquation widthProgressionEquation=&touchgfx::EasingEquations::linearEaseNone, EasingEquation heightProgressionEquation=&touchgfx::EasingEquations::linearEaseNone)

  Setup and starts the zoom and move animation.

- void cancelZoomAnimation ()

  Cancel zoom animation.
- virtual void handleTickEvent ()

  The tick handler.
- void setBitmaps (const Bitmap &smallBitmap, const Bitmap &largeBitmap)

  Initializes the bitmap of the image to be used.
- Bitmap getSmallBitmap () const

  Gets the small bitmap.
- Bitmap getLargeBitmap () const

  Gets the large bitmap.
- virtual void setPosition (int16_t x, int16_t y, int16_t width, int16_t height)

  Sets the size and position of the image, relative to its parent.
- virtual void setWidth (int16_t width)

  Sets the width of the image.
- virtual void setHeight (int16_t height)

  Sets the height of the image.
- virtual void setDimension (int16_t width, int16_t height)

  Sets the width and height of the image.
- virtual void setScalingMode (ScalableImage::ScalingAlgorithm mode)

  Sets the scaling algorithm of the ScalableImage.
- virtual ScalableImage::ScalingAlgorithm getScalingMode ()

  Gets the scaling algorithm of the ScalableImage.
- virtual void setAlpha (uint8_t alpha)

  Sets the alpha channel for the image.
- virtual uint8_t getAlpha () const

  Gets the current alpha value.
- virtual void setAnimationDelay (uint16_t delay)

  Sets a delay on animations done by the ZoomAnimationImage.
- virtual uint16_t getAnimationDelay () const

  Gets the current animation delay.
- void setAnimationEndedCallback (touchgfx::GenericCallback< const ZoomAnimationImage & > &callback)

  Associates an action to be performed when the animation ends.
- virtual bool isRunning () const

  Is there currently an animation running.
- virtual uint16_t getType () const

  For GUI testing only.

Protected Types

- enum States { ANIMATE_ZOOM, ANIMATE_ZOOM_AND_MOVE, NO_ANIMATION }

  Animation states.

Protected Member Functions

- virtual void updateRenderingMethod ()

  Chooses the optimal rendering of the image given the current width and height.
- virtual void setCurrentState (States state)

  Sets the current animation state.
- virtual void updateZoomAnimationDeltaXY ()

  Calculates the change in X and Y caused by the zoom animation given the current ZoomMode.

Protected Attributes

- States currentState

  The current animation state.
- uint32_t animationCounter

  The progress counter for the animation.
- uint16_t zoomAnimationDelay

  A delay that is applied before animation start. Expressed in ticks.
- touchgfx::Bitmap smallBmp

  The bitmap representing the small image.
- touchgfx::Bitmap largeBmp

  The bitmap representing the large image.
- touchgfx::Image image

  The image for displaying the bitmap when the width/height is equal one of the bitmaps.
- ScalableImage scalableImage

  The scalable image for displaying the bitmap when the width/height is not equal one of the bitmaps.
- ZoomMode currentZoomMode

  The ZoomMode to use by the animation.
- int16_t zoomAnimationStartWidth

  Width of the zoom animation start.
- int16_t zoomAnimationStartHeight

  Height of the zoom animation start.
- int16_t zoomAnimationEndWidth

  Width of the zoom animation end.
- int16_t zoomAnimationEndHeight

  Height of the zoom animation end.
- int16_t zoomAnimationStartX

  The zoom animation start x coordinate.
- int16_t zoomAnimationStartY

  The zoom animation start y coordinate.
- int16_t zoomAnimationDeltaX

  The zoom animation delta x.
- int16_t zoomAnimationDeltaY

  The zoom animation delta y.
- int16_t moveAnimationEndX

  The move animation end x coordinate.
- int16_t moveAnimationEndY

  The move animation end y coordinate.
- uint16_t animationDuration

  Duration of the animation.
- EasingEquation zoomAnimationWidthEquation

  The zoom animation width equation.
- EasingEquation zoomAnimationHeightEquation

  The zoom animation height equation.
- EasingEquation moveAnimationXEquation

  The move animation x coordinate equation.
- EasingEquation moveAnimationYEquation

  The move animation y coordinate equation.
- touchgfx::GenericCallback
  < const ZoomAnimationImage & > * animationEndedAction

  The animation ended action.

Additional Inherited Members

18.151.1 Detailed Description

Class for optimizing and wrapping move and zoom operations on ScalableImages. The ZoomAnimationImage takes two bitmaps representing the same image but at a small and a large resolution. These bitmaps should be the sizes that are used when not animating the image. The ZoomAnimationImage will use an Image for displaying the bitmap when its width and height matches one of them. When it does not it will use a ScalableImage instead. The main idea is that the supplied bitmaps should be the end points of the zoom animation so that it ends up using an Image when not animating. This is, however, not a required. You can animate from and to sizes that are not equal the sizes of the bitmaps. The result is a container that has the high performance of an ordinary image when the size matches the prerendered bitmaps. Moreover it supplies easy to use animation functions that lets you zoom and move the image.

Note

Note that since this container uses the ScalableImage it has the same restrictions. That means no 1 bit per pixel mode.

See also

ScalableImage

18.151.2 Member Enumeration Documentation

18.151.2.1 enum States [protected]

Enumerator

ANIMATE_ZOOM Zoom animation state.

ANIMATE_ZOOM_AND_MOVE Zoom and move animation state.

NO_ANIMATION No animation state.

18.151.3 Constructor & Destructor Documentation

18.151.3.1 ZoomAnimationImage ( )

Creates and initialize the ZoomAnimationImage.

18.151.3.2 ~ZoomAnimationImage ( ) [virtual]

Destroys the ZoomAnimationImage.

18.151.4 Member Function Documentation

18.151.4.1 uint8_t getAlpha ( ) const [virtual]

Gets the current alpha value.

Returns

The current alpha value.

18.151.4.2 uint16_t getAnimationDelay ( ) const [virtual]

Gets the current animation delay.

Returns

The current animation delay. Expressed in ticks.

18.151.4.3 Bitmap getLargeBitmap ( ) const [inline]

Gets the large bitmap.

Returns

the large bitmap.

18.151.4.4 ScalableImage::ScalingAlgorithm getScalingMode ( ) [virtual]

Gets the scaling algorithm of the ScalableImage.

Returns

the scaling algorithm used.

18.151.4.5 Bitmap getSmallBitmap ( ) const [inline]

Gets the small bitmap.

Returns

the small bitmap.

18.151.4.6 uint16_t getType ( ) const [inline], [virtual]

For GUI testing only. Returns type of this drawable.

Returns

TYPE_ZOOMANIMATIONIMAGE.

Reimplemented from Container.

18.151.4.7 void handleTickEvent ( ) [virtual]

The tick handler.

Reimplemented from Drawable.

18.151.4.8 bool isRunning ( ) const [virtual]

Is there currently an animation running.

Returns

true if there is an animation running.

18.151.4.9 void setAlpha (uint8_t *alpha*) [virtual]

Sets the alpha channel for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

18.151.4.10 void setAnimationDelay (uint16_t *delay*) [virtual]

Sets a delay on animations done by the ZoomAnimationImage. Defaults to 0.

Parameters

delay	The delay in ticks.

18.151.4.11 void setAnimationEndedCallback (touchgfx::GenericCallback< const ZoomAnimationImage & > & *callback*) [inline]

Associates an action to be performed when the animation ends.

Parameters

callback	The callback to be executed. The callback will be given a reference to the ZoomAnimationImage.

See also

GenericCallback

18.151.4.12 void setBitmaps (const Bitmap & *smallBitmap,* const Bitmap & *largeBitmap*)

Initializes the bitmap of the image to be used. The bitmaps should represent the same image in the two needed static resolutions. Note that it is possible to scale the image beyond the sizes of these bitmaps.

Parameters

smallBitmap	The image in the smallest resolution.
largeBitmap	The image in the largest resolution.

18.151.4.13 void setCurrentState (States *state*) [protected],[virtual]

Sets the current animation state.

Parameters

state	The new state.

18.151.4.14 void setDimension (int16_t *width,* int16_t *height*) [virtual]

Sets the width and height of the image. Chooses the optimal rendering method afterwards The image is automatically invalidated.

Parameters

width	The new width.
height	The new height.

18.151.4.15 void setHeight (int16_t *height*) [virtual]

Sets the height of the image. Chooses the optimal rendering method afterwards. The image is automatically invalidated.

Parameters

height	The new height.

Reimplemented from Drawable.

18.151.4.16 void setPosition (int16_t *x*, int16_t *y*, int16_t *width*, int16_t *height*) [virtual]

Sets the size and position of the image, relative to its parent. Chooses the optimal rendering method afterwards The image is automatically invalidated.

Parameters

x	The x coordinate of this Drawable.
y	The y coordinate of this Drawable.
width	The width of this Drawable.
height	The height of this Drawable.

Reimplemented from Drawable.

18.151.4.17 void setScalingMode (ScalableImage::ScalingAlgorithm *mode*) [virtual]

Sets the scaling algorithm of the ScalableImage.

Parameters

mode	The new mode.

18.151.4.18 void setWidth (int16_t *width*) [virtual]

Sets the width of the image. Chooses the optimal rendering method afterwards The image is automatically invalidated.

Parameters

width	The new width.

Reimplemented from Drawable.

18.151.4.19 void startZoomAndMoveAnimation (int16_t *endX*, int16_t *endY*, int16_t *endWidth*, int16_t *endHeight*, uint16_t *duration*, ZoomMode *zoomMode* = FIXED_LEFT_AND_TOP, EasingEquation *xProgressionEquation* = &touchgfx::EasingEquations::linearEaseNone, EasingEquation *yProgressionEquation* = &touchgfx::EasingEquations::linearEaseNone, EasingEquation *widthProgressionEquation* = &touchgfx::EasingEquations::linearEaseNone, EasingEquation *heightProgressionEquation* = &touchgfx::EasingEquations::linearEaseNone)

Setup and starts the zoom and move animation. At end of the animation the image will have been resized to the endWidth and endHeight and have moved from its original position to the endX and endY. Please note that the ZoomMode might influence the actual end position since the zoom transformation might change the X and Y of the image. The ZoomMode FIXED_LEFT_AND_TOP ensures that the endX and endY will be the actual end position.

The development of the width, height, X and Y during the animation is described by the supplied EasingEquations. The container is registered as a TimerWidget. Unregistering is handled automatically when the animation has finished.

Parameters

endX	The X position of the image at animation end. Relative to the container or view that holds the ZoomAnimationImage.
endY	The Y position of the image at animation end. Relative to the container or view that holds the ZoomAnimationImage.
endWidth	The width of the image at animation end.
endHeight	The height of the image at animation end.
duration	The duration of the animation measured in ticks.
zoomMode	The zoom mode that will be used during the animation. Default = FIXED_LEFT_AND_TOP.
xProgression↩ Equation	The equation that describes the development of the X position during the animation. Default = EasingEquations::linearEaseNone.
yProgression↩ Equation	The equation that describes the development of the Y position during the animation. Default = EasingEquations::linearEaseNone.
width↩ Progression↩ Equation	The equation that describes the development of the width during the animation. Default = EasingEquations::linearEaseNone.
height↩ Progression↩ Equation	The equation that describes the development of the height during the animation. Default = EasingEquations::linearEaseNone.

18.151.4.20 void startZoomAnimation (int16_t *endWidth*, int16_t *endHeight*, uint16_t *duration*, ZoomMode *zoomMode* = FIXED_LEFT_AND_TOP, touchgfx::EasingEquation *widthProgressionEquation* = &touchgfx::EasingEquations::linearEaseNone, EasingEquation *heightProgressionEquation* = &touchgfx::EasingEquations::linearEaseNone)

Setup and starts the zoom animation. At end of the animation the image will have been resized to the end↩ Width and endHeight. The development of the width and height during the animation is described by the supplied EasingEquations. The container is registered as a TimerWidget. Unregistering is handled automatically when the animation has finished.

Note that the animation follows the specified ZoomMode so the X and Y of the image might change during animation.

Parameters

endWidth	The width of the image at animation end.
endHeight	The height of the image at animation end.
duration	The duration of the animation measured in ticks.
zoomMode	The zoom mode that will be used during the animation. Default = FIXED_LEFT_AND_TOP.
width↩ Progression↩ Equation	The equation that describes the development of the width during the animation. Default = EasingEquations::linearEaseNone.
height↩ Progression↩ Equation	The equation that describes the development of the height during the animation. Default = EasingEquations::linearEaseNone.

18.151.4.21 void updateRenderingMethod ( ) [protected],[virtual]

Chooses the optimal rendering of the image given the current width and height.

18.151.4.22 void updateZoomAnimationDeltaXY ( ) [protected],[virtual]

Calculates the change in X and Y caused by the zoom animation given the current ZoomMode.

Index

AA_2MASK
touchgfx::Rasterizer, 488
AA_2NUM
touchgfx::Rasterizer, 488
AA_MASK
touchgfx::Rasterizer, 488
AA_NUM
touchgfx::Rasterizer, 488
AA_SHIFT
touchgfx::Rasterizer, 488
ANIMATE_ZOOM
touchgfx::ZoomAnimationImage, 596
ANIMATE_ZOOM_AND_MOVE
touchgfx::ZoomAnimationImage, 596
ARGB8888
touchgfx::Bitmap, 221
abs
touchgfx, 181
Alignment
touchgfx, 179
Application, 213

BLIT_OP_COPY
touchgfx, 179
BLIT_OP_COPY_A4
touchgfx, 179
BLIT_OP_COPY_A8
touchgfx, 179
BLIT_OP_COPY_ARGB8888
touchgfx, 179
BLIT_OP_COPY_ARGB8888_WITH_ALPHA
touchgfx, 179
BLIT_OP_COPY_WITH_ALPHA
touchgfx, 179
BLIT_OP_COPY_WITH_TRANSPARENT_PIXELS
touchgfx, 179
BLIT_OP_FILL
touchgfx, 179
BLIT_OP_FILL_WITH_ALPHA
touchgfx, 179
BUTT_CAP_ENDING
touchgfx::Line, 420
BW
touchgfx::Bitmap, 221
BW_RLE
touchgfx::Bitmap, 221
Bitmap, 219
Box, 228
Button, 231
Buttons, 234

CANCEL
touchgfx::ClickEvent, 264
Canvas, 246
Cell, 256
Circle, 257
Color, 268
colortype, 270
Container, 273

Direction
touchgfx, 179
Draggable< T >, 297
Drawable, 298

EAST
touchgfx, 179
EVENT_CLICK
touchgfx::Event, 328
EVENT_DRAG
touchgfx::Event, 328
EVENT_GESTURE
touchgfx::Event, 328
Edge, 326
Event, 328

FB_PRIMARY
touchgfx, 180
FB_SECONDARY
touchgfx, 180
FB_TERTIARY
touchgfx, 180
FILL_EVEN_ODD
touchgfx::Rasterizer, 488
FILL_NON_ZERO
touchgfx::Rasterizer, 488
FRAME_RATE
touchgfx::GPIO, 349
Font, 331

GLYPH_DATA_ADVANCE_BIT8
touchgfx, 180
GLYPH_DATA_FORMAT_A4
touchgfx, 180
GLYPH_DATA_HEIGHT_BIT8
touchgfx, 180
GLYPH_DATA_TOP_BIT8
touchgfx, 180
GLYPH_DATA_TOP_BIT9
touchgfx, 180
GLYPH_DATA_WIDTH_BIT8

touchgfx, 180
GRADIENT_HORIZONTAL
touchgfx, 180
GRADIENT_VERTICAL
touchgfx, 180
Gestures, 344
Gradient
touchgfx, 180
Gradients, 350

Image, 379

Keyboard, 387

Line, 419

MCU_ACTIVE
touchgfx::GPIO, 349
Matrix4x4, 430
memset
touchgfx, 184

NO_ANIMATION
touchgfx::ZoomAnimationImage, 596
NORTH
touchgfx, 179

ORIENTATION_LANDSCAPE
touchgfx, 180
ORIENTATION_PORTRAIT
touchgfx, 180
operator∗
touchgfx, 184
Outline, 445

POLY_BASE_MASK
touchgfx::Rasterizer, 488
POLY_BASE_SHIFT
touchgfx::Rasterizer, 488
POLY_BASE_SIZE
touchgfx::Rasterizer, 488
PRESSED
touchgfx::ClickEvent, 264
Pair< T1, T2 >, 458
Point, 464
Point4, 466
Presenter, 468

Quadruple, 476

RELEASED
touchgfx::ClickEvent, 264
RENDER_TIME
touchgfx::GPIO, 349
RGB565
touchgfx::Bitmap, 221
RGB888
touchgfx::Bitmap, 221
Rasterizer, 487
Rect, 490
Renderer, 493
rotate0
touchgfx, 180
rotate90
touchgfx, 180

SOUTH
touchgfx, 179
SQUARE_CAP_ENDING
touchgfx::Line, 420
SWIPE_HORIZONTAL
touchgfx::GestureEvent, 343
SWIPE_VERTICAL
touchgfx::GestureEvent, 343
Scanline, 503
Screen, 506
Slider, 523
Snapper< T >, 532

TEXT_ROTATE_0
touchgfx, 181
TEXT_ROTATE_180
touchgfx, 181
TEXT_ROTATE_270
touchgfx, 181
TEXT_ROTATE_90
touchgfx, 181
Texts, 550
touchgfx, 171
abs, 181
Alignment, 179
BLIT_OP_COPY, 179
BLIT_OP_COPY_A4, 179
BLIT_OP_COPY_A8, 179
BLIT_OP_COPY_ARGB8888, 179
BLIT_OP_COPY_ARGB8888_WITH_ALPHA, 179
BLIT_OP_COPY_WITH_ALPHA, 179
BLIT_OP_COPY_WITH_TRANSPARENT_PIXE↩
LS, 179
BLIT_OP_FILL, 179
BLIT_OP_FILL_WITH_ALPHA, 179
Direction, 179
EAST, 179
FB_PRIMARY, 180
FB_SECONDARY, 180
FB_TERTIARY, 180
GLYPH_DATA_ADVANCE_BIT8, 180
GLYPH_DATA_FORMAT_A4, 180
GLYPH_DATA_HEIGHT_BIT8, 180
GLYPH_DATA_TOP_BIT8, 180
GLYPH_DATA_TOP_BIT9, 180
GLYPH_DATA_WIDTH_BIT8, 180
GRADIENT_HORIZONTAL, 180
GRADIENT_VERTICAL, 180
Gradient, 180
memset, 184
NORTH, 179
ORIENTATION_LANDSCAPE, 180
ORIENTATION_PORTRAIT, 180

operator∗, 184
rotate0, 180
rotate90, 180
SOUTH, 179
TEXT_ROTATE_0, 181
TEXT_ROTATE_180, 181
TEXT_ROTATE_270, 181
TEXT_ROTATE_90, 181
WEST, 179
touchgfx::Bitmap
ARGB8888, 221
BW, 221
BW_RLE, 221
RGB565, 221
RGB888, 221
touchgfx::ClickEvent
CANCEL, 264
PRESSED, 264
RELEASED, 264
touchgfx::Event
EVENT_CLICK, 328
EVENT_DRAG, 328
EVENT_GESTURE, 328
touchgfx::GPIO
FRAME_RATE, 349
MCU_ACTIVE, 349
RENDER_TIME, 349
touchgfx::GestureEvent
SWIPE_HORIZONTAL, 343
SWIPE_VERTICAL, 343
touchgfx::Line
BUTT_CAP_ENDING, 420
SQUARE_CAP_ENDING, 420
touchgfx::Rasterizer
AA_2MASK, 488
AA_2NUM, 488
AA_MASK, 488
AA_NUM, 488
AA_SHIFT, 488
FILL_EVEN_ODD, 488
FILL_NON_ZERO, 488
POLY_BASE_MASK, 488
POLY_BASE_SHIFT, 488
POLY_BASE_SIZE, 488
touchgfx::ZoomAnimationImage
ANIMATE_ZOOM, 596
ANIMATE_ZOOM_AND_MOVE, 596
NO_ANIMATION, 596
Transition, 572

Unicode, 578

Vector< T, capacity >, 585
Vector4, 589
View< T >, 590

WEST
touchgfx, 179
Widget, 591