

# TouchGFX v4.10.0 User's Manual

(C)opyright STMicroelectronics 2014-2018

[www.st.com](http://www.st.com)  
[www.touchgfx.com](http://www.touchgfx.com)



# Contents

<b>1</b>	<b>Changelog</b>	<b>1</b>
<b>2</b>	<b>Globals</b>	<b>17</b>
<b>3</b>	<b>Namespace Index</b>	<b>19</b>
3.1	Namespace List . . . . .	19
<b>4</b>	<b>Hierarchical Index</b>	<b>21</b>
4.1	Class Hierarchy . . . . .	21
<b>5</b>	<b>Class Index</b>	<b>27</b>
5.1	Class List . . . . .	27
<b>6</b>	<b>Namespace Documentation</b>	<b>35</b>
6.1	touchgfx Namespace Reference . . . . .	35
6.1.1	Typedef Documentation . . . . .	47
6.1.1.1	Alignment . . . . .	47
6.1.1.2	EasingEquation . . . . .	47
6.1.1.3	RenderingVariant . . . . .	47
6.1.1.4	TextDirection . . . . .	47
6.1.2	Enumeration Type Documentation . . . . .	47
6.1.2.1	BlitOperations . . . . .	47
6.1.2.2	Direction . . . . .	48
6.1.2.3	DisplayOrientation . . . . .	48
6.1.2.4	DisplayRotation . . . . .	48
6.1.2.5	FrameBuffer . . . . .	49
6.1.2.6	GlyphFlags . . . . .	49
6.1.2.7	Gradient . . . . .	49
6.1.2.8	TextRotation . . . . .	50
6.1.2.9	WideTextAction . . . . .	50
6.1.3	Function Documentation . . . . .	50
6.1.3.1	abs() . . . . .	50
6.1.3.2	ceil28_4() . . . . .	51

6.1.3.3	clz()	51
6.1.3.4	finalizeTransition()	51
6.1.3.5	fixed28_4Mul()	52
6.1.3.6	fixed28_4ToFloat()	52
6.1.3.7	floatToFixed16_16()	52
6.1.3.8	floatToFixed28_4()	52
6.1.3.9	floorDivMod()	53
6.1.3.10	gcd()	53
6.1.3.11	hw_init()	53
6.1.3.12	LCD2getPixel() [1/2]	54
6.1.3.13	LCD2getPixel() [2/2]	54
6.1.3.14	LCD2setPixel() [1/2]	54
6.1.3.15	LCD2setPixel() [2/2]	55
6.1.3.16	LCD2shiftVal()	55
6.1.3.17	LCD4getPixel() [1/2]	55
6.1.3.18	LCD4getPixel() [2/2]	55
6.1.3.19	LCD4setPixel() [1/2]	56
6.1.3.20	LCD4setPixel() [2/2]	56
6.1.3.21	lookupBilinearRenderVariant()	56
6.1.3.22	lookupNearestNeighborRenderVariant()	57
6.1.3.23	makeTransition()	57
6.1.3.24	memset()	58
6.1.3.25	muldiv()	58
6.1.3.26	operator*() [1/2]	58
6.1.3.27	operator*() [2/2]	59
6.1.3.28	prepareTransition()	59
6.1.3.29	touchgfx_generic_init()	59
6.1.3.30	touchgfx_init()	60
6.1.4	Variable Documentation	60
6.1.4.1	TYPED_TEXT_INVALID	61
<b>7</b>	<b>Class Documentation</b>	<b>63</b>
7.1	AbstractButton Class Reference	63
7.1.1	Detailed Description	63
7.1.2	Constructor & Destructor Documentation	64
7.1.2.1	AbstractButton()	64
7.1.3	Member Function Documentation	64
7.1.3.1	getPressedState()	64
7.1.3.2	getType()	64
7.1.3.3	handleClickEvent()	64

7.1.3.4	setAction()	65
7.2	AbstractButtonContainer Class Reference	65
7.2.1	Detailed Description	66
7.2.2	Member Function Documentation	66
7.2.2.1	getAlpha()	66
7.2.2.2	getPressed()	66
7.2.2.3	setAction()	66
7.2.2.4	setAlpha()	67
7.2.2.5	setPressed()	67
7.3	AbstractClock Class Reference	67
7.3.1	Constructor & Destructor Documentation	68
7.3.1.1	AbstractClock()	68
7.3.1.2	~AbstractClock()	68
7.3.2	Member Function Documentation	68
7.3.2.1	getCurrentHour()	68
7.3.2.2	getCurrentMinute()	68
7.3.2.3	getCurrentSecond()	69
7.3.2.4	setTime12Hour()	69
7.3.2.5	setTime24Hour()	69
7.3.2.6	updateClock()	69
7.4	AbstractDirectionProgress Class Reference	70
7.4.1	Detailed Description	70
7.4.2	Member Enumeration Documentation	70
7.4.2.1	DirectionType	70
7.4.3	Constructor & Destructor Documentation	70
7.4.3.1	AbstractDirectionProgress()	71
7.4.3.2	~AbstractDirectionProgress()	71
7.4.4	Member Function Documentation	71
7.4.4.1	getDirection()	71
7.4.4.2	setDirection()	71
7.5	AbstractPainter Class Reference	71
7.5.1	Detailed Description	72
7.5.2	Constructor & Destructor Documentation	72
7.5.2.1	AbstractPainter()	72
7.5.2.2	~AbstractPainter()	72
7.5.3	Member Function Documentation	72
7.5.3.1	render()	72
7.5.3.2	setOffset()	74
7.5.3.3	setWidthAlpha()	74
7.6	AbstractPainterBW Class Reference	74

7.6.1	Detailed Description	75
7.6.2	Member Function Documentation	75
7.6.2.1	render()	75
7.6.2.2	renderInit()	76
7.6.2.3	renderNext()	76
7.7	AbstractPainterGRAY2 Class Reference	76
7.7.1	Detailed Description	77
7.7.2	Member Function Documentation	77
7.7.2.1	render()	77
7.7.2.2	renderInit()	77
7.7.2.3	renderNext()	78
7.7.2.4	renderPixel()	78
7.8	AbstractPainterGRAY4 Class Reference	78
7.8.1	Detailed Description	79
7.8.2	Member Function Documentation	79
7.8.2.1	render()	79
7.8.2.2	renderInit()	80
7.8.2.3	renderNext()	80
7.8.2.4	renderPixel()	80
7.9	AbstractPainterRGB565 Class Reference	81
7.9.1	Detailed Description	81
7.9.2	Member Function Documentation	82
7.9.2.1	mixColors() [1/2]	82
7.9.2.2	mixColors() [2/2]	82
7.9.2.3	render()	83
7.9.2.4	renderInit()	83
7.9.2.5	renderNext()	83
7.9.2.6	renderPixel()	84
7.10	AbstractPainterRGB888 Class Reference	84
7.10.1	Detailed Description	85
7.10.2	Member Function Documentation	85
7.10.2.1	render()	85
7.10.2.2	renderInit()	85
7.10.2.3	renderNext()	86
7.10.2.4	renderPixel()	86
7.11	AbstractPartition Class Reference	86
7.11.1	Detailed Description	87
7.11.2	Constructor & Destructor Documentation	87
7.11.2.1	~AbstractPartition()	88
7.11.2.2	AbstractPartition()	88

7.11.3	Member Function Documentation	88
7.11.3.1	allocate() [1/2]	88
7.11.3.2	allocate() [2/2]	88
7.11.3.3	allocateAt() [1/2]	89
7.11.3.4	allocateAt() [2/2]	89
7.11.3.5	at() [1/2]	90
7.11.3.6	at() [2/2]	90
7.11.3.7	capacity()	90
7.11.3.8	clear()	91
7.11.3.9	dec()	91
7.11.3.10	element() [1/2]	91
7.11.3.11	element() [2/2]	91
7.11.3.12	element_size()	91
7.11.3.13	find()	92
7.11.3.14	getAllocationCount()	92
7.11.3.15	indexOf()	92
7.12	AbstractProgressIndicator Class Reference	93
7.12.1	Detailed Description	94
7.12.2	Constructor & Destructor Documentation	94
7.12.2.1	AbstractProgressIndicator()	94
7.12.2.2	~AbstractProgressIndicator()	94
7.12.3	Member Function Documentation	94
7.12.3.1	getProgress()	95
7.12.3.2	getProgressIndicatorHeight()	95
7.12.3.3	getProgressIndicatorWidth()	95
7.12.3.4	getProgressIndicatorX()	95
7.12.3.5	getProgressIndicatorY()	96
7.12.3.6	getRange() [1/3]	96
7.12.3.7	getRange() [2/3]	96
7.12.3.8	getRange() [3/3]	96
7.12.3.9	getValue()	97
7.12.3.10	setBackground()	97
7.12.3.11	setProgressIndicatorPosition()	97
7.12.3.12	setRange()	98
7.12.3.13	setValue()	98
7.13	AbstractShape Class Reference	98
7.13.1	Detailed Description	100
7.13.2	Constructor & Destructor Documentation	100
7.13.2.1	AbstractShape()	100
7.13.2.2	~AbstractShape()	100

7.13.3	Member Function Documentation	100
7.13.3.1	drawCanvasWidget()	100
7.13.3.2	getAngle()	101
7.13.3.3	getCacheX()	101
7.13.3.4	getCacheY()	101
7.13.3.5	getCornerX()	102
7.13.3.6	getCornerY()	102
7.13.3.7	getMinimalRect()	102
7.13.3.8	getNumPoints()	102
7.13.3.9	getOrigin()	103
7.13.3.10	getScale()	103
7.13.3.11	moveOrigin()	103
7.13.3.12	setAngle()	104
7.13.3.13	setCache()	104
7.13.3.14	setCorner()	105
7.13.3.15	setOrigin()	105
7.13.3.16	setScale() [1/2]	106
7.13.3.17	setScale() [2/2]	106
7.13.3.18	setShape()	107
7.13.3.19	updateAbstractShapeCache()	107
7.13.3.20	updateAngle()	107
7.14	AnalogClock Class Reference	108
7.14.1	Detailed Description	110
7.14.2	Member Function Documentation	110
7.14.2.1	animationEnabled()	110
7.14.2.2	convertHandValueToAngle()	110
7.14.2.3	getAnimationDuration()	110
7.14.2.4	getHourHandMinuteCorrection()	111
7.14.2.5	getMinuteHandSecondCorrection()	111
7.14.2.6	initializeTime12Hour()	111
7.14.2.7	initializeTime24Hour()	112
7.14.2.8	setAnimation()	112
7.14.2.9	setBackground() [1/2]	112
7.14.2.10	setBackground() [2/2]	112
7.14.2.11	setHourHandMinuteCorrection()	113
7.14.2.12	setMinuteHandSecondCorrection()	113
7.14.2.13	setRotationCenter()	113
7.14.2.14	setupHand()	114
7.14.2.15	setupHourHand()	114
7.14.2.16	setupMinuteHand()	115



7.14.2.17	setupSecondHand()	115
7.14.2.18	updateClock()	115
7.15	AnimatedImage Class Reference	115
7.15.1	Detailed Description	117
7.15.2	Constructor & Destructor Documentation	117
7.15.2.1	AnimatedImage() [1/2]	117
7.15.2.2	AnimatedImage() [2/2]	117
7.15.3	Member Function Documentation	117
7.15.3.1	getType()	118
7.15.3.2	handleTickEvent()	118
7.15.3.3	isAnimatedImageRunning()	118
7.15.3.4	isReverse()	118
7.15.3.5	isRunning()	118
7.15.3.6	pauseAnimation()	119
7.15.3.7	setBitmap()	119
7.15.3.8	setBitmaps()	119
7.15.3.9	setDoneAction()	119
7.15.3.10	setUpdateTicksInterval()	119
7.15.3.11	startAnimation()	120
7.15.3.12	stopAnimation()	120
7.16	AnimatedImageButtonStyle< T > Class Template Reference	120
7.16.1	Detailed Description	121
7.16.2	Member Function Documentation	121
7.16.2.1	setBitmaps()	121
7.16.2.2	setBitmapXY()	121
7.16.2.3	setUpdateTicksInterval()	122
7.17	AnimationTextureMapper::AnimationSetting Struct Reference	122
7.18	AnimationTextureMapper Class Reference	122
7.18.1	Detailed Description	124
7.18.2	Member Enumeration Documentation	124
7.18.2.1	AnimationParameter	124
7.18.3	Constructor & Destructor Documentation	124
7.18.3.1	AnimationTextureMapper()	124
7.18.3.2	~AnimationTextureMapper()	124
7.18.4	Member Function Documentation	124
7.18.4.1	cancelAnimationTextureMapperAnimation()	124
7.18.4.2	getAnimationStep()	125
7.18.4.3	handleTickEvent()	125
7.18.4.4	isTextureMapperAnimationRunning()	125
7.18.4.5	setTextureMapperAnimationEndedAction()	125

7.18.4.6	setTextureMapperAnimationStepAction()	125
7.18.4.7	setupAnimation()	126
7.18.4.8	startAnimation()	126
7.19	Application Class Reference	126
7.19.1	Detailed Description	128
7.19.2	Constructor & Destructor Documentation	128
7.19.2.1	Application()	128
7.19.3	Member Function Documentation	128
7.19.3.1	appSwitchScreen()	129
7.19.3.2	cacheDrawOperations()	129
7.19.3.3	clearAllTimerWidgets()	129
7.19.3.4	draw() [1/2]	129
7.19.3.5	draw() [2/2]	129
7.19.3.6	getCurrentScreen()	130
7.19.3.7	getInstance()	130
7.19.3.8	getNumberOfRegisteredTimerWidgets()	130
7.19.3.9	getTimerWidgetCountForDrawable()	130
7.19.3.10	handleClickEvent()	131
7.19.3.11	handleDragEvent()	131
7.19.3.12	handleGestureEvent()	131
7.19.3.13	handleKeyEvent()	132
7.19.3.14	handlePendingScreenTransition()	132
7.19.3.15	handleTickEvent()	132
7.19.3.16	invalidateArea()	132
7.19.3.17	registerTimerWidget()	132
7.19.3.18	switchScreen()	133
7.19.3.19	unregisterTimerWidget()	133
7.19.4	Member Data Documentation	133
7.19.4.1	instance	133
7.19.4.2	MAX_TIMER_WIDGETS	134
7.20	Bitmap Class Reference	134
7.20.1	Detailed Description	135
7.20.2	Member Enumeration Documentation	135
7.20.2.1	BitmapFormat	136
7.20.3	Constructor & Destructor Documentation	136
7.20.3.1	Bitmap()	136
7.20.4	Member Function Documentation	136
7.20.4.1	cache()	136
7.20.4.2	cacheAll()	137
7.20.4.3	cacheGetAddress()	137

7.20.4.4	cacheIsCached()	137
7.20.4.5	cacheRemoveBitmap()	138
7.20.4.6	cacheReplaceBitmap()	138
7.20.4.7	clearCache()	139
7.20.4.8	dynamicBitmapCreate()	139
7.20.4.9	dynamicBitmapDelete()	139
7.20.4.10	dynamicBitmapGetAddress()	140
7.20.4.11	dynamicBitmapSetSolidRect()	140
7.20.4.12	getAlphaData()	140
7.20.4.13	getData()	141
7.20.4.14	getFormat()	141
7.20.4.15	getHeight()	141
7.20.4.16	getId()	141
7.20.4.17	getRect()	142
7.20.4.18	getSolidRect()	142
7.20.4.19	getWidth()	142
7.20.4.20	hasTransparentPixels()	142
7.20.4.21	isAlphaPerPixel()	142
7.20.4.22	operator!=(())	143
7.20.4.23	operator==(())	143
7.20.4.24	registerBitmapDatabase()	143
7.20.4.25	setCache()	144
7.21	Bitmap::BitmapData Struct Reference	144
7.21.1	Detailed Description	144
7.22	BlitOp Struct Reference	144
7.22.1	Detailed Description	145
7.22.2	Member Data Documentation	145
7.22.2.1	operation	145
7.23	Box Class Reference	145
7.23.1	Detailed Description	146
7.23.2	Constructor & Destructor Documentation	146
7.23.2.1	Box() [1/2]	146
7.23.2.2	Box() [2/2]	146
7.23.2.3	~Box()	147
7.23.3	Member Function Documentation	147
7.23.3.1	draw()	147
7.23.3.2	forceReportAsSolid()	147
7.23.3.3	getAlpha()	148
7.23.3.4	getColor()	148
7.23.3.5	getSolidRect()	148

7.23.3.6	getType()	148
7.23.3.7	setAlpha()	148
7.23.3.8	setColor()	149
7.23.4	Member Data Documentation	149
7.23.4.1	reportAsSolid	149
7.24	BoxProgress Class Reference	149
7.24.1	Detailed Description	150
7.24.2	Constructor & Destructor Documentation	150
7.24.2.1	BoxProgress()	150
7.24.2.2	~BoxProgress()	150
7.24.3	Member Function Documentation	150
7.24.3.1	getAlpha()	150
7.24.3.2	getColor()	151
7.24.3.3	setAlpha()	151
7.24.3.4	setColor()	151
7.24.3.5	setProgressIndicatorPosition()	151
7.24.3.6	setValue()	152
7.25	BoxWithBorder Class Reference	152
7.25.1	Constructor & Destructor Documentation	153
7.25.1.1	BoxWithBorder()	153
7.25.2	Member Function Documentation	154
7.25.2.1	draw()	154
7.25.2.2	getAlpha()	154
7.25.2.3	getBorderColor()	154
7.25.2.4	getBorderSize()	154
7.25.2.5	getColor()	154
7.25.2.6	getSolidRect()	155
7.25.2.7	getType()	155
7.25.2.8	setAlpha()	155
7.25.2.9	setBorderColor()	155
7.25.2.10	setBorderSize()	155
7.25.2.11	setColor()	156
7.26	BoxWithBorderButtonStyle< T > Class Template Reference	156
7.26.1	Detailed Description	157
7.26.2	Member Function Documentation	157
7.26.2.1	setBorderSize()	157
7.26.2.2	setBoxWithBorderColors()	157
7.26.2.3	setBoxWithBorderHeight()	158
7.26.2.4	setBoxWithBorderPosition()	158
7.26.2.5	setBoxWithBorderWidth()	158

7.27 Button Class Reference . . . . .	159
7.27.1 Detailed Description . . . . .	159
7.27.2 Constructor & Destructor Documentation . . . . .	159
7.27.2.1 Button() . . . . .	160
7.27.2.2 ~Button() . . . . .	160
7.27.3 Member Function Documentation . . . . .	160
7.27.3.1 draw() . . . . .	160
7.27.3.2 getAlpha() . . . . .	160
7.27.3.3 getCurrentlyDisplayedBitmap() . . . . .	160
7.27.3.4 getSolidRect() . . . . .	161
7.27.3.5 getType() . . . . .	161
7.27.3.6 setAlpha() . . . . .	161
7.27.3.7 setBitmaps() . . . . .	161
7.28 ButtonController Class Reference . . . . .	162
7.28.1 Detailed Description . . . . .	162
7.28.2 Constructor & Destructor Documentation . . . . .	162
7.28.2.1 ~ButtonController() . . . . .	162
7.28.3 Member Function Documentation . . . . .	162
7.28.3.1 init() . . . . .	162
7.28.3.2 reset() . . . . .	162
7.28.3.3 sample() . . . . .	162
7.29 Buttons Class Reference . . . . .	163
7.29.1 Member Function Documentation . . . . .	163
7.29.1.1 init() . . . . .	163
7.29.1.2 sample() . . . . .	163
7.30 ButtonWithIcon Class Reference . . . . .	163
7.30.1 Detailed Description . . . . .	164
7.30.2 Member Function Documentation . . . . .	164
7.30.2.1 draw() . . . . .	164
7.30.2.2 getCurrentlyDisplayedIcon() . . . . .	165
7.30.2.3 getIconX() . . . . .	165
7.30.2.4 getIconY() . . . . .	165
7.30.2.5 getType() . . . . .	165
7.30.2.6 setBitmaps() . . . . .	166
7.30.2.7 setIconX() . . . . .	166
7.30.2.8 setIconXY() . . . . .	166
7.30.2.9 setIconY() . . . . .	167
7.31 ButtonWithLabel Class Reference . . . . .	167
7.31.1 Detailed Description . . . . .	168
7.31.2 Constructor & Destructor Documentation . . . . .	168

7.31.2.1	ButtonWithLabel()	168
7.31.3	Member Function Documentation	168
7.31.3.1	draw()	168
7.31.3.2	getLabelRotation()	169
7.31.3.3	getLabelText()	169
7.31.3.4	getSolidRect()	169
7.31.3.5	getType()	169
7.31.3.6	setLabelColor()	169
7.31.3.7	setLabelColorPressed()	170
7.31.3.8	setLabelRotation()	170
7.31.3.9	setLabelText()	170
7.31.3.10	updateTextPosition()	170
7.32	Bitmap::CacheTableEntry Struct Reference	171
7.32.1	Detailed Description	171
7.33	Callback< dest_type, T1, T2, T3 > Struct Template Reference	171
7.33.1	Detailed Description	171
7.33.2	Constructor & Destructor Documentation	172
7.33.2.1	Callback() [1/2]	172
7.33.2.2	Callback() [2/2]	172
7.33.3	Member Function Documentation	172
7.33.3.1	execute()	172
7.33.3.2	isValid()	173
7.34	Callback< dest_type, T1, T2, void > Struct Template Reference	173
7.34.1	Detailed Description	173
7.34.2	Constructor & Destructor Documentation	174
7.34.2.1	Callback() [1/2]	174
7.34.2.2	Callback() [2/2]	174
7.34.3	Member Function Documentation	174
7.34.3.1	execute()	174
7.34.3.2	isValid()	175
7.35	Callback< dest_type, T1, void, void > Struct Template Reference	175
7.35.1	Detailed Description	175
7.35.2	Constructor & Destructor Documentation	176
7.35.2.1	Callback() [1/2]	176
7.35.2.2	Callback() [2/2]	176
7.35.3	Member Function Documentation	176
7.35.3.1	execute()	176
7.35.3.2	isValid()	176
7.36	Callback< dest_type, void, void, void > Struct Template Reference	177
7.36.1	Detailed Description	177

7.36.2	Constructor & Destructor Documentation	177
7.36.2.1	Callback() [1/2]	178
7.36.2.2	Callback() [2/2]	178
7.36.3	Member Function Documentation	178
7.36.3.1	execute()	178
7.36.3.2	isValid()	178
7.37	Keyboard::CallbackArea Struct Reference	178
7.38	Canvas Class Reference	179
7.38.1	Detailed Description	179
7.38.2	Constructor & Destructor Documentation	179
7.38.2.1	Canvas()	179
7.38.2.2	~Canvas()	180
7.38.3	Member Function Documentation	180
7.38.3.1	lineTo() [1/2]	180
7.38.3.2	lineTo() [2/2]	180
7.38.3.3	moveTo() [1/2]	181
7.38.3.4	moveTo() [2/2]	181
7.38.3.5	render()	181
7.39	CanvasWidget Class Reference	182
7.39.1	Detailed Description	182
7.39.2	Constructor & Destructor Documentation	182
7.39.2.1	CanvasWidget()	183
7.39.2.2	~CanvasWidget()	183
7.39.3	Member Function Documentation	183
7.39.3.1	draw()	183
7.39.3.2	drawCanvasWidget()	183
7.39.3.3	getAlpha()	184
7.39.3.4	getMinimalRect()	184
7.39.3.5	getPainter()	184
7.39.3.6	getSolidRect()	184
7.39.3.7	invalidate()	185
7.39.3.8	setAlpha()	185
7.39.3.9	setPainter()	185
7.40	CanvasWidgetRenderer Class Reference	186
7.40.1	Detailed Description	186
7.40.2	Member Function Documentation	187
7.40.2.1	getMissingBufferSize()	187
7.40.2.2	getOutlineBuffer()	187
7.40.2.3	getOutlineBufferSize()	187
7.40.2.4	getScanlineCounts()	187

7.40.2.5	<a href="#">getScanlineCovers()</a>	187
7.40.2.6	<a href="#">getScanlineStartIndices()</a>	188
7.40.2.7	<a href="#">getScanlineWidth()</a>	188
7.40.2.8	<a href="#">getUsedBufferSize()</a>	188
7.40.2.9	<a href="#">getWriteMemoryUsageReport()</a>	188
7.40.2.10	<a href="#">hasBuffer()</a>	189
7.40.2.11	<a href="#">numCellsMissing()</a>	189
7.40.2.12	<a href="#">numCellsUsed()</a>	189
7.40.2.13	<a href="#">setScanlineWidth()</a>	189
7.40.2.14	<a href="#">setupBuffer()</a>	190
7.40.2.15	<a href="#">setWriteMemoryUsageReport()</a>	190
7.41	<a href="#">Cell Struct Reference</a>	190
7.41.1	<a href="#">Detailed Description</a>	191
7.41.2	<a href="#">Member Function Documentation</a>	191
7.41.2.1	<a href="#">addCover()</a>	191
7.41.2.2	<a href="#">packedCoord()</a>	191
7.41.2.3	<a href="#">set()</a>	192
7.41.2.4	<a href="#">setCoord()</a>	192
7.41.2.5	<a href="#">setCover()</a>	192
7.42	<a href="#">Circle Class Reference</a>	193
7.42.1	<a href="#">Detailed Description</a>	194
7.42.2	<a href="#">Constructor &amp; Destructor Documentation</a>	194
7.42.2.1	<a href="#">Circle()</a>	194
7.42.3	<a href="#">Member Function Documentation</a>	194
7.42.3.1	<a href="#">drawCanvasWidget()</a>	195
7.42.3.2	<a href="#">getArc()</a>	195
7.42.3.3	<a href="#">getArcEnd()</a> [1/2]	195
7.42.3.4	<a href="#">getArcEnd()</a> [2/2]	196
7.42.3.5	<a href="#">getArcStart()</a> [1/2]	196
7.42.3.6	<a href="#">getArcStart()</a> [2/2]	196
7.42.3.7	<a href="#">getCapPrecision()</a>	197
7.42.3.8	<a href="#">getCenter()</a>	197
7.42.3.9	<a href="#">getLineWidth()</a>	197
7.42.3.10	<a href="#">getMinimalRect()</a> [1/3]	198
7.42.3.11	<a href="#">getMinimalRect()</a> [2/3]	198
7.42.3.12	<a href="#">getMinimalRect()</a> [3/3]	198
7.42.3.13	<a href="#">getPrecision()</a>	199
7.42.3.14	<a href="#">getRadius()</a>	199
7.42.3.15	<a href="#">setArc()</a> [1/2]	199
7.42.3.16	<a href="#">setArc()</a> [2/2]	200



7.42.3.17	setCapPrecision()	200
7.42.3.18	setCenter() [1/2]	201
7.42.3.19	setCenter() [2/2]	201
7.42.3.20	setCircle() [1/2]	202
7.42.3.21	setCircle() [2/2]	202
7.42.3.22	setLineWidth()	202
7.42.3.23	setPrecision()	203
7.42.3.24	setRadius()	203
7.42.3.25	updateArcEnd()	204
7.42.3.26	updateArcStart()	204
7.43	CircleProgress Class Reference	205
7.43.1	Detailed Description	206
7.43.2	Constructor & Destructor Documentation	206
7.43.2.1	CircleProgress()	206
7.43.2.2	~CircleProgress()	206
7.43.3	Member Function Documentation	206
7.43.3.1	getAlpha()	206
7.43.3.2	getCapPrecision()	207
7.43.3.3	getCenter()	207
7.43.3.4	getEndAngle()	207
7.43.3.5	getLineWidth()	207
7.43.3.6	getRadius()	208
7.43.3.7	getStartAngle()	208
7.43.3.8	setAlpha()	208
7.43.3.9	setCapPrecision()	208
7.43.3.10	setCenter()	209
7.43.3.11	setLineWidth()	209
7.43.3.12	setPainter()	209
7.43.3.13	setProgressIndicatorPosition()	210
7.43.3.14	setRadius()	210
7.43.3.15	setStartEndAngle()	210
7.43.3.16	setValue()	211
7.44	ClickButtonTrigger Class Reference	211
7.44.1	Detailed Description	211
7.44.2	Member Function Documentation	211
7.44.2.1	handleClickEvent()	211
7.45	ClickEvent Class Reference	212
7.45.1	Detailed Description	212
7.45.2	Member Enumeration Documentation	212
7.45.2.1	ClickEventType	213

7.45.3	Constructor & Destructor Documentation	214
7.45.3.1	ClickEvent()	214
7.45.3.2	~ClickEvent()	214
7.45.4	Member Function Documentation	214
7.45.4.1	getEventType()	214
7.45.4.2	getForce()	215
7.45.4.3	getType()	215
7.45.4.4	getX()	215
7.45.4.5	getY()	215
7.45.4.6	setType()	215
7.45.4.7	setX()	216
7.45.4.8	setY()	216
7.46	ClickListener< T > Class Template Reference	216
7.46.1	Detailed Description	217
7.46.2	Constructor & Destructor Documentation	217
7.46.2.1	ClickListener()	217
7.46.3	Member Function Documentation	217
7.46.3.1	handleClickEvent()	217
7.46.3.2	setClickAction()	217
7.47	Color Class Reference	218
7.47.1	Detailed Description	218
7.47.2	Member Function Documentation	218
7.47.2.1	getBlueColor()	218
7.47.2.2	getColorFrom24BitRGB()	218
7.47.2.3	getGreenColor()	219
7.47.2.4	getRedColor()	219
7.48	colortype Struct Reference	220
7.48.1	Detailed Description	220
7.48.2	Constructor & Destructor Documentation	220
7.48.2.1	colortype() [1/2]	220
7.48.2.2	colortype() [2/2]	220
7.48.3	Member Function Documentation	221
7.48.3.1	getColor32()	221
7.48.3.2	operator uint16_t()	221
7.49	ConstFont Class Reference	221
7.49.1	Detailed Description	222
7.49.2	Constructor & Destructor Documentation	222
7.49.2.1	ConstFont()	222
7.49.3	Member Function Documentation	223
7.49.3.1	find()	223

7.49.3.2	getGlyph()	223
7.49.3.3	getKerning()	224
7.49.3.4	getPixelData()	224
7.50	Container Class Reference	224
7.50.1	Detailed Description	225
7.50.2	Constructor & Destructor Documentation	226
7.50.2.1	Container()	226
7.50.2.2	~Container()	226
7.50.3	Member Function Documentation	226
7.50.3.1	add()	226
7.50.3.2	contains()	226
7.50.3.3	draw()	227
7.50.3.4	forEachChild()	227
7.50.3.5	getContainedArea()	227
7.50.3.6	getFirstChild()	227
7.50.3.7	getLastChild()	228
7.50.3.8	getSolidRect()	228
7.50.3.9	getType()	228
7.50.3.10	insert()	228
7.50.3.11	moveChildrenRelative()	229
7.50.3.12	remove()	229
7.50.3.13	removeAll()	229
7.50.3.14	setupDrawChain()	230
7.50.3.15	unlink()	230
7.51	CoverTransition< templateDirection > Class Template Reference	230
7.51.1	Detailed Description	231
7.51.2	Constructor & Destructor Documentation	231
7.51.2.1	CoverTransition()	231
7.51.2.2	~CoverTransition()	231
7.51.3	Member Function Documentation	232
7.51.3.1	handleTickEvent()	232
7.51.3.2	init()	232
7.51.3.3	initMoveDrawable()	232
7.51.3.4	tearDown()	232
7.51.3.5	tickMoveDrawable()	232
7.52	CWRUtil Struct Reference	233
7.52.1	Detailed Description	234
7.52.2	Member Function Documentation	234
7.52.2.1	angle() [1/4]	234
7.52.2.2	angle() [2/4]	234

7.52.2.3	<a href="#">angle()</a> [3/4]	235
7.52.2.4	<a href="#">angle()</a> [4/4]	235
7.52.2.5	<a href="#">arcsine()</a>	236
7.52.2.6	<a href="#">cosine()</a> [1/2]	236
7.52.2.7	<a href="#">cosine()</a> [2/2]	236
7.52.2.8	<a href="#">muldivQ5()</a>	237
7.52.2.9	<a href="#">mulQ5()</a>	237
7.52.2.10	<a href="#">sine()</a> [1/2]	237
7.52.2.11	<a href="#">sine()</a> [2/2]	238
7.52.2.12	<a href="#">sqrtQ10()</a>	238
7.52.2.13	<a href="#">toQ10()</a>	238
7.52.2.14	<a href="#">toQ5()</a>	239
7.53	<a href="#">DigitalClock Class Reference</a>	239
7.53.1	<a href="#">Detailed Description</a>	240
7.53.2	<a href="#">Member Enumeration Documentation</a>	241
7.53.2.1	<a href="#">DisplayMode</a>	241
7.53.3	<a href="#">Constructor &amp; Destructor Documentation</a>	241
7.53.3.1	<a href="#">DigitalClock()</a>	241
7.53.3.2	<a href="#">~DigitalClock()</a>	241
7.53.4	<a href="#">Member Function Documentation</a>	241
7.53.4.1	<a href="#">displayLeadingZeroForHourIndicator()</a>	241
7.53.4.2	<a href="#">getAlpha()</a>	242
7.53.4.3	<a href="#">getDisplayMode()</a>	242
7.53.4.4	<a href="#">getTextWidth()</a>	242
7.53.4.5	<a href="#">setAlpha()</a>	242
7.53.4.6	<a href="#">setBaselineY()</a>	242
7.53.4.7	<a href="#">setColor()</a>	243
7.53.4.8	<a href="#">setDisplayMode()</a>	243
7.53.4.9	<a href="#">setHeight()</a>	243
7.53.4.10	<a href="#">setTypedText()</a>	243
7.53.4.11	<a href="#">setWidth()</a>	244
7.53.4.12	<a href="#">updateClock()</a>	244
7.54	<a href="#">DisplayTransformation Class Reference</a>	244
7.54.1	<a href="#">Detailed Description</a>	245
7.54.2	<a href="#">Member Function Documentation</a>	245
7.54.2.1	<a href="#">transformDisplayToFrameBuffer()</a> [1/6]	245
7.54.2.2	<a href="#">transformDisplayToFrameBuffer()</a> [2/6]	245
7.54.2.3	<a href="#">transformDisplayToFrameBuffer()</a> [3/6]	245
7.54.2.4	<a href="#">transformDisplayToFrameBuffer()</a> [4/6]	246
7.54.2.5	<a href="#">transformDisplayToFrameBuffer()</a> [5/6]	246

7.54.2.6	transformDisplayToFramebuffer() [ 6 / 6 ]	246
7.54.2.7	transformFramebufferToDisplay()	247
7.55	DMA_Interface Class Reference	247
7.55.1	Detailed Description	248
7.55.2	Constructor & Destructor Documentation	248
7.55.2.1	~DMA_Interface()	248
7.55.2.2	DMA_Interface()	248
7.55.3	Member Function Documentation	249
7.55.3.1	addToQueue()	249
7.55.3.2	disableAlpha()	249
7.55.3.3	enableAlpha()	249
7.55.3.4	enableCopyWithTransparentPixels()	249
7.55.3.5	execute()	250
7.55.3.6	executeCompleted()	250
7.55.3.7	flush()	250
7.55.3.8	getAllowed()	250
7.55.3.9	getBlitCaps()	250
7.55.3.10	initialize()	250
7.55.3.11	isDmaQueueEmpty()	250
7.55.3.12	isDmaQueueFull()	251
7.55.3.13	isDMARunning()	251
7.55.3.14	seedExecution()	251
7.55.3.15	setAllowed()	251
7.55.3.16	setupDataCopy()	252
7.55.3.17	setupDataFill()	252
7.55.3.18	signalDMAInterrupt()	252
7.55.3.19	start()	252
7.55.3.20	waitForFramebufferSemaphore()	252
7.56	DMA_Queue Class Reference	252
7.56.1	Detailed Description	253
7.56.2	Constructor & Destructor Documentation	253
7.56.2.1	~DMA_Queue()	253
7.56.2.2	DMA_Queue()	253
7.56.3	Member Function Documentation	253
7.56.3.1	first()	254
7.56.3.2	isEmpty()	254
7.56.3.3	isFull()	254
7.56.3.4	pop()	254
7.56.3.5	pushCopyOf()	254
7.57	DragEvent Class Reference	255

7.57.1	Detailed Description	255
7.57.2	Constructor & Destructor Documentation	256
7.57.2.1	DragEvent()	256
7.57.2.2	~DragEvent()	256
7.57.3	Member Function Documentation	256
7.57.3.1	getDeltaX()	256
7.57.3.2	getDeltaY()	256
7.57.3.3	getEventType()	257
7.57.3.4	getNewX()	257
7.57.3.5	getNewY()	257
7.57.3.6	getOldX()	257
7.57.3.7	getOldY()	258
7.57.3.8	getType()	258
7.58	Draggable< T > Class Template Reference	258
7.58.1	Detailed Description	258
7.58.2	Constructor & Destructor Documentation	258
7.58.2.1	Draggable()	259
7.58.2.2	~Draggable()	259
7.58.3	Member Function Documentation	259
7.58.3.1	handleDragEvent()	259
7.59	Drawable Class Reference	259
7.59.1	Detailed Description	262
7.59.2	Member Enumeration Documentation	262
7.59.2.1	DrawableType	262
7.59.3	Constructor & Destructor Documentation	262
7.59.3.1	Drawable()	262
7.59.3.2	~Drawable()	262
7.59.4	Member Function Documentation	262
7.59.4.1	childGeometryChanged()	263
7.59.4.2	draw()	263
7.59.4.3	getAbsoluteRect()	263
7.59.4.4	getCachedAbsX()	263
7.59.4.5	getCachedAbsY()	264
7.59.4.6	getCachedVisibleRect()	264
7.59.4.7	getHeight()	264
7.59.4.8	getLastChild()	264
7.59.4.9	getNextSibling()	265
7.59.4.10	getParent()	265
7.59.4.11	getRect()	265
7.59.4.12	getSolidRect()	265

7.59.4.13	<a href="#">getSolidRectAbsolute()</a>	266
7.59.4.14	<a href="#">getType()</a>	266
7.59.4.15	<a href="#">getVisibleRect()</a>	266
7.59.4.16	<a href="#">getWidth()</a>	266
7.59.4.17	<a href="#">getX()</a>	267
7.59.4.18	<a href="#">getY()</a>	267
7.59.4.19	<a href="#">handleClickEvent()</a>	267
7.59.4.20	<a href="#">handleDragEvent()</a>	267
7.59.4.21	<a href="#">handleGestureEvent()</a>	268
7.59.4.22	<a href="#">handleTickEvent()</a>	268
7.59.4.23	<a href="#">invalidate()</a>	268
7.59.4.24	<a href="#">invalidateRect()</a>	268
7.59.4.25	<a href="#">isTouchable()</a>	269
7.59.4.26	<a href="#">isVisible()</a>	269
7.59.4.27	<a href="#">moveRelative()</a>	269
7.59.4.28	<a href="#">moveTo()</a>	270
7.59.4.29	<a href="#">resetDrawChainCache()</a>	270
7.59.4.30	<a href="#">setHeight()</a>	270
7.59.4.31	<a href="#">setPosition()</a>	271
7.59.4.32	<a href="#">setTouchable()</a>	271
7.59.4.33	<a href="#">setupDrawChain()</a>	271
7.59.4.34	<a href="#">setVisible()</a>	272
7.59.4.35	<a href="#">setWidth()</a>	272
7.59.4.36	<a href="#">setX()</a>	272
7.59.4.37	<a href="#">setXY()</a>	273
7.59.4.38	<a href="#">setY()</a>	273
7.59.4.39	<a href="#">translateRectToAbsolute()</a>	273
7.60	<a href="#">DrawableList Class Reference</a>	273
7.60.1	<a href="#">Detailed Description</a>	275
7.60.2	<a href="#">Constructor &amp; Destructor Documentation</a>	275
7.60.2.1	<a href="#">DrawableList()</a>	275
7.60.2.2	<a href="#">~DrawableList()</a>	275
7.60.3	<a href="#">Member Function Documentation</a>	275
7.60.3.1	<a href="#">getCircular()</a>	275
7.60.3.2	<a href="#">getDrawableIndex()</a>	275
7.60.3.3	<a href="#">getDrawableIndices()</a>	276
7.60.3.4	<a href="#">getDrawableMargin()</a>	276
7.60.3.5	<a href="#">getDrawableSize()</a>	277
7.60.3.6	<a href="#">getHorizontal()</a>	277
7.60.3.7	<a href="#">getItemIndex()</a>	277

7.60.3.8	<a href="#">getItemSize()</a>	277
7.60.3.9	<a href="#">getNumberOfDrawables()</a>	278
7.60.3.10	<a href="#">getNumberOfItems()</a>	278
7.60.3.11	<a href="#">getOffset()</a>	278
7.60.3.12	<a href="#">getRequiredNumberOfDrawables()</a>	279
7.60.3.13	<a href="#">itemChanged()</a>	279
7.60.3.14	<a href="#">refreshDrawables()</a>	279
7.60.3.15	<a href="#">setCircular()</a>	279
7.60.3.16	<a href="#">setDrawables()</a>	280
7.60.3.17	<a href="#">setDrawableSize()</a>	280
7.60.3.18	<a href="#">setHeight()</a>	280
7.60.3.19	<a href="#">setHorizontal()</a>	281
7.60.3.20	<a href="#">setNumberOfItems()</a>	281
7.60.3.21	<a href="#">setOffset()</a>	281
7.60.3.22	<a href="#">setWidth()</a>	282
7.61	<a href="#">DrawableListItems&lt; TYPE, SIZE &gt; Class Template Reference</a>	282
7.61.1	<a href="#">Detailed Description</a>	283
7.61.2	<a href="#">Constructor &amp; Destructor Documentation</a>	283
7.61.2.1	<a href="#">DrawableListItems()</a>	283
7.61.2.2	<a href="#">~DrawableListItems()</a>	283
7.61.3	<a href="#">Member Function Documentation</a>	283
7.61.3.1	<a href="#">getDrawable()</a>	283
7.61.3.2	<a href="#">getNumberOfDrawables()</a>	284
7.61.3.3	<a href="#">operator[]()</a>	284
7.62	<a href="#">DrawableListItemsInterface Class Reference</a>	284
7.62.1	<a href="#">Detailed Description</a>	284
7.62.2	<a href="#">Constructor &amp; Destructor Documentation</a>	285
7.62.2.1	<a href="#">~DrawableListItemsInterface()</a>	285
7.62.3	<a href="#">Member Function Documentation</a>	285
7.62.3.1	<a href="#">getDrawable()</a>	285
7.62.3.2	<a href="#">getNumberOfDrawables()</a>	285
7.63	<a href="#">DrawingSurface Struct Reference</a>	285
7.64	<a href="#">Bitmap::DynamicBitmapData Struct Reference</a>	286
7.64.1	<a href="#">Detailed Description</a>	286
7.65	<a href="#">EasingEquations Class Reference</a>	286
7.65.1	<a href="#">Detailed Description</a>	288
7.65.2	<a href="#">Member Function Documentation</a>	288
7.65.2.1	<a href="#">backEaseIn()</a>	288
7.65.2.2	<a href="#">backEaseInOut()</a>	288
7.65.2.3	<a href="#">backEaseOut()</a>	289



7.65.2.4	<a href="#">bounceEaseln()</a>	289
7.65.2.5	<a href="#">bounceEaselnOut()</a>	290
7.65.2.6	<a href="#">bounceEaseOut()</a>	290
7.65.2.7	<a href="#">circEaseln()</a>	291
7.65.2.8	<a href="#">circEaselnOut()</a>	291
7.65.2.9	<a href="#">circEaseOut()</a>	291
7.65.2.10	<a href="#">cubicEaseln()</a>	292
7.65.2.11	<a href="#">cubicEaselnOut()</a>	292
7.65.2.12	<a href="#">cubicEaseOut()</a>	293
7.65.2.13	<a href="#">elasticEaseln()</a>	293
7.65.2.14	<a href="#">elasticEaselnOut()</a>	294
7.65.2.15	<a href="#">elasticEaseOut()</a>	294
7.65.2.16	<a href="#">expoEaseln()</a>	294
7.65.2.17	<a href="#">expoEaselnOut()</a>	295
7.65.2.18	<a href="#">expoEaseOut()</a>	295
7.65.2.19	<a href="#">linearEaseln()</a>	296
7.65.2.20	<a href="#">linearEaselnOut()</a>	296
7.65.2.21	<a href="#">linearEaseNone()</a>	297
7.65.2.22	<a href="#">linearEaseOut()</a>	297
7.65.2.23	<a href="#">quadEaseln()</a>	297
7.65.2.24	<a href="#">quadEaselnOut()</a>	298
7.65.2.25	<a href="#">quadEaseOut()</a>	298
7.65.2.26	<a href="#">quartEaseln()</a>	299
7.65.2.27	<a href="#">quartEaselnOut()</a>	299
7.65.2.28	<a href="#">quartEaseOut()</a>	300
7.65.2.29	<a href="#">quintEaseln()</a>	300
7.65.2.30	<a href="#">quintEaselnOut()</a>	300
7.65.2.31	<a href="#">quintEaseOut()</a>	301
7.65.2.32	<a href="#">sineEaseln()</a>	301
7.65.2.33	<a href="#">sineEaselnOut()</a>	302
7.65.2.34	<a href="#">sineEaseOut()</a>	302
7.66	<a href="#">Edge Struct Reference</a>	303
7.66.1	<a href="#">Constructor &amp; Destructor Documentation</a>	304
7.66.1.1	<a href="#">Edge()</a>	304
7.66.2	<a href="#">Member Function Documentation</a>	304
7.66.2.1	<a href="#">step() [1/2]</a>	304
7.66.2.2	<a href="#">step() [2/2]</a>	304
7.67	<a href="#">Event Class Reference</a>	305
7.67.1	<a href="#">Detailed Description</a>	305
7.67.2	<a href="#">Member Enumeration Documentation</a>	305

7.67.2.1	EventType	305
7.67.3	Constructor & Destructor Documentation	305
7.67.3.1	~Event()	305
7.67.4	Member Function Documentation	306
7.67.4.1	getEventType()	306
7.68	FadeAnimator< T > Class Template Reference	306
7.68.1	Detailed Description	307
7.68.2	Constructor & Destructor Documentation	307
7.68.2.1	FadeAnimator()	307
7.68.2.2	~FadeAnimator()	307
7.68.3	Member Function Documentation	308
7.68.3.1	clearFadeAnimationEndedAction()	308
7.68.3.2	getFadeAnimationDelay()	308
7.68.3.3	handleTickEvent()	308
7.68.3.4	isFadeAnimationRunning()	308
7.68.3.5	isRunning()	308
7.68.3.6	nextFadeAnimationStep()	309
7.68.3.7	setFadeAnimationDelay()	309
7.68.3.8	setFadeAnimationEndedAction()	309
7.68.3.9	startFadeAnimation()	309
7.69	Font Class Reference	310
7.69.1	Detailed Description	311
7.69.2	Constructor & Destructor Documentation	311
7.69.2.1	Font()	311
7.69.3	Member Function Documentation	312
7.69.3.1	getBitsPerPixel()	312
7.69.3.2	getCharWidth()	312
7.69.3.3	getEllipsisChar()	312
7.69.3.4	getFallbackChar()	313
7.69.3.5	getFontHeight()	313
7.69.3.6	getGlyph() [1/2]	313
7.69.3.7	getGlyph() [2/2]	314
7.69.3.8	getKerning()	314
7.69.3.9	getMaxPixelsLeft()	314
7.69.3.10	getMaxPixelsRight()	314
7.69.3.11	getMaxTextHeight()	315
7.69.3.12	getMinimumTextHeight()	315
7.69.3.13	getNumberOfLines()	315
7.69.3.14	getSpacingAbove()	316
7.69.3.15	getStringWidth() [1/2]	316

7.69.3.16 getStringWidth() [2/2]	316
7.69.3.17 getStringWidthLTR()	317
7.69.3.18 getStringWidthRTL()	317
7.70 FontManager Class Reference	318
7.70.1 Detailed Description	318
7.70.2 Member Function Documentation	318
7.70.2.1 getFont()	318
7.70.2.2 setFontProvider()	318
7.71 FontProvider Class Reference	319
7.71.1 Detailed Description	319
7.71.2 Constructor & Destructor Documentation	319
7.71.2.1 ~FontProvider()	319
7.71.3 Member Function Documentation	319
7.71.3.1 getFont()	319
7.72 CoverTransition< templateDirection >::FullSolidRect Class Reference	319
7.72.1 Member Function Documentation	320
7.72.1.1 draw()	320
7.72.1.2 getSolidRect()	320
7.73 GenericCallback< T1, T2, T3 > Class Template Reference	320
7.73.1 Detailed Description	321
7.73.2 Constructor & Destructor Documentation	321
7.73.2.1 ~GenericCallback()	321
7.73.3 Member Function Documentation	321
7.73.3.1 execute()	322
7.73.3.2 isValid()	322
7.74 GenericCallback< T1, T2, void > Class Template Reference	322
7.74.1 Detailed Description	323
7.74.2 Constructor & Destructor Documentation	323
7.74.2.1 ~GenericCallback()	323
7.74.3 Member Function Documentation	323
7.74.3.1 execute()	323
7.74.3.2 isValid()	324
7.75 GenericCallback< T1, void, void > Class Template Reference	324
7.75.1 Detailed Description	324
7.75.2 Constructor & Destructor Documentation	325
7.75.2.1 ~GenericCallback()	325
7.75.3 Member Function Documentation	325
7.75.3.1 execute()	325
7.75.3.2 isValid()	325
7.76 GenericCallback< void > Class Template Reference	325

7.76.1 Detailed Description . . . . .	326
7.76.2 Constructor & Destructor Documentation . . . . .	326
7.76.2.1 ~GenericCallback() . . . . .	326
7.76.3 Member Function Documentation . . . . .	326
7.76.3.1 execute() . . . . .	326
7.76.3.2 isValid() . . . . .	326
7.77 GestureEvent Class Reference . . . . .	327
7.77.1 Detailed Description . . . . .	327
7.77.2 Member Enumeration Documentation . . . . .	327
7.77.2.1 GestureType . . . . .	327
7.77.3 Constructor & Destructor Documentation . . . . .	328
7.77.3.1 GestureEvent() . . . . .	328
7.77.4 Member Function Documentation . . . . .	328
7.77.4.1 getEventType() . . . . .	328
7.77.4.2 getType() . . . . .	328
7.77.4.3 getVelocity() . . . . .	328
7.77.4.4 getX() . . . . .	329
7.77.4.5 getY() . . . . .	329
7.78 Gestures Class Reference . . . . .	329
7.78.1 Detailed Description . . . . .	329
7.78.2 Constructor & Destructor Documentation . . . . .	330
7.78.2.1 Gestures() . . . . .	330
7.78.3 Member Function Documentation . . . . .	330
7.78.3.1 registerClickEvent() . . . . .	330
7.78.3.2 registerDragEvent() . . . . .	330
7.78.3.3 registerEventListener() . . . . .	331
7.78.3.4 setDragThreshold() . . . . .	331
7.78.3.5 tick() . . . . .	331
7.79 GlyphNode Struct Reference . . . . .	331
7.79.1 Detailed Description . . . . .	332
7.79.2 Member Function Documentation . . . . .	332
7.79.2.1 advance() . . . . .	332
7.79.2.2 height() . . . . .	332
7.79.2.3 top() . . . . .	333
7.79.2.4 width() . . . . .	333
7.80 GPIO Class Reference . . . . .	333
7.80.1 Detailed Description . . . . .	333
7.80.2 Member Enumeration Documentation . . . . .	333
7.80.2.1 GPIO_ID . . . . .	334
7.80.3 Member Function Documentation . . . . .	334

7.80.3.1	clear()	334
7.80.3.2	get()	334
7.80.3.3	init()	334
7.80.3.4	set()	335
7.80.3.5	toggle()	335
7.81	Gradients Struct Reference	335
7.81.1	Constructor & Destructor Documentation	336
7.81.1.1	Gradients()	336
7.82	HAL Class Reference	336
7.82.1	Detailed Description	340
7.82.2	Member Enumeration Documentation	341
7.82.2.1	FrameRefreshStrategy	341
7.82.3	Constructor & Destructor Documentation	341
7.82.3.1	HAL()	341
7.82.3.2	~HAL()	341
7.82.4	Member Function Documentation	341
7.82.4.1	allowDMATransfers()	342
7.82.4.2	backPorchExited()	342
7.82.4.3	beginFrame()	342
7.82.4.4	blitCopy()	342
7.82.4.5	blitCopyARGB8888()	343
7.82.4.6	blitCopyGlyph()	343
7.82.4.7	blitFill()	344
7.82.4.8	blitSetTransparencyKey()	344
7.82.4.9	blockCopy()	344
7.82.4.10	cacheTextString()	345
7.82.4.11	configureInterrupts()	345
7.82.4.12	copyFBRegionToMemory() [1/2]	345
7.82.4.13	copyFBRegionToMemory() [2/2]	346
7.82.4.14	disableInterrupts()	346
7.82.4.15	enableInterrupts()	346
7.82.4.16	enableLCDControllerInterrupt()	347
7.82.4.17	enableMCULoadCalculation()	347
7.82.4.18	endFrame()	347
7.82.4.19	flushDMA()	347
7.82.4.20	flushFrameBuffer() [1/2]	347
7.82.4.21	flushFrameBuffer() [2/2]	347
7.82.4.22	frontPorchEntered()	348
7.82.4.23	getAnimationStorage()	348
7.82.4.24	getBlitCaps()	348

7.82.4.25	getButtonController()	348
7.82.4.26	getClientFrameBuffer()	348
7.82.4.27	getCPUCycles()	349
7.82.4.28	getDisplayHeight()	349
7.82.4.29	getDisplayOrientation()	349
7.82.4.30	getDisplayWidth()	349
7.82.4.31	getFingerSize()	349
7.82.4.32	getFrameRefreshStrategy()	350
7.82.4.33	getInstance()	350
7.82.4.34	getLCDRefreshCount()	350
7.82.4.35	getMCULoadPct()	350
7.82.4.36	getTFTCurrentLine()	350
7.82.4.37	getTFTFrameBuffer()	351
7.82.4.38	getTouchSampleRate()	351
7.82.4.39	initialize()	351
7.82.4.40	lcd()	351
7.82.4.41	lockDMAToFrontPorch()	352
7.82.4.42	lockFrameBuffer()	352
7.82.4.43	noTouch()	352
7.82.4.44	performDisplayOrientationChange()	352
7.82.4.45	registerEventListener()	353
7.82.4.46	registerTaskDelayFunction()	353
7.82.4.47	registerTextCache()	353
7.82.4.48	sampleKey()	354
7.82.4.49	setButtonController()	354
7.82.4.50	setDisplayOrientation()	354
7.82.4.51	setDragThreshold()	354
7.82.4.52	setFingerSize()	355
7.82.4.53	setFrameBufferStartAddress()	355
7.82.4.54	setFrameBufferStartAddresses()	356
7.82.4.55	setFrameRateCompensation()	356
7.82.4.56	setFrameRefreshStrategy()	356
7.82.4.57	setMCUActive()	357
7.82.4.58	setMCUInstrumentation()	357
7.82.4.59	setTFTFrameBuffer()	357
7.82.4.60	setTouchSampleRate()	357
7.82.4.61	signalDMAInterrupt()	358
7.82.4.62	swapFrameBuffers()	358
7.82.4.63	taskDelay()	358
7.82.4.64	taskEntry()	358

7.82.4.65 tick()	358
7.82.4.66 touch()	359
7.82.4.67 unlockFrameBuffer()	359
7.82.4.68 vSync()	359
7.83 HALSDL2 Class Reference	359
7.83.1 Detailed Description	360
7.83.2 Constructor & Destructor Documentation	361
7.83.2.1 HALSDL2()	361
7.83.3 Member Function Documentation	361
7.83.3.1 blitSetTransparencyKey()	361
7.83.3.2 blockCopy()	361
7.83.3.3 configureInterrupts()	362
7.83.3.4 configureLCDInterrupt()	362
7.83.3.5 copyScreenshotToClipboard()	362
7.83.3.6 disableInterrupts()	362
7.83.3.7 doSampleTouch()	362
7.83.3.8 enableInterrupts()	363
7.83.3.9 enableLCDControllerInterrupt()	363
7.83.3.10 flushFrameBuffer() [1/2]	363
7.83.3.11 flushFrameBuffer() [2/2]	363
7.83.3.12 getArgv()	363
7.83.3.13 getTFTFrameBuffer()	364
7.83.3.14 getWindowTitle()	364
7.83.3.15 loadSkin()	364
7.83.3.16 performDisplayOrientationChange()	365
7.83.3.17 renderLCD_FrameBufferToMemory()	365
7.83.3.18 sampleKey()	365
7.83.3.19 saveNextScreenshots()	365
7.83.3.20 saveScreenshot() [1/2]	366
7.83.3.21 saveScreenshot() [2/2]	366
7.83.3.22 sdl_init()	366
7.83.3.23 setTFTFrameBuffer()	366
7.83.3.24 setVsyncInterval()	367
7.83.3.25 setTitle()	367
7.83.3.26 taskEntry()	367
7.84 I2C Class Reference	368
7.84.1 Detailed Description	368
7.84.2 Constructor & Destructor Documentation	368
7.84.2.1 I2C()	368
7.84.2.2 ~I2C()	369

7.84.3	Member Function Documentation	369
7.84.3.1	init()	369
7.84.3.2	readRegister()	369
7.84.3.3	writeRegister()	369
7.85	I2CTouchController Class Reference	370
7.85.1	Detailed Description	370
7.85.2	Constructor & Destructor Documentation	370
7.85.2.1	I2CTouchController()	370
7.85.2.2	~I2CTouchController()	371
7.85.3	Member Function Documentation	371
7.85.3.1	init()	371
7.85.3.2	sampleTouch()	371
7.86	IconButtonStyle< T > Class Template Reference	371
7.86.1	Detailed Description	372
7.86.2	Member Function Documentation	373
7.86.2.1	getCurrentlyDisplayedIcon()	373
7.86.2.2	getIconX()	373
7.86.2.3	getIconY()	373
7.86.2.4	setIconBitmaps()	373
7.86.2.5	setIconX()	373
7.86.2.6	setIconXY()	374
7.86.2.7	setIconY()	374
7.87	Image Class Reference	374
7.87.1	Detailed Description	375
7.87.2	Constructor & Destructor Documentation	375
7.87.2.1	Image()	375
7.87.3	Member Function Documentation	375
7.87.3.1	draw()	375
7.87.3.2	getAlpha()	376
7.87.3.3	getBitmap()	376
7.87.3.4	getSolidRect()	376
7.87.3.5	getType()	376
7.87.3.6	setAlpha()	377
7.87.3.7	setBitmap()	377
7.88	ImageButtonStyle< T > Class Template Reference	377
7.88.1	Detailed Description	378
7.88.2	Member Function Documentation	378
7.88.2.1	getCurrentlyDisplayedBitmap()	378
7.88.2.2	setBitmaps()	379
7.88.2.3	setBitmapXY()	379



7.89 ImageProgress Class Reference . . . . .	379
7.89.1 Detailed Description . . . . .	380
7.89.2 Constructor & Destructor Documentation . . . . .	380
7.89.2.1 ImageProgress() . . . . .	380
7.89.2.2 ~ImageProgress() . . . . .	380
7.89.3 Member Function Documentation . . . . .	380
7.89.3.1 getAlpha() . . . . .	380
7.89.3.2 getAnchorAtZero() . . . . .	381
7.89.3.3 getBitmap() . . . . .	381
7.89.3.4 setAlpha() . . . . .	381
7.89.3.5 setAnchorAtZero() . . . . .	381
7.89.3.6 setBitmap() . . . . .	382
7.89.3.7 setProgressIndicatorPosition() . . . . .	382
7.89.3.8 setValue() . . . . .	383
7.90 InternalFlashFont Class Reference . . . . .	383
7.90.1 Detailed Description . . . . .	383
7.90.2 Constructor & Destructor Documentation . . . . .	383
7.90.2.1 InternalFlashFont() . . . . .	383
7.90.3 Member Function Documentation . . . . .	384
7.90.3.1 getKerning() . . . . .	384
7.90.3.2 getPixelData() . . . . .	384
7.91 Scanline::iterator Class Reference . . . . .	385
7.91.1 Detailed Description . . . . .	385
7.91.2 Constructor & Destructor Documentation . . . . .	385
7.91.2.1 iterator() . . . . .	385
7.91.3 Member Function Documentation . . . . .	386
7.91.3.1 getCovers() . . . . .	386
7.91.3.2 getNumPix() . . . . .	386
7.91.3.3 next() . . . . .	386
7.92 JSMOCHelper Class Reference . . . . .	386
7.92.1 Detailed Description . . . . .	387
7.92.2 Constructor & Destructor Documentation . . . . .	387
7.92.2.1 JSMOCHelper() . . . . .	387
7.92.3 Member Function Documentation . . . . .	387
7.92.3.1 draw() . . . . .	387
7.92.3.2 getCachedAbsX() . . . . .	387
7.92.3.3 getCachedAbsY() . . . . .	388
7.92.3.4 getCachedVisibleRect() . . . . .	388
7.92.3.5 getHeight() . . . . .	388
7.92.3.6 getWidget() . . . . .	388

7.92.3.7	getWidth()	388
7.92.3.8	setWidget()	389
7.93	KerningNode Struct Reference	389
7.93.1	Detailed Description	389
7.94	Keyboard::Key Struct Reference	389
7.95	Keyboard Class Reference	389
7.95.1	Detailed Description	391
7.95.2	Constructor & Destructor Documentation	392
7.95.2.1	Keyboard()	392
7.95.2.2	~Keyboard()	392
7.95.3	Member Function Documentation	392
7.95.3.1	draw()	392
7.95.3.2	getBuffer()	392
7.95.3.3	getBufferPosition()	392
7.95.3.4	getCallbackAreaForCoordinates()	393
7.95.3.5	getCharForKey()	393
7.95.3.6	getKeyForCoordinates()	393
7.95.3.7	getKeyMappingList()	394
7.95.3.8	getLayout()	394
7.95.3.9	getType()	394
7.95.3.10	handleClickEvent()	394
7.95.3.11	handleDragEvent()	395
7.95.3.12	setBuffer()	395
7.95.3.13	setBufferPosition()	395
7.95.3.14	setKeyListener()	395
7.95.3.15	setKeymappingList()	396
7.95.3.16	setLayout()	396
7.95.3.17	setTextIndentation()	396
7.95.3.18	setupDrawChain()	396
7.96	Keyboard::KeyMapping Struct Reference	397
7.97	Keyboard::KeyMappingList Struct Reference	397
7.98	Keyboard::Layout Struct Reference	397
7.99	LCD Class Reference	398
7.99.1	Detailed Description	400
7.99.2	Constructor & Destructor Documentation	400
7.99.2.1	~LCD()	400
7.99.3	Member Function Documentation	400
7.99.3.1	bitDepth()	400
7.99.3.2	blitCopy() [1/2]	401
7.99.3.3	blitCopy() [2/2]	401

7.99.3.4	<a href="#">copyFrameBufferRegionToMemory()</a>	402
7.99.3.5	<a href="#">drawBorder()</a>	402
7.99.3.6	<a href="#">drawGlyph()</a> [1/2]	402
7.99.3.7	<a href="#">drawGlyph()</a> [2/2]	403
7.99.3.8	<a href="#">drawHorizontalLine()</a>	404
7.99.3.9	<a href="#">drawPartialBitmap()</a>	404
7.99.3.10	<a href="#">drawRect()</a>	405
7.99.3.11	<a href="#">drawString()</a>	405
7.99.3.12	<a href="#">drawStringLTR()</a>	406
7.99.3.13	<a href="#">drawStringRTL()</a>	406
7.99.3.14	<a href="#">drawTextureMapScanLine()</a>	407
7.99.3.15	<a href="#">drawTextureMapTriangle()</a>	407
7.99.3.16	<a href="#">drawVerticalLine()</a>	408
7.99.3.17	<a href="#">fillRect()</a>	408
7.99.3.18	<a href="#">getNumLines()</a>	409
7.99.3.19	<a href="#">init()</a>	409
7.99.3.20	<a href="#">realX()</a>	409
7.99.3.21	<a href="#">realY()</a>	410
7.99.3.22	<a href="#">rotateRect()</a>	410
7.99.3.23	<a href="#">stringWidth()</a>	410
7.100	<a href="#">LCD16bpp Class Reference</a>	411
7.100.1	<a href="#">Detailed Description</a>	412
7.100.2	<a href="#">Member Function Documentation</a>	412
7.100.2.1	<a href="#">bitDepth()</a>	412
7.100.2.2	<a href="#">blitCopy()</a> [1/2]	413
7.100.2.3	<a href="#">blitCopy()</a> [2/2]	413
7.100.2.4	<a href="#">blitCopyAlphaPerPixel()</a>	414
7.100.2.5	<a href="#">blitCopyARGB8888()</a>	414
7.100.2.6	<a href="#">copyFrameBufferRegionToMemory()</a>	414
7.100.2.7	<a href="#">drawGlyph()</a>	415
7.100.2.8	<a href="#">drawPartialBitmap()</a>	415
7.100.2.9	<a href="#">drawTextureMapScanLine()</a>	416
7.100.2.10	<a href="#">fillRect()</a>	417
7.100.2.11	<a href="#">init()</a>	417
7.100.2.12	<a href="#">nextLine()</a>	417
7.100.2.13	<a href="#">nextPixel()</a>	417
7.101	<a href="#">LCD1bpp Class Reference</a>	418
7.101.1	<a href="#">Detailed Description</a>	419
7.101.2	<a href="#">Member Function Documentation</a>	419
7.101.2.1	<a href="#">bitDepth()</a>	419

7.101.2.2 blitCopy() [1/2]	419
7.101.2.3 blitCopy() [2/2]	420
7.101.2.4 blitCopyRLE()	420
7.101.2.5 copyFrameBufferRegionToMemory()	421
7.101.2.6 drawGlyph()	421
7.101.2.7 drawPartialBitmap()	422
7.101.2.8 drawTextureMapScanLine()	422
7.101.2.9 fillMemory()	423
7.101.2.10 fillRect()	423
7.101.2.11 nextLine()	424
7.101.2.12 nextPixel()	424
7.102LCD24bpp Class Reference	424
7.102.1 Detailed Description	425
7.102.2 Member Function Documentation	426
7.102.2.1 bitDepth()	426
7.102.2.2 blitCopy() [1/2]	426
7.102.2.3 blitCopy() [2/2]	426
7.102.2.4 blitCopyARGB8888()	427
7.102.2.5 copyFrameBufferRegionToMemory()	427
7.102.2.6 drawGlyph()	428
7.102.2.7 drawPartialBitmap()	428
7.102.2.8 drawTextureMapScanLine()	429
7.102.2.9 fillRect()	429
7.102.2.10 init()	430
7.102.2.11 nextLine()	430
7.102.2.12 nextPixel()	430
7.103LCD2bpp Class Reference	431
7.103.1 Detailed Description	432
7.103.2 Member Function Documentation	432
7.103.2.1 bitDepth()	432
7.103.2.2 blitCopy() [1/2]	432
7.103.2.3 blitCopy() [2/2]	433
7.103.2.4 blitCopyAlphaPerPixel()	433
7.103.2.5 copyFrameBufferRegionToMemory()	434
7.103.2.6 drawGlyph()	434
7.103.2.7 drawPartialBitmap()	435
7.103.2.8 drawTextureMapScanLine()	435
7.103.2.9 fillRect()	436
7.103.2.10 init()	436
7.103.2.11 nextLine()	437

7.103.2.12nextPixel()	437
7.104LCD4bpp Class Reference	437
7.104.1 Detailed Description	438
7.104.2 Member Function Documentation	439
7.104.2.1 bitDepth()	439
7.104.2.2 blitCopy() [1/2]	439
7.104.2.3 blitCopy() [2/2]	439
7.104.2.4 blitCopyAlphaPerPixel()	440
7.104.2.5 copyFrameBufferRegionToMemory()	440
7.104.2.6 drawGlyph()	441
7.104.2.7 drawPartialBitmap()	441
7.104.2.8 drawTextureMapScanLine()	442
7.104.2.9 fillRect()	443
7.104.2.10init()	443
7.104.2.11nextLine()	443
7.104.2.12nextPixel()	443
7.105LED Class Reference	444
7.105.1 Member Function Documentation	444
7.105.1.1 get()	444
7.105.1.2 init()	444
7.105.1.3 off()	445
7.105.1.4 on()	445
7.105.1.5 toggle()	445
7.106Line Class Reference	445
7.106.1 Detailed Description	447
7.106.2 Member Enumeration Documentation	447
7.106.2.1 LINE_ENDING_STYLE	447
7.106.3 Constructor & Destructor Documentation	447
7.106.3.1 Line()	447
7.106.4 Member Function Documentation	448
7.106.4.1 drawCanvasWidget()	448
7.106.4.2 getEnd()	448
7.106.4.3 getLineEndingStyle()	448
7.106.4.4 getLineWidth() [1/2]	449
7.106.4.5 getLineWidth() [2/2]	449
7.106.4.6 getMinimalRect()	450
7.106.4.7 getStart()	450
7.106.4.8 setCapPrecision()	450
7.106.4.9 setEnd() [1/2]	451
7.106.4.10setEnd() [2/2]	451

7.106.4.11	setLine()	452
7.106.4.12	setLineEndingStyle()	452
7.106.4.13	setLineWidth() [1/2]	453
7.106.4.14	setLineWidth() [2/2]	453
7.106.4.15	setStart() [1/2]	453
7.106.4.16	setStart() [2/2]	454
7.106.4.17	updateEnd() [1/2]	454
7.106.4.18	updateEnd() [2/2]	455
7.106.4.19	updateLineWidth() [1/2]	455
7.106.4.20	updateLineWidth() [2/2]	456
7.106.4.21	updateStart() [1/2]	456
7.106.4.22	updateStart() [2/2]	457
7.107	LineProgress Class Reference	457
7.107.1	Detailed Description	458
7.107.2	Constructor & Destructor Documentation	458
7.107.2.1	LineProgress()	458
7.107.2.2	~LineProgress()	458
7.107.3	Member Function Documentation	458
7.107.3.1	getAlpha()	459
7.107.3.2	getEnd()	459
7.107.3.3	getLineEndingStyle()	459
7.107.3.4	getLineWidth()	459
7.107.3.5	getStart()	460
7.107.3.6	setAlpha()	460
7.107.3.7	setEnd()	460
7.107.3.8	setLineEndingStyle()	460
7.107.3.9	setLineWidth()	461
7.107.3.10	setPainter()	461
7.107.3.11	setProgressIndicatorPosition()	461
7.107.3.12	setStart()	462
7.107.3.13	setValue()	462
7.108	ListLayout Class Reference	462
7.108.1	Detailed Description	463
7.108.2	Constructor & Destructor Documentation	463
7.108.2.1	ListLayout()	463
7.108.2.2	~ListLayout()	463
7.108.3	Member Function Documentation	464
7.108.3.1	add()	464
7.108.3.2	getDirection()	464
7.108.3.3	getType()	464

7.108.3.4 insert()	464
7.108.3.5 remove()	465
7.108.3.6 removeAll()	465
7.108.3.7 setDirection()	465
7.109 LockFreeDMA_Queue Class Reference	465
7.109.1 Detailed Description	466
7.109.2 Constructor & Destructor Documentation	466
7.109.2.1 LockFreeDMA_Queue()	466
7.109.3 Member Function Documentation	467
7.109.3.1 first()	467
7.109.3.2 isEmpty()	467
7.109.3.3 isFull()	467
7.109.3.4 pop()	467
7.109.3.5 pushCopyOf()	467
7.110 Matrix4x4 Class Reference	468
7.110.1 Detailed Description	469
7.110.2 Constructor & Destructor Documentation	469
7.110.2.1 Matrix4x4()	469
7.110.3 Member Function Documentation	469
7.110.3.1 concatenateXRotation()	469
7.110.3.2 concatenateXScale()	469
7.110.3.3 concatenateXTranslation()	470
7.110.3.4 concatenateYRotation()	470
7.110.3.5 concatenateYScale()	470
7.110.3.6 concatenateYTranslation()	470
7.110.3.7 concatenateZRotation()	471
7.110.3.8 concatenateZScale()	471
7.110.3.9 concatenateZTranslation()	471
7.110.3.10 getElement()	472
7.110.3.11 identity()	472
7.110.3.12 setElement()	472
7.110.3.13 setViewDistance()	473
7.111 MCUInstrumentation Class Reference	473
7.111.1 Detailed Description	473
7.111.2 Constructor & Destructor Documentation	473
7.111.2.1 MCUInstrumentation()	474
7.111.2.2 ~MCUInstrumentation()	474
7.111.3 Member Function Documentation	474
7.111.3.1 getCCConsumed()	474
7.111.3.2 getCPUCycles()	474

7.111.3.3 getElapsedUS()	474
7.111.3.4 init()	475
7.111.3.5 setCCConsumed()	475
7.111.3.6 setMCUActive()	475
7.112ModalWindow Class Reference	475
7.112.1 Detailed Description	476
7.112.2 Constructor & Destructor Documentation	476
7.112.2.1 ModalWindow()	477
7.112.2.2 ~ModalWindow()	477
7.112.3 Member Function Documentation	477
7.112.3.1 add()	477
7.112.3.2 getBackgroundHeight()	477
7.112.3.3 getBackgroundWidth()	477
7.112.3.4 getShadeAlpha()	478
7.112.3.5 getShadeColor()	478
7.112.3.6 hide()	478
7.112.3.7 isShowing()	478
7.112.3.8 remove()	478
7.112.3.9 setBackground() [1/2]	479
7.112.3.10setBackground() [2/2]	479
7.112.3.11setShadeAlpha()	479
7.112.3.12setShadeColor()	479
7.112.3.13show()	480
7.113MoveAnimator< T > Class Template Reference	480
7.113.1 Detailed Description	481
7.113.2 Constructor & Destructor Documentation	481
7.113.2.1 MoveAnimator()	481
7.113.2.2 ~MoveAnimator()	482
7.113.3 Member Function Documentation	482
7.113.3.1 cancelMoveAnimation()	482
7.113.3.2 clearMoveAnimationEndedAction()	482
7.113.3.3 getMoveAnimationDelay()	482
7.113.3.4 handleTickEvent()	482
7.113.3.5 isMoveAnimationRunning()	482
7.113.3.6 isRunning()	483
7.113.3.7 nextMoveAnimationStep()	483
7.113.3.8 setMoveAnimationDelay()	483
7.113.3.9 setMoveAnimationEndedAction()	483
7.113.3.10startMoveAnimation()	483
7.114MVPAApplication Class Reference	484



7.114.1 Detailed Description . . . . .	484
7.114.2 Constructor & Destructor Documentation . . . . .	485
7.114.2.1 MVPApplication() . . . . .	485
7.114.2.2 ~MVPApplication() . . . . .	485
7.114.3 Member Function Documentation . . . . .	485
7.114.3.1 evaluatePendingScreenTransition() . . . . .	485
7.114.3.2 handlePendingScreenTransition() . . . . .	485
7.115MVPHeap Class Reference . . . . .	485
7.115.1 Detailed Description . . . . .	486
7.115.2 Constructor & Destructor Documentation . . . . .	486
7.115.2.1 MVPHeap() . . . . .	486
7.115.2.2 ~MVPHeap() . . . . .	486
7.116NoDMA Class Reference . . . . .	486
7.116.1 Detailed Description . . . . .	487
7.116.2 Constructor & Destructor Documentation . . . . .	487
7.116.2.1 NoDMA() . . . . .	487
7.116.3 Member Function Documentation . . . . .	487
7.116.3.1 flush() . . . . .	487
7.116.3.2 getBlitCaps() . . . . .	488
7.116.3.3 setupDataCopy() . . . . .	488
7.116.3.4 setupDataFill() . . . . .	488
7.116.3.5 signalDMAInterrupt() . . . . .	488
7.117NoTouchController Class Reference . . . . .	488
7.117.1 Detailed Description . . . . .	489
7.117.2 Constructor & Destructor Documentation . . . . .	489
7.117.2.1 ~NoTouchController() . . . . .	489
7.117.3 Member Function Documentation . . . . .	489
7.117.3.1 init() . . . . .	489
7.117.3.2 sampleTouch() . . . . .	489
7.118NoTransition Class Reference . . . . .	490
7.118.1 Detailed Description . . . . .	490
7.118.2 Constructor & Destructor Documentation . . . . .	490
7.118.2.1 NoTransition() . . . . .	490
7.118.2.2 ~NoTransition() . . . . .	490
7.118.3 Member Function Documentation . . . . .	490
7.118.3.1 handleTickEvent() . . . . .	490
7.119OSWrappers Class Reference . . . . .	491
7.119.1 Detailed Description . . . . .	491
7.119.2 Member Function Documentation . . . . .	491
7.119.2.1 giveFrameBufferSemaphore() . . . . .	491

7.119.2.2 giveFrameBufferSemaphoreFromISR()	491
7.119.2.3 initialize()	491
7.119.2.4 signalVSync()	492
7.119.2.5 takeFrameBufferSemaphore()	492
7.119.2.6 taskDelay()	492
7.119.2.7 tryTakeFrameBufferSemaphore()	492
7.119.2.8 waitForVSync()	492
7.120 Outline Class Reference	493
7.120.1 Detailed Description	493
7.120.2 Member Typedef Documentation	494
7.120.2.1 OutlineFlags_t	494
7.120.3 Constructor & Destructor Documentation	494
7.120.3.1 Outline()	494
7.120.3.2 ~Outline()	494
7.120.4 Member Function Documentation	494
7.120.4.1 getCells()	494
7.120.4.2 getNumCells()	494
7.120.4.3 lineTo()	495
7.120.4.4 moveTo()	495
7.120.4.5 reset()	495
7.120.4.6 setMaxRenderY()	495
7.120.4.7 wasOutlineTooComplex()	495
7.121 PainterBW Class Reference	496
7.121.1 Detailed Description	496
7.121.2 Member Function Documentation	496
7.121.2.1 bw()	497
7.121.2.2 getColor()	497
7.121.2.3 render()	497
7.121.2.4 renderNext()	498
7.121.2.5 setColor()	498
7.122 PainterBWBitmap Class Reference	498
7.122.1 Detailed Description	499
7.122.2 Constructor & Destructor Documentation	499
7.122.2.1 PainterBWBitmap()	499
7.122.3 Member Function Documentation	499
7.122.3.1 render()	499
7.122.3.2 renderInit()	500
7.122.3.3 renderNext()	500
7.122.3.4 setBitmap()	500
7.123 PainterGRAY2 Class Reference	501

7.123.1 Detailed Description . . . . .	501
7.123.2 Constructor & Destructor Documentation . . . . .	501
7.123.2.1 PainterGRAY2() . . . . .	502
7.123.3 Member Function Documentation . . . . .	502
7.123.3.1 getAlpha() . . . . .	502
7.123.3.2 getColor() . . . . .	502
7.123.3.3 render() . . . . .	502
7.123.3.4 renderNext() . . . . .	503
7.123.3.5 setAlpha() . . . . .	503
7.123.3.6 setColor() . . . . .	503
7.124PainterGRAY2Bitmap Class Reference . . . . .	504
7.124.1 Detailed Description . . . . .	505
7.124.2 Constructor & Destructor Documentation . . . . .	505
7.124.2.1 PainterGRAY2Bitmap() . . . . .	505
7.124.3 Member Function Documentation . . . . .	505
7.124.3.1 getAlpha() . . . . .	505
7.124.3.2 render() . . . . .	505
7.124.3.3 renderInit() . . . . .	506
7.124.3.4 renderNext() . . . . .	506
7.124.3.5 setAlpha() . . . . .	507
7.124.3.6 setBitmap() . . . . .	507
7.125PainterGRAY4 Class Reference . . . . .	507
7.125.1 Detailed Description . . . . .	508
7.125.2 Constructor & Destructor Documentation . . . . .	508
7.125.2.1 PainterGRAY4() . . . . .	508
7.125.3 Member Function Documentation . . . . .	508
7.125.3.1 getAlpha() . . . . .	508
7.125.3.2 getColor() . . . . .	509
7.125.3.3 render() . . . . .	509
7.125.3.4 renderNext() . . . . .	509
7.125.3.5 setAlpha() . . . . .	510
7.125.3.6 setColor() . . . . .	510
7.126PainterGRAY4Bitmap Class Reference . . . . .	510
7.126.1 Detailed Description . . . . .	511
7.126.2 Constructor & Destructor Documentation . . . . .	511
7.126.2.1 PainterGRAY4Bitmap() . . . . .	511
7.126.3 Member Function Documentation . . . . .	511
7.126.3.1 getAlpha() . . . . .	511
7.126.3.2 render() . . . . .	512
7.126.3.3 renderInit() . . . . .	512

7.126.3.4 renderNext()	512
7.126.3.5 setAlpha()	513
7.126.3.6 setBitmap()	513
7.127PainterRGB565 Class Reference	513
7.127.1 Detailed Description	514
7.127.2 Constructor & Destructor Documentation	514
7.127.2.1 PainterRGB565()	514
7.127.3 Member Function Documentation	515
7.127.3.1 getAlpha()	515
7.127.3.2 getColor()	515
7.127.3.3 render()	515
7.127.3.4 renderNext()	516
7.127.3.5 setAlpha()	516
7.127.3.6 setColor()	516
7.128PainterRGB565Bitmap Class Reference	517
7.128.1 Detailed Description	518
7.128.2 Constructor & Destructor Documentation	518
7.128.2.1 PainterRGB565Bitmap()	518
7.128.3 Member Function Documentation	518
7.128.3.1 getAlpha()	518
7.128.3.2 render()	518
7.128.3.3 renderInit()	519
7.128.3.4 renderNext()	519
7.128.3.5 setAlpha()	520
7.128.3.6 setBitmap()	520
7.129PainterRGB888 Class Reference	520
7.129.1 Detailed Description	521
7.129.2 Constructor & Destructor Documentation	521
7.129.2.1 PainterRGB888()	521
7.129.3 Member Function Documentation	521
7.129.3.1 getAlpha()	521
7.129.3.2 getColor()	522
7.129.3.3 render()	522
7.129.3.4 renderNext()	522
7.129.3.5 setAlpha()	523
7.129.3.6 setColor()	523
7.130PainterRGB888Bitmap Class Reference	523
7.130.1 Detailed Description	524
7.130.2 Constructor & Destructor Documentation	524
7.130.2.1 PainterRGB888Bitmap()	524

7.130.3 Member Function Documentation	525
7.130.3.1 getAlpha()	525
7.130.3.2 render()	525
7.130.3.3 renderInit()	526
7.130.3.4 renderNext()	526
7.130.3.5 setAlpha()	526
7.130.3.6 setBitmap()	526
7.131 Pair< T1, T2 > Struct Template Reference	527
7.131.1 Detailed Description	527
7.131.2 Constructor & Destructor Documentation	527
7.131.2.1 Pair() [1/3]	527
7.131.2.2 Pair() [2/3]	528
7.131.2.3 Pair() [3/3]	528
7.132 Partition< ListOfTypes, NUMBER_OF_ELEMENTS > Class Template Reference	528
7.132.1 Detailed Description	529
7.132.2 Member Typedef Documentation	529
7.132.2.1 SupportedTypesList	529
7.132.3 Member Enumeration Documentation	529
7.132.3.1 anonymous enum	529
7.132.4 Constructor & Destructor Documentation	530
7.132.4.1 Partition()	530
7.132.5 Member Function Documentation	530
7.132.5.1 capacity()	530
7.132.5.2 element() [1/2]	530
7.132.5.3 element() [2/2]	530
7.132.5.4 element_size()	531
7.133 PixelDataWidget Class Reference	531
7.133.1 Detailed Description	532
7.133.2 Constructor & Destructor Documentation	532
7.133.2.1 PixelDataWidget()	532
7.133.3 Member Function Documentation	532
7.133.3.1 draw()	532
7.133.3.2 getAlpha()	533
7.133.3.3 getSolidRect()	533
7.133.3.4 setAlpha()	533
7.133.3.5 setBitmapFormat()	533
7.133.3.6 setPixelData()	534
7.134 Point Struct Reference	534
7.134.1 Member Function Documentation	534
7.134.1.1 dist_sqr()	534

7.135Point3D Struct Reference . . . . .	534
7.136Point4 Class Reference . . . . .	535
7.136.1 Detailed Description . . . . .	535
7.136.2 Constructor & Destructor Documentation . . . . .	535
7.136.2.1 Point4() [1/2] . . . . .	535
7.136.2.2 Point4() [2/2] . . . . .	536
7.137PreRenderable< T > Class Template Reference . . . . .	536
7.137.1 Detailed Description . . . . .	536
7.137.2 Constructor & Destructor Documentation . . . . .	537
7.137.2.1 PreRenderable() . . . . .	537
7.137.3 Member Function Documentation . . . . .	537
7.137.3.1 draw() . . . . .	537
7.137.3.2 isPreRendered() . . . . .	537
7.137.3.3 preRender() . . . . .	537
7.137.3.4 setupDrawChain() . . . . .	537
7.138Presenter Class Reference . . . . .	538
7.138.1 Detailed Description . . . . .	538
7.138.2 Constructor & Destructor Documentation . . . . .	538
7.138.2.1 ~Presenter() . . . . .	538
7.138.2.2 Presenter() . . . . .	538
7.138.3 Member Function Documentation . . . . .	538
7.138.3.1 activate() . . . . .	539
7.138.3.2 deactivate() . . . . .	539
7.139CWRUtil::Q10 Class Reference . . . . .	539
7.139.1 Detailed Description . . . . .	539
7.139.2 Constructor & Destructor Documentation . . . . .	539
7.139.2.1 Q10() [1/2] . . . . .	540
7.139.2.2 Q10() [2/2] . . . . .	540
7.139.3 Member Function Documentation . . . . .	540
7.139.3.1 operator int() . . . . .	540
7.139.3.2 operator*() . . . . .	540
7.139.3.3 operator+() . . . . .	540
7.139.3.4 operator-() . . . . .	541
7.139.3.5 operator/() . . . . .	541
7.140CWRUtil::Q15 Class Reference . . . . .	541
7.140.1 Detailed Description . . . . .	542
7.140.2 Constructor & Destructor Documentation . . . . .	542
7.140.2.1 Q15() . . . . .	542
7.140.3 Member Function Documentation . . . . .	542
7.140.3.1 operator int() . . . . .	542

7.140.3.2 operator+()	542
7.140.3.3 operator-()	543
7.140.3.4 operator/()	543
7.141 CWRUtil::Q5 Class Reference	543
7.141.1 Detailed Description	544
7.141.2 Constructor & Destructor Documentation	544
7.141.2.1 Q5() [1/2]	544
7.141.2.2 Q5() [2/2]	545
7.141.3 Member Function Documentation	545
7.141.3.1 operator int()	545
7.141.3.2 operator*() [1/3]	545
7.141.3.3 operator*() [2/3]	545
7.141.3.4 operator*() [3/3]	546
7.141.3.5 operator+()	546
7.141.3.6 operator-() [1/2]	546
7.141.3.7 operator-() [2/2]	547
7.141.3.8 operator/() [1/2]	547
7.141.3.9 operator/() [2/2]	547
7.141.3.10 to()	548
7.142 Quadruple Class Reference	548
7.142.1 Detailed Description	549
7.142.2 Constructor & Destructor Documentation	549
7.142.2.1 Quadruple() [1/2]	549
7.142.2.2 Quadruple() [2/2]	549
7.142.3 Member Function Documentation	549
7.142.3.1 getElement()	549
7.142.3.2 getW()	550
7.142.3.3 getX()	550
7.142.3.4 getY()	550
7.142.3.5 getZ()	550
7.142.3.6 setElement()	551
7.142.3.7 setW()	551
7.142.3.8 setX()	551
7.142.3.9 setY()	551
7.142.3.10 setZ()	552
7.143 RadioButton Class Reference	552
7.143.1 Detailed Description	553
7.143.2 Constructor & Destructor Documentation	553
7.143.2.1 RadioButton()	553
7.143.2.2 ~RadioButton()	553

7.143.3 Member Function Documentation	553
7.143.3.1 draw()	554
7.143.3.2 getAlpha()	554
7.143.3.3 getCurrentlyDisplayedBitmap()	554
7.143.3.4 getDeselectionEnabled()	554
7.143.3.5 getSelected()	555
7.143.3.6 getSolidRect()	555
7.143.3.7 getType()	555
7.143.3.8 handleClickEvent()	555
7.143.3.9 setAlpha()	556
7.143.3.10setBitmaps()	556
7.143.3.11setDeselectedAction()	556
7.143.3.12setDeselectionEnabled()	556
7.143.3.13setSelected()	558
7.144 RadioButtonGroup< CAPACITY > Class Template Reference	558
7.144.1 Detailed Description	559
7.144.2 Constructor & Destructor Documentation	559
7.144.2.1 RadioButtonGroup()	559
7.144.2.2 ~RadioButtonGroup()	560
7.144.3 Member Function Documentation	560
7.144.3.1 add()	560
7.144.3.2 getDeselectionEnabled()	560
7.144.3.3 getRadioButton()	560
7.144.3.4 getSelectedRadioButton()	560
7.144.3.5 getSelectedRadioButtonIndex()	561
7.144.3.6 radioButtonClickedHandler()	561
7.144.3.7 radioButtonDeselectedHandler()	561
7.144.3.8 setDeselectionEnabled()	561
7.144.3.9 setRadioButtonDeselectedHandler()	562
7.144.3.10setRadioButtonSelectedHandler()	562
7.144.3.11setSelected()	562
7.145 Rasterizer Class Reference	563
7.145.1 Detailed Description	563
7.145.2 Member Enumeration Documentation	564
7.145.2.1 anonymous enum	564
7.145.2.2 anonymous enum	564
7.145.2.3 FillingRule	564
7.145.3 Constructor & Destructor Documentation	565
7.145.3.1 Rasterizer()	565
7.145.4 Member Function Documentation	565



7.145.4.1 calculateAlpha()	565
7.145.4.2 lineTo()	565
7.145.4.3 moveTo()	565
7.145.4.4 render()	566
7.145.4.5 reset()	566
7.145.4.6 setFillingRule()	566
7.145.4.7 setMaxRenderY()	566
7.145.4.8 wasOutlineTooComplex()	567
7.146Rect Class Reference	567
7.146.1 Detailed Description	568
7.146.2 Constructor & Destructor Documentation	568
7.146.2.1 Rect() [1/2]	568
7.146.2.2 Rect() [2/2]	568
7.146.3 Member Function Documentation	568
7.146.3.1 area()	569
7.146.3.2 bottom()	569
7.146.3.3 expandToFit()	569
7.146.3.4 includes()	569
7.146.3.5 intersect() [1/2]	569
7.146.3.6 intersect() [2/2]	570
7.146.3.7 isEmpty()	570
7.146.3.8 operator!=(())	570
7.146.3.9 operator&()	571
7.146.3.10operator&=()	571
7.146.3.11operator==()	571
7.146.3.12right()	572
7.147Renderer Class Reference	572
7.147.1 Detailed Description	572
7.147.2 Constructor & Destructor Documentation	572
7.147.2.1 Renderer() [1/2]	572
7.147.2.2 Renderer() [2/2]	572
7.147.3 Member Function Documentation	573
7.147.3.1 getRenderingBuffer()	573
7.147.3.2 render()	573
7.147.3.3 setRenderingBuffer()	573
7.148RenderingBuffer Class Reference	573
7.148.1 Detailed Description	574
7.148.2 Constructor & Destructor Documentation	575
7.148.2.1 RenderingBuffer() [1/2]	575
7.148.2.2 ~RenderingBuffer()	575

7.148.2.3 RenderingBuffer() [2/2]	575
7.148.3 Member Function Documentation	575
7.148.3.1 attach()	575
7.148.3.2 getHeight()	576
7.148.3.3 getWidth()	576
7.148.3.4 getXAdjust()	576
7.148.3.5 inbox()	576
7.148.3.6 row() [1/2]	577
7.148.3.7 row() [2/2]	577
7.149 RepeatButton Class Reference	577
7.149.1 Detailed Description	578
7.149.2 Constructor & Destructor Documentation	578
7.149.2.1 RepeatButton()	578
7.149.3 Member Function Documentation	578
7.149.3.1 getDelay()	578
7.149.3.2 getInterval()	579
7.149.3.3 handleClickEvent()	579
7.149.3.4 handleTickEvent()	579
7.149.3.5 setDelay()	579
7.149.3.6 setInterval()	580
7.150 RepeatButtonTrigger Class Reference	580
7.150.1 Detailed Description	581
7.150.2 Member Function Documentation	581
7.150.2.1 getDelay()	581
7.150.2.2 getInterval()	581
7.150.2.3 handleClickEvent()	581
7.150.2.4 setDelay()	581
7.150.2.5 setInterval()	582
7.151 ScalableImage Class Reference	582
7.151.1 Detailed Description	583
7.151.2 Member Enumeration Documentation	583
7.151.2.1 ScalingAlgorithm	583
7.151.3 Constructor & Destructor Documentation	583
7.151.3.1 ScalableImage()	583
7.151.3.2 ~ScalableImage()	583
7.151.4 Member Function Documentation	584
7.151.4.1 draw()	584
7.151.4.2 drawTriangle()	584
7.151.4.3 getAlpha()	584
7.151.4.4 getBitmap()	585

7.151.4.5 getScalingAlgorithm()	585
7.151.4.6 getSolidRect()	585
7.151.4.7 getType()	585
7.151.4.8 lookupRenderVariant()	586
7.151.4.9 setAlpha()	586
7.151.4.10setBitmap()	586
7.151.4.11setScalingAlgorithm()	586
7.152Scanline Class Reference	586
7.152.1 Detailed Description	587
7.152.2 Constructor & Destructor Documentation	588
7.152.2.1 Scanline()	588
7.152.2.2 ~Scanline()	588
7.152.3 Member Function Documentation	588
7.152.3.1 addCell()	588
7.152.3.2 addSpan()	589
7.152.3.3 getNumSpans()	589
7.152.3.4 getY()	589
7.152.3.5 isReady()	589
7.152.3.6 reset()	590
7.152.3.7 resetSpans()	590
7.153Screen Class Reference	590
7.153.1 Detailed Description	591
7.153.2 Constructor & Destructor Documentation	591
7.153.2.1 Screen()	591
7.153.2.2 ~Screen()	591
7.153.3 Member Function Documentation	592
7.153.3.1 add()	592
7.153.3.2 afterTransition()	592
7.153.3.3 bindTransition()	592
7.153.3.4 draw() [1/2]	592
7.153.3.5 draw() [2/2]	593
7.153.3.6 getRootContainer()	593
7.153.3.7 handleClickEvent()	593
7.153.3.8 handleDragEvent()	593
7.153.3.9 handleGestureEvent()	594
7.153.3.10handleKeyEvent()	594
7.153.3.11handleTickEvent()	594
7.153.3.12JSMOC()	594
7.153.3.13remove()	595
7.153.3.14setupScreen()	595

7.153.3.15startSMOC()	595
7.153.3.16gearDownScreen()	595
7.153.3.17useSMOCDrawing()	595
7.153.3.18usingSMOC()	596
7.154ScrollableContainer Class Reference	596
7.154.1 Detailed Description	599
7.154.2 Constructor & Destructor Documentation	599
7.154.2.1 ScrollableContainer()	599
7.154.2.2 ~ScrollableContainer()	599
7.154.3 Member Function Documentation	599
7.154.3.1 add()	599
7.154.3.2 childGeometryChanged()	600
7.154.3.3 doScroll()	600
7.154.3.4 enableHorizontalScroll()	600
7.154.3.5 enableVerticalScroll()	601
7.154.3.6 getContainedArea()	601
7.154.3.7 getLastChild()	601
7.154.3.8 getScrolledX()	601
7.154.3.9 getScrolledY()	602
7.154.3.10getType()	602
7.154.3.11getXBorder()	602
7.154.3.12getXScrollbar()	602
7.154.3.13getYBorder()	603
7.154.3.14getYScrollbar()	603
7.154.3.15handleClickEvent()	603
7.154.3.16handleDragEvent()	603
7.154.3.17handleGestureEvent()	604
7.154.3.18handleTickEvent()	604
7.154.3.19invalidateScrollbars()	604
7.154.3.20isScrollableXY()	604
7.154.3.21moveChildrenRelative()	605
7.154.3.22reset()	605
7.154.3.23setMaxVelocity()	605
7.154.3.24setScrollbarPadding()	605
7.154.3.25setScrollbarsAlpha()	605
7.154.3.26setScrollbarsColor()	606
7.154.3.27setScrollbarsPermanentlyVisible()	606
7.154.3.28setScrollbarsVisible()	606
7.154.3.29setScrollbarWidth()	606
7.154.3.30setScrollThreshold()	607

7.155 ScrollBase Class Reference . . . . .	607
7.155.1 Detailed Description . . . . .	610
7.155.2 Member Enumeration Documentation . . . . .	610
7.155.2.1 AnimationState . . . . .	610
7.155.3 Constructor & Destructor Documentation . . . . .	610
7.155.3.1 ScrollBase() . . . . .	610
7.155.3.2 ~ScrollBase() . . . . .	611
7.155.4 Member Function Documentation . . . . .	611
7.155.4.1 allowHorizontalDrag() . . . . .	611
7.155.4.2 allowVerticalDrag() . . . . .	611
7.155.4.3 animateToItem() . . . . .	611
7.155.4.4 animateToPosition() . . . . .	612
7.155.4.5 getAnimationSteps() . . . . .	612
7.155.4.6 getCircular() . . . . .	612
7.155.4.7 getDragAcceleration() . . . . .	613
7.155.4.8 getDrawableMargin() . . . . .	613
7.155.4.9 getDrawableSize() . . . . .	613
7.155.4.10 getHorizontal() . . . . .	613
7.155.4.11 getMaxSwipeItems() . . . . .	614
7.155.4.12 getNearestAlignedOffset() . . . . .	614
7.155.4.13 getNormalizedOffset() . . . . .	614
7.155.4.14 getNumberOfItems() . . . . .	614
7.155.4.15 getOffset() . . . . .	615
7.155.4.16 getPositionForItem() . . . . .	615
7.155.4.17 getSwipeAcceleration() . . . . .	615
7.155.4.18 handleDragEvent() . . . . .	616
7.155.4.19 handleGestureEvent() . . . . .	616
7.155.4.20 handleTickEvent() . . . . .	616
7.155.4.21 initialize() . . . . .	616
7.155.4.22 isAnimating() . . . . .	617
7.155.4.23 itemChanged() . . . . .	617
7.155.4.24 keepOffsetInsideLimits() . . . . .	617
7.155.4.25 setAnimationEndedCallback() . . . . .	617
7.155.4.26 setAnimationSteps() . . . . .	618
7.155.4.27 setCircular() . . . . .	618
7.155.4.28 setDragAcceleration() . . . . .	618
7.155.4.29 setDrawableSize() . . . . .	619
7.155.4.30 setEasingEquation() . . . . .	619
7.155.4.31 setHeight() . . . . .	619
7.155.4.32 setHorizontal() . . . . .	620

7.155.4.33	setItemPressedCallback()	620
7.155.4.34	setItemSelectedCallback()	620
7.155.4.35	setMaxSwipeItems()	621
7.155.4.36	setNumberOfItems()	621
7.155.4.37	setOffset()	621
7.155.4.38	setSwipeAcceleration()	621
7.155.4.39	setWidth()	622
7.155.4.40	stopAnimation()	622
7.156	ScrollList Class Reference	622
7.156.1	Detailed Description	623
7.156.2	Constructor & Destructor Documentation	623
7.156.2.1	ScrollList()	624
7.156.2.2	~ScrollList()	624
7.156.3	Member Function Documentation	624
7.156.3.1	getItem()	624
7.156.3.2	getNearestAlignedOffset()	624
7.156.3.3	getPaddingAfter()	624
7.156.3.4	getPaddingBefore()	625
7.156.3.5	getPositionForItem()	625
7.156.3.6	getSnapping()	625
7.156.3.7	handleClickEvent()	626
7.156.3.8	keepOffsetInsideLimits()	626
7.156.3.9	setDrawables()	626
7.156.3.10	setPadding()	627
7.156.3.11	setSnapping()	627
7.156.3.12	setWindowSize()	627
7.157	ScrollWheel Class Reference	628
7.157.1	Detailed Description	628
7.157.2	Constructor & Destructor Documentation	628
7.157.2.1	ScrollWheel()	628
7.157.2.2	~ScrollWheel()	628
7.157.3	Member Function Documentation	628
7.157.3.1	setDrawables()	629
7.158	ScrollWheelBase Class Reference	629
7.158.1	Detailed Description	630
7.158.2	Constructor & Destructor Documentation	630
7.158.2.1	ScrollWheelBase()	630
7.158.2.2	~ScrollWheelBase()	630
7.158.3	Member Function Documentation	630
7.158.3.1	animateToPosition()	630

7.158.3.2 getPositionForItem()	631
7.158.3.3 getSelectedItem()	631
7.158.3.4 getSelectedItemOffset()	631
7.158.3.5 handleClickEvent()	631
7.158.3.6 handleDragEvent()	632
7.158.3.7 handleGestureEvent()	632
7.158.3.8 keepOffsetInsideLimits()	632
7.158.3.9 setAnimateToCallback()	632
7.158.3.10 setSelectedItemOffset()	633
7.159 ScrollWheelWithSelectionStyle Class Reference	633
7.159.1 Detailed Description	635
7.159.2 Constructor & Destructor Documentation	635
7.159.2.1 ScrollWheelWithSelectionStyle()	635
7.159.2.2 ~ScrollWheelWithSelectionStyle()	635
7.159.3 Member Function Documentation	635
7.159.3.1 getSelectedItemExtraSizeAfter()	635
7.159.3.2 getSelectedItemExtraSizeBefore()	635
7.159.3.3 getSelectedItemMarginAfter()	636
7.159.3.4 getSelectedItemMarginBefore()	636
7.159.3.5 initialize()	636
7.159.3.6 itemChanged()	636
7.159.3.7 refreshDrawableListsLayout()	637
7.159.3.8 setCircular()	637
7.159.3.9 setDrawables()	637
7.159.3.10 setDrawableSize()	638
7.159.3.11 setHeight()	638
7.159.3.12 setHorizontal()	638
7.159.3.13 setNumberOfItems()	639
7.159.3.14 setOffset()	639
7.159.3.15 setSelectedItemExtraSize()	639
7.159.3.16 setSelectedItemMargin()	640
7.159.3.17 setSelectedItemPosition()	640
7.159.3.18 setSelectedItemOffset()	641
7.159.3.19 setWidth()	641
7.160 SDL2TouchController Class Reference	641
7.160.1 Detailed Description	641
7.160.2 Member Function Documentation	641
7.160.2.1 init()	642
7.160.2.2 sampleTouch()	642
7.161 SDLTouchController Class Reference	642

7.161.1 Detailed Description . . . . .	642
7.161.2 Member Function Documentation . . . . .	642
7.161.2.1 init() . . . . .	643
7.161.2.2 sampleTouch() . . . . .	643
7.162Shape< POINTS > Class Template Reference . . . . .	643
7.162.1 Detailed Description . . . . .	644
7.162.2 Constructor & Destructor Documentation . . . . .	644
7.162.2.1 ~Shape() . . . . .	644
7.162.3 Member Function Documentation . . . . .	644
7.162.3.1 getCacheX() . . . . .	644
7.162.3.2 getCacheY() . . . . .	645
7.162.3.3 getCornerX() . . . . .	645
7.162.3.4 getCornerY() . . . . .	645
7.162.3.5 getNumPoints() . . . . .	646
7.162.3.6 setCache() . . . . .	646
7.162.3.7 setCorner() . . . . .	646
7.163AbstractShape::ShapePoint< T > Struct Template Reference . . . . .	647
7.163.1 Detailed Description . . . . .	647
7.164SlideMenu Class Reference . . . . .	647
7.164.1 Detailed Description . . . . .	650
7.164.2 Member Function Documentation . . . . .	650
7.164.2.1 add() . . . . .	650
7.164.2.2 animateToState() . . . . .	650
7.164.2.3 animationEndedHandler() . . . . .	651
7.164.2.4 getAnimationDuration() . . . . .	651
7.164.2.5 getAnimationEasingEquation() . . . . .	651
7.164.2.6 getBackgroundX() . . . . .	651
7.164.2.7 getBackgroundY() . . . . .	651
7.164.2.8 getCollapsedXCoordinate() . . . . .	651
7.164.2.9 getCollapsedYCoordinate() . . . . .	652
7.164.2.10getExpandDirection() . . . . .	652
7.164.2.11getExpandedStateTimeout() . . . . .	652
7.164.2.12getExpandedStateTimer() . . . . .	652
7.164.2.13getExpandedXCoordinate() . . . . .	652
7.164.2.14getExpandedYCoordinate() . . . . .	653
7.164.2.15getHiddenPixelsWhenExpanded() . . . . .	653
7.164.2.16getState() . . . . .	653
7.164.2.17getStateChangeButtonX() . . . . .	653
7.164.2.18getStateChangeButtonY() . . . . .	653
7.164.2.19getVisiblePixelsWhenCollapsed() . . . . .	653



7.164.2.20	remove()	654
7.164.2.21	resetExpandedStateTimer()	654
7.164.2.22	setAnimationDuration()	654
7.164.2.23	setAnimationEasingEquation()	654
7.164.2.24	setExpandDirection()	654
7.164.2.25	setExpandedStateTimeout()	655
7.164.2.26	setHiddenPixelsWhenExpanded()	655
7.164.2.27	setState()	655
7.164.2.28	setStateChangedAnimationEndedCallback()	655
7.164.2.29	setStateChangedCallback()	655
7.164.2.30	setup() [1/2]	656
7.164.2.31	setup() [2/2]	656
7.164.2.32	setVisiblePixelsWhenCollapsed()	657
7.164.2.33	stateChangeButtonClickedHandler()	657
7.165	Slider Class Reference	657
7.165.1	Detailed Description	659
7.165.2	Constructor & Destructor Documentation	659
7.165.2.1	Slider()	659
7.165.3	Member Function Documentation	660
7.165.3.1	getIndicatorMax()	660
7.165.3.2	getIndicatorMin()	660
7.165.3.3	getIndicatorPositionRangeSize()	660
7.165.3.4	getIndicatorRadius()	660
7.165.3.5	getMaxValue()	661
7.165.3.6	getMinValue()	661
7.165.3.7	getType()	661
7.165.3.8	getValue()	661
7.165.3.9	getValueRangeSize()	662
7.165.3.10	handleClickEvent()	662
7.165.3.11	handleDragEvent()	662
7.165.3.12	positionToValue()	662
7.165.3.13	setBitmaps() [1/2]	663
7.165.3.14	setBitmaps() [2/2]	663
7.165.3.15	setNewValueCallback()	663
7.165.3.16	setStartValueCallback()	664
7.165.3.17	setStopValueCallback()	664
7.165.3.18	setupHorizontalSlider()	664
7.165.3.19	setupVerticalSlider()	665
7.165.3.20	setValue()	665
7.165.3.21	setValueRange() [1/2]	666

7.165.3.22 setValueRange() [2/2]	666
7.165.3.23 updateIndicatorPosition()	667
7.165.3.24 valueToPosition()	667
7.166 SlideTransition< templateDirection > Class Template Reference	667
7.166.1 Detailed Description	668
7.166.2 Constructor & Destructor Documentation	668
7.166.2.1 SlideTransition()	668
7.166.2.2 ~SlideTransition()	668
7.166.3 Member Function Documentation	668
7.166.3.1 handleTickEvent()	669
7.166.3.2 init()	669
7.166.3.3 initMoveDrawable()	669
7.166.3.4 tearDown()	669
7.166.3.5 tickMoveDrawable()	669
7.167 Snapper< T > Class Template Reference	670
7.167.1 Detailed Description	670
7.167.2 Constructor & Destructor Documentation	670
7.167.2.1 Snapper()	670
7.167.2.2 ~Snapper()	671
7.167.3 Member Function Documentation	671
7.167.3.1 handleClickEvent()	671
7.167.3.2 handleDragEvent()	671
7.167.3.3 setDragAction()	671
7.167.3.4 setSnappedAction()	672
7.167.3.5 setSnapPosition()	672
7.168 SnapshotWidget Class Reference	672
7.168.1 Detailed Description	673
7.168.2 Constructor & Destructor Documentation	673
7.168.2.1 SnapshotWidget()	673
7.168.2.2 ~SnapshotWidget()	673
7.168.3 Member Function Documentation	673
7.168.3.1 draw()	674
7.168.3.2 getAlpha()	674
7.168.3.3 getSolidRect()	674
7.168.3.4 getType()	674
7.168.3.5 makeSnapshot() [1/2]	674
7.168.3.6 makeSnapshot() [2/2]	675
7.168.3.7 setAlpha()	675
7.169 LCD::StringVisuals Struct Reference	675
7.169.1 Detailed Description	676

7.169.2 Constructor & Destructor Documentation . . . . .	676
7.169.2.1 StringVisuals() [1/2] . . . . .	676
7.169.2.2 StringVisuals() [2/2] . . . . .	676
7.170SwipeContainer Class Reference . . . . .	677
7.170.1 Detailed Description . . . . .	677
7.170.2 Member Function Documentation . . . . .	677
7.170.2.1 add() . . . . .	678
7.170.2.2 getNumberOfPages() . . . . .	678
7.170.2.3 handleClickEvent() . . . . .	678
7.170.2.4 handleDragEvent() . . . . .	678
7.170.2.5 handleGestureEvent() . . . . .	679
7.170.2.6 handleTickEvent() . . . . .	679
7.170.2.7 remove() . . . . .	679
7.170.2.8 setEndSwipeElasticWidth() . . . . .	679
7.170.2.9 setPageIndicatorBitmaps() . . . . .	680
7.170.2.10setPageIndicatorXY() . . . . .	680
7.170.2.11setPageIndicatorXYWithCenteredX() . . . . .	680
7.170.2.12setSelectedPage() . . . . .	681
7.170.2.13setSwipeCutoff() . . . . .	681
7.171TextArea Class Reference . . . . .	681
7.171.1 Detailed Description . . . . .	683
7.171.2 Constructor & Destructor Documentation . . . . .	683
7.171.2.1 TextArea() . . . . .	683
7.171.3 Member Function Documentation . . . . .	683
7.171.3.1 draw() . . . . .	683
7.171.3.2 getAlpha() . . . . .	683
7.171.3.3 getColor() . . . . .	684
7.171.3.4 getIndentation() . . . . .	684
7.171.3.5 getLinespacing() . . . . .	684
7.171.3.6 getRotation() . . . . .	684
7.171.3.7 getSolidRect() . . . . .	685
7.171.3.8 getTextHeight() . . . . .	685
7.171.3.9 getTextWidth() . . . . .	685
7.171.3.10getType() . . . . .	685
7.171.3.11getTypedText() . . . . .	686
7.171.3.12getWideTextAction() . . . . .	686
7.171.3.13resizeHeightToCurrentText() . . . . .	686
7.171.3.14resizeToCurrentText() . . . . .	686
7.171.3.15setAlpha() . . . . .	687
7.171.3.16setBaselineY() . . . . .	687

7.171.3.17	setColor()	687
7.171.3.18	setIndentation()	687
7.171.3.19	setLinespacing()	688
7.171.3.20	setRotation()	688
7.171.3.21	setTypedText()	688
7.171.3.22	setWideTextAction()	689
7.171.3.23	setXBaselineY()	689
7.172	TextAreaWithOneWildcard Class Reference	689
7.172.1	Detailed Description	690
7.172.2	Constructor & Destructor Documentation	690
7.172.2.1	TextAreaWithOneWildcard()	690
7.172.3	Member Function Documentation	690
7.172.3.1	draw()	690
7.172.3.2	getTextHeight()	691
7.172.3.3	getTextWidth()	691
7.172.3.4	getType()	691
7.172.3.5	getWildcard()	691
7.172.3.6	setWildcard()	692
7.173	TextAreaWithTwoWildcards Class Reference	692
7.173.1	Detailed Description	692
7.173.2	Constructor & Destructor Documentation	693
7.173.2.1	TextAreaWithTwoWildcards()	693
7.173.3	Member Function Documentation	693
7.173.3.1	draw()	693
7.173.3.2	getTextHeight()	693
7.173.3.3	getTextWidth()	693
7.173.3.4	getType()	694
7.173.3.5	getWildcard1()	694
7.173.3.6	getWildcard2()	694
7.173.3.7	setWildcard1()	694
7.173.3.8	setWildcard2()	695
7.174	TextAreaWithWildcardBase Class Reference	695
7.174.1	Detailed Description	695
7.174.2	Constructor & Destructor Documentation	695
7.174.2.1	TextAreaWithWildcardBase()	695
7.174.3	Member Function Documentation	695
7.174.3.1	calculateTextHeight()	696
7.175	TextButtonStyle< T > Class Template Reference	696
7.175.1	Detailed Description	697
7.175.2	Member Function Documentation	697

7.175.2.1 setText()	697
7.175.2.2 setTextColors()	697
7.175.2.3 setTextPosition()	698
7.175.2.4 setTextRotation()	698
7.175.2.5 setTextX()	698
7.175.2.6 setTextXY()	698
7.175.2.7 setTextY()	699
7.176TextProgress Class Reference	699
7.176.1 Detailed Description	700
7.176.2 Constructor & Destructor Documentation	700
7.176.2.1 TextProgress()	700
7.176.2.2 ~TextProgress()	700
7.176.3 Member Function Documentation	700
7.176.3.1 getAlpha()	700
7.176.3.2 getColor()	701
7.176.3.3 getNumberOfDecimals()	701
7.176.3.4 getTypedText()	701
7.176.3.5 setAlpha()	701
7.176.3.6 setColor()	702
7.176.3.7 setNumberOfDecimals()	702
7.176.3.8 setProgressIndicatorPosition()	702
7.176.3.9 setTypedText()	703
7.176.3.10setValue()	703
7.177TextProvider Class Reference	703
7.177.1 Detailed Description	704
7.177.2 Constructor & Destructor Documentation	704
7.177.2.1 TextProvider()	704
7.177.3 Member Function Documentation	704
7.177.3.1 getNextChar()	704
7.177.3.2 getNextLigature()	704
7.177.3.3 initialize()	705
7.178Texts Class Reference	705
7.178.1 Detailed Description	706
7.178.2 Member Function Documentation	706
7.178.2.1 getLanguage()	706
7.178.2.2 getText()	706
7.178.2.3 setLanguage()	706
7.179TextureMapper Class Reference	706
7.179.1 Detailed Description	710
7.179.2 Member Enumeration Documentation	710

7.179.2.1 RenderingAlgorithm . . . . .	711
7.179.3 Constructor & Destructor Documentation . . . . .	711
7.179.3.1 ~TextureMapper() . . . . .	711
7.179.4 Member Function Documentation . . . . .	711
7.179.4.1 applyTransformation() . . . . .	711
7.179.4.2 draw() . . . . .	711
7.179.4.3 drawTriangle() . . . . .	711
7.179.4.4 getAlpha() . . . . .	712
7.179.4.5 getBitmap() . . . . .	712
7.179.4.6 getBitmapPositionX() . . . . .	712
7.179.4.7 getBitmapPositionY() . . . . .	713
7.179.4.8 getBoundingRect() . . . . .	713
7.179.4.9 getCameraDistance() . . . . .	713
7.179.4.10 getCameraX() . . . . .	713
7.179.4.11 getCameraY() . . . . .	713
7.179.4.12 getOrigoX() . . . . .	714
7.179.4.13 getOrigoY() . . . . .	714
7.179.4.14 getOrigoZ() . . . . .	714
7.179.4.15 getRenderingAlgorithm() . . . . .	714
7.179.4.16 getScale() . . . . .	714
7.179.4.17 getSolidRect() . . . . .	715
7.179.4.18 getType() . . . . .	715
7.179.4.19 getX0() . . . . .	715
7.179.4.20 getX1() . . . . .	715
7.179.4.21 getX2() . . . . .	716
7.179.4.22 getX3() . . . . .	716
7.179.4.23 getXAngle() . . . . .	716
7.179.4.24 getY0() . . . . .	716
7.179.4.25 getY1() . . . . .	716
7.179.4.26 getY2() . . . . .	717
7.179.4.27 getY3() . . . . .	717
7.179.4.28 getYAngle() . . . . .	717
7.179.4.29 getZ0() . . . . .	717
7.179.4.30 getZ1() . . . . .	717
7.179.4.31 getZ2() . . . . .	718
7.179.4.32 getZ3() . . . . .	718
7.179.4.33 getZAngle() . . . . .	718
7.179.4.34 lookupRenderVariant() . . . . .	718
7.179.4.35 setAlpha() . . . . .	718
7.179.4.36 setBitmap() . . . . .	719

7.179.4.37	setBitmapPosition() [1/2]	719
7.179.4.38	setBitmapPosition() [2/2]	719
7.179.4.39	setCamera()	719
7.179.4.40	setCameraDistance()	720
7.179.4.41	setOrigo() [1/2]	720
7.179.4.42	setOrigo() [2/2]	720
7.179.4.43	setRenderingAlgorithm()	721
7.179.4.44	setScale()	721
7.179.4.45	updateAngles()	721
7.179.4.46	updateXAngle()	721
7.179.4.47	updateYAngle()	722
7.179.4.48	updateZAngle()	722
7.180	TextureSurface Struct Reference	722
7.181	TiledImage Class Reference	722
7.181.1	Detailed Description	723
7.181.2	Constructor & Destructor Documentation	723
7.181.2.1	TiledImage()	724
7.181.3	Member Function Documentation	725
7.181.3.1	draw()	725
7.181.3.2	getOffset()	725
7.181.3.3	getSolidRect()	725
7.181.3.4	getType()	726
7.181.3.5	getXOffset()	726
7.181.3.6	getYOffset()	726
7.181.3.7	setBitmap()	726
7.181.3.8	setOffset()	727
7.181.3.9	setXOffset()	727
7.181.3.10	setYOffset()	727
7.182	TiledImageButtonStyle< T > Class Template Reference	729
7.182.1	Detailed Description	730
7.182.2	Member Function Documentation	730
7.182.2.1	setHeight()	730
7.182.2.2	setTileBitmaps()	730
7.182.2.3	setTileOffset()	730
7.182.2.4	setWidth()	731
7.183	ToggleButton Class Reference	731
7.183.1	Detailed Description	731
7.183.2	Constructor & Destructor Documentation	732
7.183.2.1	ToggleButton()	732
7.183.3	Member Function Documentation	732

7.183.3.1 forceState()	732
7.183.3.2 getState()	732
7.183.3.3 getType()	732
7.183.3.4 handleClickEvent()	733
7.183.3.5 setBitmaps()	733
7.184ToggleButtonTrigger Class Reference	733
7.184.1 Detailed Description	734
7.184.2 Member Function Documentation	734
7.184.2.1 forceState()	734
7.184.2.2 getToggleCanceled()	734
7.184.2.3 handleClickEvent()	734
7.184.2.4 setToggleCanceled()	735
7.185TouchArea Class Reference	735
7.185.1 Detailed Description	735
7.185.2 Constructor & Destructor Documentation	736
7.185.2.1 TouchArea()	736
7.185.3 Member Function Documentation	736
7.185.3.1 draw()	736
7.185.3.2 getSolidRect()	736
7.185.3.3 getType()	736
7.185.3.4 handleClickEvent()	737
7.185.3.5 handleDragEvent()	737
7.185.3.6 setPressedAction()	737
7.186TouchButtonTrigger Class Reference	737
7.186.1 Detailed Description	738
7.186.2 Member Function Documentation	738
7.186.2.1 handleClickEvent()	738
7.187TouchCalibration Class Reference	738
7.187.1 Detailed Description	738
7.187.2 Member Function Documentation	739
7.187.2.1 setCalibrationMatrix()	739
7.187.2.2 translatePoint()	739
7.188TouchController Class Reference	739
7.188.1 Detailed Description	739
7.188.2 Constructor & Destructor Documentation	739
7.188.2.1 ~TouchController()	740
7.188.3 Member Function Documentation	740
7.188.3.1 init()	740
7.188.3.2 sampleTouch()	740
7.189Transition Class Reference	740



7.189.1 Detailed Description . . . . .	741
7.189.2 Constructor & Destructor Documentation . . . . .	741
7.189.2.1 Transition() . . . . .	741
7.189.2.2 ~Transition() . . . . .	741
7.189.3 Member Function Documentation . . . . .	741
7.189.3.1 handleTickEvent() . . . . .	741
7.189.3.2 init() . . . . .	742
7.189.3.3 isDone() . . . . .	742
7.189.3.4 setScreenContainer() . . . . .	742
7.189.3.5 tearDown() . . . . .	742
7.190 TwoWildcardTextButtonStyle< T > Class Template Reference . . . . .	742
7.190.1 Detailed Description . . . . .	743
7.190.2 Member Function Documentation . . . . .	744
7.190.2.1 setTwoWildcardText() . . . . .	744
7.190.2.2 setTwoWildcardTextColors() . . . . .	744
7.190.2.3 setTwoWildcardTextPosition() . . . . .	744
7.190.2.4 setTwoWildcardTextRotation() . . . . .	745
7.190.2.5 setTwoWildcardTextX() . . . . .	745
7.190.2.6 setTwoWildcardTextXY() . . . . .	745
7.190.2.7 setTwoWildcardTextY() . . . . .	745
7.190.2.8 setWildcardTextBuffer1() . . . . .	746
7.190.2.9 setWildcardTextBuffer2() . . . . .	746
7.191 TypedText Class Reference . . . . .	746
7.191.1 Detailed Description . . . . .	747
7.191.2 Constructor & Destructor Documentation . . . . .	747
7.191.2.1 TypedText() . . . . .	747
7.191.3 Member Function Documentation . . . . .	747
7.191.3.1 getAlignment() . . . . .	747
7.191.3.2 getFont() . . . . .	748
7.191.3.3 getId() . . . . .	748
7.191.3.4 getText() . . . . .	748
7.191.3.5 getTextDirection() . . . . .	748
7.191.3.6 hasValidId() . . . . .	748
7.191.3.7 registerTexts() . . . . .	749
7.191.3.8 registerTypedTextDatabase() . . . . .	749
7.192 TypedText::TypedTextData Struct Reference . . . . .	749
7.192.1 Detailed Description . . . . .	749
7.193 UIEventListener Class Reference . . . . .	750
7.193.1 Detailed Description . . . . .	750
7.193.2 Constructor & Destructor Documentation . . . . .	750

7.193.2.1 ~UIEventListener()	750
7.193.3 Member Function Documentation	750
7.193.3.1 handleClickEvent()	750
7.193.3.2 handleDragEvent()	751
7.193.3.3 handleGestureEvent()	751
7.193.3.4 handleKeyEvent()	751
7.193.3.5 handlePendingScreenTransition()	751
7.193.3.6 handleTickEvent()	752
7.194 Unicode Class Reference	752
7.194.1 Member Typedef Documentation	753
7.194.1.1 UnicodeChar	753
7.194.2 Member Function Documentation	753
7.194.2.1 atoi()	753
7.194.2.2 fromUTF8()	754
7.194.2.3 itoa()	754
7.194.2.4 snprintf() [1/2]	754
7.194.2.5 snprintf() [2/2]	755
7.194.2.6 snprintfFloat() [1/2]	756
7.194.2.7 snprintfFloat() [2/2]	757
7.194.2.8 snprintfFloats() [1/2]	758
7.194.2.9 snprintfFloats() [2/2]	760
7.194.2.10 strlen() [1/2]	761
7.194.2.11 strlen() [2/2]	761
7.194.2.12 strncmp()	761
7.194.2.13 strncmp_ignore_white_spaces()	762
7.194.2.14 strncpy() [1/2]	762
7.194.2.15 strncpy() [2/2]	763
7.194.2.16 toUTF8()	763
7.194.2.17 toa()	764
7.194.2.18 vsnprintf() [1/2]	764
7.194.2.19 vsnprintf() [2/2]	765
7.195 Vector< T, capacity > Class Template Reference	765
7.195.1 Detailed Description	766
7.195.2 Constructor & Destructor Documentation	766
7.195.2.1 Vector()	766
7.195.3 Member Function Documentation	766
7.195.3.1 add()	766
7.195.3.2 clear()	767
7.195.3.3 contains()	767
7.195.3.4 isEmpty()	767

7.195.3.5 maxCapacity()	767
7.195.3.6 operator[]() [1/2]	767
7.195.3.7 operator[]() [2/2]	768
7.195.3.8 quickRemoveAt()	768
7.195.3.9 remove()	768
7.195.3.10removeAt()	769
7.195.3.11reverse()	769
7.195.3.12size()	769
7.196Vector4 Class Reference	769
7.196.1 Detailed Description	769
7.196.2 Constructor & Destructor Documentation	770
7.196.2.1 Vector4() [1/2]	770
7.196.2.2 Vector4() [2/2]	770
7.196.3 Member Function Documentation	770
7.196.3.1 crossProduct()	770
7.197View< T > Class Template Reference	770
7.197.1 Detailed Description	771
7.197.2 Constructor & Destructor Documentation	771
7.197.2.1 View()	771
7.197.3 Member Function Documentation	771
7.197.3.1 bind()	771
7.198Widget Class Reference	772
7.198.1 Detailed Description	772
7.198.2 Constructor & Destructor Documentation	772
7.198.2.1 Widget()	772
7.198.2.2 ~Widget()	772
7.198.3 Member Function Documentation	773
7.198.3.1 getLastChild()	773
7.198.3.2 getType()	773
7.199WildcardTextStyle< T > Class Template Reference	773
7.199.1 Detailed Description	774
7.199.2 Member Function Documentation	775
7.199.2.1 setWildcardText()	775
7.199.2.2 setWildcardTextBuffer()	775
7.199.2.3 setWildcardTextColors()	775
7.199.2.4 setWildcardTextPosition()	775
7.199.2.5 setWildcardTextRotation()	776
7.199.2.6 setWildcardTextX()	776
7.199.2.7 setWildcardTextXY()	776
7.199.2.8 setWildcardTextY()	776

7.200ZoomAnimationImage Class Reference . . . . .	777
7.200.1 Detailed Description . . . . .	780
7.200.2 Member Enumeration Documentation . . . . .	780
7.200.2.1 States . . . . .	780
7.200.3 Constructor & Destructor Documentation . . . . .	780
7.200.3.1 ZoomAnimationImage() . . . . .	780
7.200.3.2 ~ZoomAnimationImage() . . . . .	780
7.200.4 Member Function Documentation . . . . .	781
7.200.4.1 getAlpha() . . . . .	781
7.200.4.2 getAnimationDelay() . . . . .	781
7.200.4.3 getLargeBitmap() . . . . .	781
7.200.4.4 getScalingMode() . . . . .	781
7.200.4.5 getSmallBitmap() . . . . .	781
7.200.4.6 getType() . . . . .	782
7.200.4.7 handleTickEvent() . . . . .	782
7.200.4.8 isRunning() . . . . .	782
7.200.4.9 isZoomAnimationRunning() . . . . .	782
7.200.4.10setAlpha() . . . . .	782
7.200.4.11setAnimationDelay() . . . . .	783
7.200.4.12setAnimationEndedCallback() . . . . .	783
7.200.4.13setBitmaps() . . . . .	783
7.200.4.14setCurrentState() . . . . .	784
7.200.4.15setDimension() . . . . .	784
7.200.4.16setHeight() . . . . .	784
7.200.4.17setPosition() . . . . .	784
7.200.4.18setScalingMode() . . . . .	785
7.200.4.19setWidth() . . . . .	785
7.200.4.20startTimerAndSetParameters() . . . . .	785
7.200.4.21startZoomAndMoveAnimation() . . . . .	786
7.200.4.22startZoomAnimation() . . . . .	787
7.200.4.23updateRenderingMethod() . . . . .	787
7.200.4.24updateZoomAnimationDeltaXY() . . . . .	787

# Chapter 1

## Changelog

4.10.0

=====

- \* Release date: November 5th, 2018
- \* Requirements:
  - \* TouchGFX is now only available for STM32 microcontrollers.
- \* New TouchGFX Designer Features:
  - \* Added support for the following widgets: ImageProgress, BoxProgress, TextProgress, LineProgress, CircleProgress, Line, Circle, BoxWithBorder, FlexButton, ScrollList, ScrollWheel and SwipeContainer.
  - \* Canvas Buffer setting can be adjusted on screens.
  - \* Support for screen transition: CoverTransition.
  - \* Now logs the following system information for use in support scenarios: Username, Designer version, Designer installation path, Windows version, Current culture, Installed .NET versions.
  - \* It is now possible to import a UI with any resolution to an application (resolution check has been removed).
  - \* Added button to show/hide clipped widgets.
  - \* Improved performance when dragging and resizing widgets on the canvas.
- \* New TouchGFX Core Features:
  - \* Circle and AbstractShape now supports higher precision on arc start and arc end for smoother arcs.
  - \* The internal Q5 structure now uses 32 bit instead of 16 bits for increased value range.
  - \* Added Circle::getPrecision().
  - \* Added functions FadeAnimator::isFadeAnimationRunning(), MoveAnimator::isMoveAnimationRunning(), AnimatedImage::isAnimatedImageRunning() and ZoomAnimationImage::isZoomAnimationImageRunning(). The old isRunning() functions have been deprecated.
  - \* ListLayout::setDirection() and getDirection() added.
  - \* Updated roo gem from 1.13.1 to 2.7.1.
  - \* Pressing SHIFT-F3 will copy the screen to the clipboard (Windows only).
  - \* Pressing CTRL-F3 will save the next 50 screens to the screenshots folder.
  - \* Generated assets are now indented properly.
  - \* ScrollableContainer::setScrollbarsPermanentlyVisible() added.
- \* Bugfixes in TouchGFX Designer
  - \* Fixed ModalWindow widget not resizing when Screen or Custom Container size changes.
  - \* Fixed generating code failing if a files hidden attribute was set to hidden.
  - \* Fixed changing the casing of a screen or custom container name resulting in a recompilation error.
  - \* Fixed bug where internet loss would crash the Designer if no Online Applications are available.

- \* Fixed ModalWindow widget position being generated incorrectly after loading a project.
  - \* Removed unnecessary recompilation when loading Designer project.
  - \* Fixed visual bug in ImagePicker where the "empty placeholder" would show up even though you have subfolders in current folder.
  - \* Fixed bug where the Designer was not using default credentials through proxy server.
  - \* Fixed bug where the Designer would not correctly report an error when trying to flash to a wrong target.
  - \* Fixed bug where having insufficient permissions to write to the chosen touchgfx path would crash the Designer.
  - \* Fixed bug where the Designer was incorrectly interpreting screen changes as an unsaved change.
  - \* Fixed a visual bug, where widgets inside a Container would not display properly when resizing the Container.
  - \* The Designer now closes a running Simulator process, when you load another application.
  - \* Fixed a bug where it was possible to drag widgets inside an instance of a Custom Container.
  - \* Circle did sometimes not render correctly, and invalidated rectangle was not calculated properly.
  - \* Fixed Circle when half line width was greater than radius.
- \* Bugfixes in TouchGFX Core:
- \* Fixed erroneous calculation of x & y values in setValue in LineProgress.cpp.
  - \* Circle did sometimes not render correctly, and invalidated rectangle was not calculated properly.
  - \* Fixed Circle when half line width was greater than radius.
  - \* Fixed drawing lines longer than 2047 pixels, e.g. 1449 pixels wide and 1449 pixels high.
  - \* Fixed bug preventing some Arabic ligatures from being rendered correctly.

#### 4.9.4

=====

\* Release date: January 25th, 2018

- \* Bugfixes:
- \* Reduced the time it takes to load an application in the Designer.

#### 4.9.3

=====

\* Release date: December 15th, 2017

- \* Bugfixes:
- \* Designer now uses default Windows proxy settings.
  - \* Package manager updates available packages when online.
  - \* Improved error description when offline.
  - \* Set text interaction works with resource texts.
  - \* Project updater updates MSVS projects with correct image formats.
  - \* Text size calculated wrongly in Designer in rare occasions.
  - \* Recent files ordered by date.
  - \* Corrected initialization of counter in Wait For interaction.
  - \* Fixed drawing of child elements in list layout, when resized.
  - \* Fixed loading of application with list layout widgets.
  - \* .otf font files now correctly rendered.
  - \* Dragging containers could in rare cases introduce wrong coordinates.
  - \* Fixed zero termination of wildcard text buffers.
  - \* Button With Label text rendering correction.
  - \* tgfx.exe packager works for more complex file layouts.
  - \* Source code included for containers.
  - \* Additional minor Designer UI fixes and improvements.

#### 4.9.2

=====

\* Release date: November 20th, 2017

---

\* Bugfixes:

- \* Fixed Designer issue where dragging elements on the canvas would in some cases cause an exception.

4.9.1

=====

\* Release date: November 16th, 2017

\* Bugfixes:

- \* Fixed several Designer issues with TextArea widgets when placed inside containers.
- \* Fixed an issue with interactions triggered by "Another interaction is done" disappearing when loading a project.
- \* On PCs with certain security policy configurations, the Designer was not able to create new projects correctly.
- \* Improved error handling in Designer if the asset generation, code compilation or post generation commands fail.
- \* Fixed an issue where the TouchgfxPath in Designer project files was not interpreted correctly.
- \* Some typography changes in Designer did not cause new code to be generated.
- \* Fixed issue with ImageConverter when assets folder was under svn control.
- \* ImageConverter could in certain cases fail to detect changes in assets.

4.9.0

=====

\* Release date: November 8th, 2017

\* New Features:

- \* Added a package manager for handling board support packages, demos and examples. The Designer will now fetch these from an online repository. Read more about this feature here:  
<https://touchgfx.zendesk.com/hc/en-us/articles/115002589211>
- \* All the old examples, demos and ports for various boards have been removed from the framework, and are now available as packages instead.
- \* Substantially improved text handling in the Designer. It is now possible to work with translations and wildcards in the Designer, so it should no longer be necessary to edit the texts.xlsx file manually.
- \* Designer is now much more flexible regarding application file structure, and is now able to auto-update IAR and Keil IDE projects regardless of file location.
- \* Added Designer support for the ScrollableContainer and ListLayout widgets.
- \* Added support for the SW4STM32 IDE.
- \* Added support for version 8.10 of IAR Embedded Workbench.
- \* Image converter now has an option to operate on folders, instead of being invoked once per .png file. This substantially speeds up the process of converting images. This mode is the default behavior for new projects.
- \* The GNU Arm Embedded toolchain (GCC cross-compiler) has been updated to version 6-2017-q2-update (gcc version 6.3.1).
- \* The GNU compiler for the PC simulator has been updated to version 6.3.0.
- \* Added gcc core libs compiled with -mfloat-abi=hard for Cortex-M4f and Cortex-M7.
- \* Increased number of widgets that can be registered as timer widgets from 24 to 32. Also added functions for obtaining information about which widgets are currently registered.

\* Bugfixes:

- \* AnimationTextureMapper::cancelMoveAnimation() is renamed to cancelAnimationTextureMapperAnimation() to avoid problems with MoveAnimator::cancelMoveAnimation().
- \* Fixed bug in PainterRGB565Bitmap when rendering solid pixels from

- an ARGB8888 Bitmap.
- \* Fixed rare bug in FontConvert if all used characters are missing from the font.
- \* Fixed uninitialized variables in the DMA class.
- \* Update procedure:
  - \* For this release additional steps might be needed. Please refer to the Known Issues article for details:
   
<https://touchgfx.zendesk.com/hc/en-us/articles/207507415>

4.8.0

=====

- \* Release date: March 10th, 2017
- \* Performance
  - \* LCD4bpp now draws characters up to 15% faster.
  - \* Canvas widgets now render slightly faster in certain situations.
- \* New Features:
  - \* TouchGFX Designer released. The core framework, Designer and environment shell are now bundled in a single installation. To get started with the Designer, please see
   
<https://touchgfx.zendesk.com/hc/en-us/articles/115001801745>
  - \* Support for Farsi and Arabic ligatures where sequences of up to three character are recognized.
  - \* Added support for Microsoft Visual Studio 2017.
  - \* TextArea and TextAreaWithWildcard(s) now support
   
setWidthTextAction() to automatically break lines and insert ellipsis at end of line, when the line is too long.
  - \* Added getter functions to Slider.
  - \* MoveAnimator and FadeAnimator can now clear the callback set for animation ended.
  - \* Errors from ImageConvert, TextConvert and FontConvert are now shown in the Error List window of Visual Studio.
  - \* Simulator applications are now Windows programs instead of Console programs. To use printf() or std::out, please see
   
<https://touchgfx.zendesk.com/hc/en-us/articles/205074511-Tips-tricks>
- \* Bugfixes
  - \* AbstractShape::updateAbstractShapeCache() is now a public function and should be called after one or more calls to
   
AbstractShape::setCorner(), to ensure shape is correct.
  - \* Simulator window can no longer be unintentionally resized.
  - \* F2 to highlight invalidated areas now works with old HALSDL.
  - \* PainterGRAY2Bitmap, PainterGRAY4Bitmap, PainterRGB565Bitmap and PainterRGB888Bitmap all failed to validate that painting was inside the size of the bitmap in some situations.
  - \* HALSDL2 (simulator) now uses 24bpp on screen to make colors in screenshots correct.
  - \* TiledImage::setOffset() now handles an empty bitmap correctly.
  - \* TiledImage::getSolidRect() would sometimes report wrong rect.
  - \* If text in a TextArea was rotated, resizeToCurrentText() and
   
resizeHeightToCurrentText() would swap the width/height.
  - \* Function getTextHeight() would not take line spacing into account. Functions like resizeToCurrentText() and others that use
   
the getTextHeight() function would not resize correctly.
  - \* SlideMenu::setState() did not handle EXPANDED state correctly.
- \* Update Procedure
  - \* Due to the addition of TouchGFX Designer, installation is now done via an .msi installer. For details and manual installation refer to
   
<https://touchgfx.zendesk.com/hc/en-us/articles/206819819>
  - \* Compatible with existing 4.x applications and HAL ports.

4.7.0

=====

- \* Release date: December 14th, 2016
- \* New Features:



- \* Source code for all the standard widgets and containers is now included. See the `touchgfx/framework/source/touchgfx` directory. Note that these classes are still present in the core library, and the source code files are not added to the IAR/Keil/gcc projects per default. For details, see the following article:  
<https://touchgfx.zendesk.com/hc/en-us/articles/115000035825>
- \* Optimized the handling of single frame buffer configuration on TFT controller based platforms, which in many cases eliminate the need for external RAM. For details on this feature, and how to enable it please see the following article:  
<https://touchgfx.zendesk.com/hc/en-us/articles/203649441>
- \* Substantial performance optimizations of the canvas widget system and all the standard painters. Expect a very significant increase in performance if many pixels are being drawn, and a smaller increase in performance for minor shapes (e.g. graph lines). The "PainterVerticalAlpha" used in our demos have also been updated. If you are using custom painters please refer to the Known Issues article:  
<https://touchgfx.zendesk.com/hc/en-us/articles/207507415>
- \* The text converter tool will now combine identical translations across all languages, resulting in reduced footprint. The result of this process will be printed during asset generation.  
NOTE: This behavior is enabled by default. If you have an existing project where you manipulate the text data structures (e.g. load a single language into RAM), this optimization might break your code. The optimization can be disabled by adding the following  

```
remap_identical_texts := no (for "make"-based generation)
<RemapIdenticalTexts>no</RemapIdenticalTexts> (for MSVS)
```
- \* Updated SDL version used by simulator from 1.2 to 2.0.4. SDL1.2 is still present in the distribution, but all examples and projects now use SDL2. For more details, please see:  
<https://touchgfx.zendesk.com/hc/en-us/articles/115000011245>
- \* Support for skinning the simulator with .png files. If the .png files contain non-opaque areas, the simulator window will be shaped accordingly. See `display_orientation_example` for a code example or read the following article:  
<https://touchgfx.zendesk.com/hc/en-us/articles/115000014669>
- \* On ST targets with Chrom-ART, the Box widget will now be drawn by DMA even when `alpha < 255` (BLIT\_OP\_FILL\_WITH\_ALPHA support).
- \* `TextArea` and `TextArea` with `wildcard(s)` now support `setWidthTextAction()` to automatically wrap long lines.
- \* Added the ability to display a "fallback" character in case a non-existing glyph is encountered at runtime. This is configured in the typography sheet of the text database.
- \* Added options for forcing the inclusion of additional glyphs in a font. This makes it much easier to handle dynamic texts where the glyphs are not known at compile time. This is configured in the typography sheet of the text database.
- \* Output from the `TextConvert` utility is now post-processed to give significant saving by mapping identical strings to the same memory areas.
- \* Added built-in `BitmapId` called `BITMAP_ANIMATION_STORAGE` which can be used to refer to the animation storage when assigning a `Bitmap` to a widget.
- \* Added dither algorithm selection from `config/gcc/app.mk` and `config/msvs/Application.props`.
- \* It is possible to save a simulator screenshot programatically, by using:

```
#ifdef SIMULATOR
    (static_cast<HALSDL2*>(HAL::getInstance()))->saveScreenshot();
#endif
```
- \* `ScrollableContainer` now properly ignores invisible elements.
- \* `DigitalClock` now supports a zero to be displayed in front of the hour indicator (if `hour < 10`).
- \* The simulator can now highlight the areas being invalidated. Press F2 to toggle this feature.
- \* Added `Unicode::vsprintf` functions that take `va_list` arguments instead of `ellipsis`.
- \* Bugfixes
  - \* `Unicode::sprintfFloat` did not print `<space>` instead of `'+'` if the format string was `"% f"`. Also, the sign of floating point numbers in range `]-1..0[` would not be printed with sign so for example `-0.5` would print as `0.5`.

- \* Fixed a bug that could cause TextureMapper to read outside source bitmap memory area.
- \* GPIO.cpp for STM32F769-Discovery and Eval boards had some incorrect GPIO pin manipulations (used for performance measurement).
- \* Some methods in Slider.hpp were missing a virtual declaration.
- \* Fixed a bug in BoardConfiguration for STM32F769-Discovery board causing 24bpp color mode to be displayed incorrectly.
- \* AnimatedImage - setBitmap(..) should not be used and is now private For AnimatedImage use setBitmaps(..) instead.
- \* Project files and Makefile have been updated to allow the TouchGFX framework to be placed on another disk drive than the project being developed.
- \* TouchGFX Environment (version 2.8)
  - \* "make.exe" is now version 4.1 which allows for parallel compilation, by adding e.g. "-j8" to your make command. This substantially speeds up compilation. If your makefile is from TouchGFX 4.2.0 or earlier, you will need to either update it, or to use make-3.81.exe
  - \* g++ could in some cases report "There is no disk in the drive. Please insert a disk into drive E:.". This has been fixed by upgrading gcc from version 4.8.1 to version 4.9.3.

#### 4.6.1

=====

- \* Release date: September 12th, 2016
- \* Performance
  - \* Optimization improvements of core library for GCC on Cortex-M4 and Cortex-M7, providing significant speedup of especially TextureMapper and Canvas widgets compared to TouchGFX 4.6.0.
- \* New Features
  - \* New function in HALSDL to set title of simulator window see HALSDL::setWindowTitle().
  - \* BW\_RLE format (1bpp displays) now compresses better. Remember to remove old generated files and re-generate assets.
  - \* STM32F756G-EVAL using IAR now supports flashing of external memory.
- \* Bugfixes
  - \* Added IAR linker redirect commands to fix linker errors when compiling a Cortex-M4 based target with IAR 7.x.
  - \* Assigning different memory buffers to CanvasWidgetRenderer using setupBuffer() could in rare cases result in memory corruption.
  - \* TextureMapper could in rare cases draw outside the frame buffer.
  - \* Setting the offset of a TiledImage did not work properly.
  - \* Fixed two issues that would in some cases cause memory corruption when deleting dynamic bitmaps.
  - \* Missing virtual method declarations in AnalogClock added.
  - \* Fixed a problem in GCC linker script for LPC4088DisplayModule which caused texts and fonts to be placed in external flash.
  - \* For those using fontconvert.out on its own, the output directory is now automatically created if it does not exist.
  - \* ScrollableContainers could in rare cases send a wrong drag event to a child.
  - \* Monochrome (1bpp) displays with width not divisible by 8 would not display text correctly.
  - \* Slightly increased default touch sample rate on STM32F746G Discovery board.

#### 4.6.0

=====

- \* Release date: June 14th, 2016
- \* New features
  - \* Added support for 2bpp grayscale displays. See <https://touchgfx.zendesk.com/hc/en-us/articles/208237889> for details.

- \* Added support for 4bpp grayscale displays. See <https://touchgfx.zendesk.com/hc/en-us/articles/209003105> for details.
- \* New widget TiledImage. Will display one or more repetitions of an image. The number of repetitions depends on the size of the widget and the size of the image.
- \* New widget RepeatButton. A button that will repeatedly fire click events when pressed.
- \* New widget AnimationTextureMapper. TextureMapper with build in animation features. See [animation\\_texture\\_mapper\\_example](#).
- \* New containers AnalogClock and DigitalClock, see [clock\\_example](#).
- \* New containers ProgressIndicators, see [progress\\_indicator\\_example](#).
- \* New container ModalWindow. Creates a window on top of the main screen and a shade on the rest of the main screen. No clicks are passed on to the main screen as long as the modal window is visible. See [example\\_modal\\_window\\_example](#).
- \* New container SlideMenu. Animating side/top/bottom-menu that has an activate button for sliding it in/out of the screen. A timeout can be set for automatical hiding when idle for a period of time.
- \* Canvas Widget Line supports ROUND\_CAP\_ENDING and setCapPrecision() to control the round cap.
- \* Simulator can now generate ticks very close to the frequency of the hardware. See <https://touchgfx.zendesk.com/hc/en-us/articles/205074511> for details.
- \* Mouse X and Y coordinates are put in the title of the window in the simulator. (press F1 to (de)active this when running the simulator). See <https://touchgfx.zendesk.com/hc/en-us/articles/205074511> for details.
- \* ST Cube drivers updated to version 1.4.0 for STM32F7 MCU and STM32F7 based boards.
- \* Added support for the STM32769I-EVAL board.
- \* Added support for the STM32F769I-Discovery board.
- \* Screenshots made from the simulator (F3) are now saved under a name with timestamp to prevent old screenshots to be overwritten by accident.
- \* Simulator now outputs canvas widget memory usage to easily find optimal canvas memory buffer size. See <https://touchgfx.zendesk.com/hc/en-us/articles/205074511> for details.
- \* Bugfixes
  - \* DMA drivers for ST boards: express DMA2D instance initialization for STM32F7. Fixed incorrect used of CLUT\_CM for F4-Discovery.
  - \* DMA drivers for LPC17xx, LPC18xx, LPC43xx did not behave correctly if other DMA channels are in use simultanously. They now properly look at flags for channel 0 only.
  - \* Touch controller drivers for ST boards now properly checks that initialization was OK before querying.
  - \* Mouse clicks in the simulator would not always be detected.
  - \* ImageConvert.exe has RGB565 as default (and sensible defaults for other opque formats)
  - \* ImageConvert would not work for a BW image scheduled for compression (BW\_RLE) and rotation (.90. in filename) if the image would become too large if compressed (falling back to BW format).
  - \* All Makefiles now use abspath instead of realpath.
  - \* AnimatedImage now allows the animation to be restarted from the AnimationEnded callback function.
  - \* QSPI flash size corrected to 64MBytes for STM32756G-EVAL board.
  - \* Added D-cache invalidation to STM32F7HAL::flushFrameBuffer. This fixes occasional graphics errors on STM32F7 when in single frame buffer mode and fb was located in SRAM.
  - \* The otm8009a displays (STM32769-DISCO, STM32769-EVAL, STM32469-DISCO, STM32469-EVAL) are now using maximum display brightness.
  - \* Added a workaround for a bug in IAR 7.50.x regarding va\_list name mangling.
- \* 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>

4.5.1  
=====

\* Release date: March 14th, 2016

#### \* Bugfixes

- \* Fixed two IAR linker issues related to resolving the `va_list` symbol, which would cause some versions of IAR being unable to link the example projects.
- \* STM32F4-Discovery board would draw solid rectangles with the wrong color in 16bpp mode.
- \* The Canvas Widget Renderer no longer performs unaligned memory accesses.
- \* `vApplicationIdleHook` (FreeRTOS specific) no longer blocks, which previously prevented FreeRTOS from freeing memory if tasks were deleted.
- \* Arabic words with accent in the middle would not render properly.
- \* Added `PixelDataWidget::getAlpha()`.
- \* `Unicode::strncpy()` with a `char*` as source would not copy characters with ascii codes above 127 properly.

#### 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.
  - \* sprintf("%x") would generate uppper case hex. Now "%X" generates uppercase hex and "%x" generates lower case hex, just like the standard sprintf().
  - \* 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 transparency.
  - \* 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 lbpp displays when width and/or height was not a multiple of 8.
  - \* Fixed bug in ScrollableContainer where CANCEL events were 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](mailto: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 2

## 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 3

# Namespace Index

### 3.1 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 . . . . .

[35](#)





## Chapter 4

# Hierarchical Index

### 4.1 Class Hierarchy

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

AbstractPainter . . . . .	71
AbstractPainterBW . . . . .	74
PainterBW . . . . .	496
PainterBWBitmap . . . . .	498
AbstractPainterGRAY2 . . . . .	76
PainterGRAY2 . . . . .	501
PainterGRAY2Bitmap . . . . .	504
AbstractPainterGRAY4 . . . . .	78
PainterGRAY4 . . . . .	507
PainterGRAY4Bitmap . . . . .	510
AbstractPainterRGB565 . . . . .	81
PainterRGB565 . . . . .	513
PainterRGB565Bitmap . . . . .	517
AbstractPainterRGB888 . . . . .	84
PainterRGB888 . . . . .	520
PainterRGB888Bitmap . . . . .	523
AbstractPartition . . . . .	86
Partition< ListOfTypes, NUMBER_OF_ELEMENTS > . . . . .	528
AnimationTextureMapper::AnimationSetting . . . . .	122
Bitmap . . . . .	134
Bitmap::BitmapData . . . . .	144
BlitOp . . . . .	144
ButtonController . . . . .	162
Buttons . . . . .	163
Bitmap::CacheTableEntry . . . . .	171
Keyboard::CallbackArea . . . . .	178
Canvas . . . . .	179
Cell . . . . .	190
Color . . . . .	218
colortype . . . . .	220
CWRUtil . . . . .	233
DisplayTransformation . . . . .	244
DMA_Interface . . . . .	247
NoDMA . . . . .	486
DMA_Queue . . . . .	252
LockFreeDMA_Queue . . . . .	465

Drawable . . . . .	259
Container . . . . .	224
MoveAnimator< touchgfx::Container > . . . . .	480
AbstractButtonContainer . . . . .	65
ClickButtonTrigger . . . . .	211
RepeatButtonTrigger . . . . .	580
ToggleButtonTrigger . . . . .	733
TouchButtonTrigger . . . . .	737
AbstractClock . . . . .	67
AnalogClock . . . . .	108
DigitalClock . . . . .	239
AbstractProgressIndicator . . . . .	93
AbstractDirectionProgress . . . . .	70
BoxProgress . . . . .	149
ImageProgress . . . . .	379
CircleProgress . . . . .	205
LineProgress . . . . .	457
TextProgress . . . . .	699
DrawableList . . . . .	273
Keyboard . . . . .	389
ListLayout . . . . .	462
ModalWindow . . . . .	475
ScrollableContainer . . . . .	596
ScrollBase . . . . .	607
ScrollList . . . . .	622
ScrollWheelBase . . . . .	629
ScrollWheel . . . . .	628
ScrollWheelWithSelectionMode . . . . .	633
SlideMenu . . . . .	647
Slider . . . . .	657
SwipeContainer . . . . .	677
ZoomAnimationImage . . . . .	777
Widget . . . . .	772
AbstractButton . . . . .	63
Button . . . . .	159
ButtonWithIcon . . . . .	163
ButtonWithLabel . . . . .	167
RepeatButton . . . . .	577
ToggleButton . . . . .	731
RadioButton . . . . .	552
TouchArea . . . . .	735
Box . . . . .	145
BoxWithBorder . . . . .	152
CanvasWidget . . . . .	182
AbstractShape . . . . .	98
Shape< POINTS > . . . . .	643
Circle . . . . .	193
Line . . . . .	445
CanvasWidgetRenderer . . . . .	186
CoverTransition< templateDirection >::FullSolidRect . . . . .	319
Image . . . . .	374
AnimatedImage . . . . .	115
TiledImage . . . . .	722
PixelDataWidget . . . . .	531
ScalableImage . . . . .	582
SnapshotWidget . . . . .	672
TextArea . . . . .	681

TextAreaWithWildcardBase . . . . .	695
TextAreaWithOneWildcard . . . . .	689
TextAreaWithTwoWildcards . . . . .	692
TextureMapper . . . . .	706
AnimationTextureMapper . . . . .	122
DrawableListItemsInterface . . . . .	284
DrawableListItems< TYPE, SIZE > . . . . .	282
DrawingSurface . . . . .	285
Bitmap::DynamicBitmapData . . . . .	286
EasingEquations . . . . .	286
Edge . . . . .	303
Event . . . . .	305
ClickEvent . . . . .	212
DragEvent . . . . .	255
GestureEvent . . . . .	327
Font . . . . .	310
ConstFont . . . . .	221
InternalFlashFont . . . . .	383
FontManager . . . . .	318
FontProvider . . . . .	319
GenericCallback< T1, T2, T3 > . . . . .	320
Callback< dest_type, T1, T2, T3 > . . . . .	171
GenericCallback< const T &, const touchgfx::ClickEvent &> . . . . .	320
GenericCallback< const touchgfx::AbstractButton & > . . . . .	320
GenericCallback< const touchgfx::AbstractButton &, void, void > . . . . .	320
Callback< touchgfx::RadioButtonGroup, const touchgfx::AbstractButton & > . . . . .	171
Callback< touchgfx::SlideMenu, const touchgfx::AbstractButton &> . . . . .	171
GenericCallback< const touchgfx::AbstractButtonContainer & > . . . . .	320
GenericCallback< const touchgfx::AnimatedImage &> . . . . .	320
GenericCallback< const touchgfx::AnimationTextureMapper & > . . . . .	320
GenericCallback< const touchgfx::DragEvent &> . . . . .	320
GenericCallback< const touchgfx::FadeAnimator< T > & > . . . . .	320
GenericCallback< const touchgfx::MoveAnimator< T > & > . . . . .	320
GenericCallback< const touchgfx::MoveAnimator< touchgfx::Container > & > . . . . .	320
GenericCallback< const touchgfx::MoveAnimator< touchgfx::Container > &, void, void > . . . . .	320
Callback< touchgfx::SlideMenu, const touchgfx::MoveAnimator< touchgfx::Container > &> . . . . .	171
GenericCallback< const touchgfx::SlideMenu & > . . . . .	320
GenericCallback< const touchgfx::Slider &, int > . . . . .	320
GenericCallback< const touchgfx::ZoomAnimationImage &> . . . . .	320
GenericCallback< int16_t > . . . . .	320
GenericCallback< T1 > . . . . .	320
Callback< dest_type, T1, void, void > . . . . .	175
GenericCallback< T1, T2 > . . . . .	320
Callback< dest_type, T1, T2, void > . . . . .	173
GenericCallback< T1, T2, void > . . . . .	322
GenericCallback< T1, void, void > . . . . .	324
GenericCallback< touchgfx::Drawable &, void, void > . . . . .	320
Callback< touchgfx::CoverTransition, touchgfx::Drawable &> . . . . .	171
Callback< touchgfx::SlideTransition, touchgfx::Drawable &> . . . . .	171
GenericCallback< touchgfx::DrawableListItemsInterface *, int16_t, int16_t > . . . . .	320
GenericCallback< Unicode::UnicodeChar > . . . . .	320
GenericCallback< void > . . . . .	325
GenericCallback<> . . . . .	320
Callback< dest_type, void, void, void > . . . . .	177
Gestures . . . . .	329

GlyphNode . . . . .	331
GPIO . . . . .	333
Gradients . . . . .	335
HAL . . . . .	336
HALSDL2 . . . . .	359
I2C . . . . .	368
Scanline::iterator . . . . .	385
JSMOCHelper . . . . .	386
KerningNode . . . . .	389
Keyboard::Key . . . . .	389
Keyboard::KeyMapping . . . . .	397
Keyboard::KeyMappingList . . . . .	397
Keyboard::Layout . . . . .	397
LCD . . . . .	398
LCD16bpp . . . . .	411
LCD1bpp . . . . .	418
LCD24bpp . . . . .	424
LCD2bpp . . . . .	431
LCD4bpp . . . . .	437
LED . . . . .	444
Matrix4x4 . . . . .	468
MCUInstrumentation . . . . .	473
MVPHeap . . . . .	485
OSWrappers . . . . .	491
Outline . . . . .	493
Pair< T1, T2 > . . . . .	527
Pair< int16_t, int16_t > . . . . .	527
Point . . . . .	534
Point3D . . . . .	534
Presenter . . . . .	538
CWRUtil::Q10 . . . . .	539
CWRUtil::Q15 . . . . .	541
CWRUtil::Q5 . . . . .	543
Quadruple . . . . .	548
Point4 . . . . .	535
Vector4 . . . . .	769
RadioButtonGroup< CAPACITY > . . . . .	558
Rasterizer . . . . .	563
Rect . . . . .	567
Renderer . . . . .	572
RenderingBuffer . . . . .	573
Scanline . . . . .	586
Screen . . . . .	590
View< T > . . . . .	770
AbstractShape::ShapePoint< T > . . . . .	647
LCD::StringVisuals . . . . .	675
TextProvider . . . . .	703
Texts . . . . .	705
TextureSurface . . . . .	722
TouchCalibration . . . . .	738
TouchController . . . . .	739
I2CTouchController . . . . .	370
NoTouchController . . . . .	488
SDL2TouchController . . . . .	641
SDLTouchController . . . . .	642
Transition . . . . .	740
CoverTransition< templateDirection > . . . . .	230

NoTransition . . . . .	490
SlideTransition< templateDirection > . . . . .	667
TypedText . . . . .	746
TypedText::TypedTextData . . . . .	749
UIEventListener . . . . .	750
Application . . . . .	126
MVPApplication . . . . .	484
Unicode . . . . .	752
Vector< T, capacity > . . . . .	765
Vector< touchgfx::Drawable *, MAX_TIMER_WIDGETS > . . . . .	765
Vector< touchgfx::Rect, 8 > . . . . .	765
T	
AnimatedImageButtonStyle< T > . . . . .	120
BoxWithBorderButtonStyle< T > . . . . .	156
ClickListener< T > . . . . .	216
Draggable< T > . . . . .	258
Snapper< T > . . . . .	670
FadeAnimator< T > . . . . .	306
IconButtonStyle< T > . . . . .	371
ImageButtonStyle< T > . . . . .	377
MoveAnimator< T > . . . . .	480
PreRenderable< T > . . . . .	536
TextButtonStyle< T > . . . . .	696
TiledImageButtonStyle< T > . . . . .	729
TwoWildcardTextButtonStyle< T > . . . . .	742
WildcardTextButtonStyle< T > . . . . .	773



## Chapter 5

# Class Index

### 5.1 Class List

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

<a href="#">AbstractButton</a>	This class defines an abstract interface for button-like elements . . . . .	63
<a href="#">AbstractButtonContainer</a>	An abstract button container . . . . .	65
<a href="#">AbstractClock</a>	Superclass of clock widgets . . . . .	67
<a href="#">AbstractDirectionProgress</a>	An abstract direction progress . . . . .	70
<a href="#">AbstractPainter</a>	An abstract class for creating painter classes for drawing canvas widgets . . . . .	71
<a href="#">AbstractPainterBW</a>	A Painter that will paint using a color on a <a href="#">LCD1bpp</a> display . . . . .	74
<a href="#">AbstractPainterGRAY2</a>	A Painter that will paint using a color and an alpha value . . . . .	76
<a href="#">AbstractPainterGRAY4</a>	A Painter that will paint using a color and an alpha value . . . . .	78
<a href="#">AbstractPainterRGB565</a>	A Painter that will paint using a color and an alpha value . . . . .	81
<a href="#">AbstractPainterRGB888</a>	A Painter that will paint using a color and an alpha value . . . . .	84
<a href="#">AbstractPartition</a>	This type defines an abstract interface to a storage partition for allocating memory slots of equal size . . . . .	86
<a href="#">AbstractProgressIndicator</a>	An abstract progress indicator . . . . .	93
<a href="#">AbstractShape</a>	Simple widget capable of drawing a <a href="#">abstractShape</a> . . . . .	98
<a href="#">AnalogClock</a>	An analog clock . . . . .	108
<a href="#">AnimatedImage</a>	A widget capable of basic animation using a range of bitmaps . . . . .	115
<a href="#">AnimatedImageButtonStyle&lt; T &gt;</a>	An animated image button style. An animated image button style. This class is supposed to be used with one of the <a href="#">ButtonTrigger</a> classes to create a functional button. This class will show the first or last image of an animated image depending on the state of the button (pressed or released). When the state changes the button will show the sequence of images in forward or reversed order . . . . .	120

<a href="#">AnimationTextureMapper::AnimationSetting</a>	Information about how a specific animation parameter should be animated . . . . .	122
<a href="#">AnimationTextureMapper</a>	A texture mapper with animation capabilities . . . . .	122
<a href="#">Application</a>	Main interface for manipulating screen contents . . . . .	126
<a href="#">Bitmap</a>	This class provides a proxy object for a bitmap image . . . . .	134
<a href="#">Bitmap::BitmapData</a>	Data of a bitmap . . . . .	144
<a href="#">BlitOp</a>	<a href="#">BlitOp</a> instances carry the required information for performing operations on the <a href="#">LCD</a> (frame buffer) using DMA . . . . .	144
<a href="#">Box</a>	Simple widget capable of showing a rectangle of a specific color and an optional alpha . . . . .	145
<a href="#">BoxProgress</a>	A box progress . . . . .	149
<a href="#">BoxWithBorder</a>	A box with border . . . . .	152
<a href="#">BoxWithBorderStyle&lt; T &gt;</a>	A box with border button style . . . . .	156
<a href="#">Button</a>	A button with two states . . . . .	159
<a href="#">ButtonController</a>	Interface for sampling external key events . . . . .	162
<a href="#">Buttons</a>	A buttons . . . . .	163
<a href="#">ButtonWithIcon</a>	A <a href="#">Button</a> specialization that also displays an icon on top of the button bitmap . . . . .	163
<a href="#">ButtonWithLabel</a>	A <a href="#">Button</a> specialization that also displays a text on top of the button bitmap . . . . .	167
<a href="#">Bitmap::CacheTableEntry</a>	Cache bookkeeping . . . . .	171
<a href="#">Callback&lt; dest_type, T1, T2, T3 &gt;</a>	A <a href="#">Callback</a> is basically a wrapper of a pointer-to-member-function . . . . .	171
<a href="#">Callback&lt; dest_type, T1, T2, void &gt;</a>	A <a href="#">Callback</a> is basically a wrapper of a pointer-to-member-function . . . . .	173
<a href="#">Callback&lt; dest_type, T1, void, void &gt;</a>	A <a href="#">Callback</a> is basically a wrapper of a pointer-to-member-function . . . . .	175
<a href="#">Callback&lt; dest_type, void, void, void &gt;</a>	A <a href="#">Callback</a> is basically a wrapper of a pointer-to-member-function . . . . .	177
<a href="#">Keyboard::CallbackArea</a>	Mapping from rectangle to a callback method to execute . . . . .	178
<a href="#">Canvas</a>	Class for easy rendering using <a href="#">CanvasWidgetRenderer</a> . . . . .	179
<a href="#">CanvasWidget</a>	Class for drawing complex polygons on the <a href="#">LCD</a> using <a href="#">CanvasWidgetRenderer</a> . . . . .	182
<a href="#">CanvasWidgetRenderer</a>	Class for supporting drawing of figures . . . . .	186
<a href="#">Cell</a>	A pixel cell . . . . .	190
<a href="#">Circle</a>	Simple widget capable of drawing a circle . . . . .	193
<a href="#">CircleProgress</a>	A circle progress . . . . .	205
<a href="#">ClickButtonTrigger</a>	A click button trigger . . . . .	211



<a href="#">ClickEvent</a>	
A click event	212
<a href="#">ClickListener&lt; T &gt;</a>	
Mix-in class that extends a class with a click action event	216
<a href="#">Color</a>	
Contains functionality for color conversion	218
<a href="#">colortype</a>	
Type for representing a color	220
<a href="#">ConstFont</a>	
A <a href="#">ConstFont</a> is a <a href="#">Font</a> implementation that has its contents defined at compile-time and usually placed in read-only memory	221
<a href="#">Container</a>	
A <a href="#">Container</a> is a <a href="#">Drawable</a> that can have child nodes	224
<a href="#">CoverTransition&lt; templateDirection &gt;</a>	
A <a href="#">Transition</a> that slides from one screen to the next	230
<a href="#">CWRUtil</a>	
Helper classes and functions for <a href="#">CanvasWidget</a>	233
<a href="#">DigitalClock</a>	
A digital clock	239
<a href="#">DisplayTransformation</a>	
Defines transformations from display space to frame buffer space	244
<a href="#">DMA_Interface</a>	
<a href="#">DMA_Interface</a> provides basic functionality and structure for processing "blit" operations using DMA	247
<a href="#">DMA_Queue</a>	
This class provides an interface for a FIFO (circular) list used by <a href="#">DMA_Interface</a> and descendants for storing <a href="#">BlitOp</a> 's	252
<a href="#">DragEvent</a>	
A drag event	255
<a href="#">Draggable&lt; T &gt;</a>	
Mix-in class that extends a class to become draggable	258
<a href="#">Drawable</a>	
Abstract definition of something that can be drawn	259
<a href="#">DrawableList</a>	
A container able to display many items using only a few drawables	273
<a href="#">DrawableListItems&lt; TYPE, SIZE &gt;</a>	
An array of drawables used by <a href="#">DrawableList</a>	282
<a href="#">DrawableListItemsInterface</a>	
A drawable list items interface	284
<a href="#">DrawingSurface</a>	
The destination of a draw operation. Contains a pointer to where to draw and the stride of the drawing surface	285
<a href="#">Bitmap::DynamicBitmapData</a>	
Data of a dynamic bitmap	286
<a href="#">EasingEquations</a>	
Defines the "Penner easing functions", which are a de facto standard computing aesthetically pleasing motion animations	286
<a href="#">Edge</a>	
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	303
<a href="#">Event</a>	
Simple base class for events	305
<a href="#">FadeAnimator&lt; T &gt;</a>	
A <a href="#">FadeAnimator</a> makes the template class T able to animate an alpha fade	306
<a href="#">Font</a>	
The font base class	310
<a href="#">FontManager</a>	
This class is the entry point for looking up a font based on a font id	318

<a href="#">FontProvider</a>	
A generic pure virtual definition of a <a href="#">FontProvider</a>	319
<a href="#">CoverTransition&lt; templateDirection &gt;::FullSolidRect</a>	
A <a href="#">Widget</a> that returns a solid rect of the same size as the application	319
<a href="#">GenericCallback&lt; T1, T2, T3 &gt;</a>	
<a href="#">GenericCallback</a> is the base class for callbacks	320
<a href="#">GenericCallback&lt; T1, T2, void &gt;</a>	
<a href="#">GenericCallback</a> is the base class for callbacks	322
<a href="#">GenericCallback&lt; T1, void, void &gt;</a>	
<a href="#">GenericCallback</a> is the base class for callbacks	324
<a href="#">GenericCallback&lt; void &gt;</a>	
<a href="#">GenericCallback</a> is the base class for callbacks	325
<a href="#">GestureEvent</a>	
A gesture event	327
<a href="#">Gestures</a>	
This class implements the detection of gestures	329
<a href="#">GlyphNode</a>	
Struct providing information about a glyph	331
<a href="#">GPIO</a>	
Interface class for manipulating GPIOs in order to do performance measurements on target	333
<a href="#">Gradients</a>	
<a href="#">Gradients</a> contains all the data to interpolate u,v texture coordinates and z coordinates across a planar surface	335
<a href="#">HAL</a>	
Hardware Abstraction Layer	336
<a href="#">HALSDL2</a>	
<a href="#">HAL</a> implementation for the TouchGFX simulator	359
<a href="#">I2C</a>	
Platform independent interface for <a href="#">I2C</a> drivers	368
<a href="#">I2CTouchController</a>	
Specific I2C-enabled type of Touch Controller	370
<a href="#">IconButtonStyle&lt; T &gt;</a>	
An icon button style	371
<a href="#">Image</a>	
Simple widget capable of showing a bitmap	374
<a href="#">ImageButtonStyle&lt; T &gt;</a>	
An image button style	377
<a href="#">ImageProgress</a>	
An image progress	379
<a href="#">InternalFlashFont</a>	
An <a href="#">InternalFlashFont</a> has both glyph table and glyph data placed in a flash which supports random access read (i.e. not a NAND flash)	383
<a href="#">Scanline::iterator</a>	
An iterator to help go through all the elements that make up a <a href="#">Scanline</a>	385
<a href="#">JSMOCHelper</a>	
Helper class providing caching of certain information while the JSMOC algorithm runs during draw operations	386
<a href="#">KerningNode</a>	
Structure providing information about a kerning for a given char pair	389
<a href="#">Keyboard::Key</a>	
Mapping from rectangle to key id	389
<a href="#">Keyboard</a>	
The keyboard provides text input for touch devices	389
<a href="#">Keyboard::KeyMapping</a>	
Mapping from key id to <a href="#">Unicode</a> character	397
<a href="#">Keyboard::KeyMappingList</a>	
List of KeyMappings to use	397

<a href="#">Keyboard::Layout</a>	Definition of the keyboard layout. The keyboard can handle changing layouts, so different keyboard modes can be implemented by changing layouts and key mappings . . . . .	397
<a href="#">LCD</a>	This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles . . . . .	398
<a href="#">LCD16bpp</a>	This class contains the various low-level drawing routines for drawing bitmaps . . . . .	411
<a href="#">LCD1bpp</a>	This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles . . . . .	418
<a href="#">LCD24bpp</a>	This class contains the various low-level drawing routines for drawing bitmaps . . . . .	424
<a href="#">LCD2bpp</a>	This class contains the various low-level drawing routines for drawing bitmaps . . . . .	431
<a href="#">LCD4bpp</a>	This class contains the various low-level drawing routines for drawing bitmaps . . . . .	437
<a href="#">LED</a>	A led . . . . .	444
<a href="#">Line</a>	Simple <a href="#">CanvasWidget</a> capable of drawing a line . . . . .	445
<a href="#">LineProgress</a>	A line progress . . . . .	457
<a href="#">ListLayout</a>	This class provides a layout mechanism for arranging <a href="#">Drawable</a> instances adjacently in the specified Direction . . . . .	462
<a href="#">LockFreeDMA_Queue</a>	This implements a simple lock-free FIFO queue (single producer, single consumer) . . . . .	465
<a href="#">Matrix4x4</a>	This class represents row major 4x4 homogeneous matrices . . . . .	468
<a href="#">MCUInstrumentation</a>	Interface for instrumenting processors to measure MCU load via measured CPU cycles . . . . .	473
<a href="#">ModalWindow</a>	<a href="#">Container</a> for displaying a modal window and hijacking touch event to underlying view and widgets . . . . .	475
<a href="#">MoveAnimator&lt; T &gt;</a>	A <a href="#">MoveAnimator</a> makes the template class T able to animate a movement . . . . .	480
<a href="#">MVPApplication</a>	A specialization of the TouchGFX <a href="#">Application</a> class . . . . .	484
<a href="#">MVPHeap</a>	Generic heap class for MVP applications . . . . .	485
<a href="#">NoDMA</a>	This is an "empty" DMA subclass that does nothing except assert if accidentally used . . . . .	486
<a href="#">NoTouchController</a>	. . . . .	488
<a href="#">NoTransition</a>	The most simple <a href="#">Transition</a> without any visual effects . . . . .	490
<a href="#">OSWrappers</a>	This class specifies OS wrappers for dealing with the frame buffer semaphore and the VSYNC signal . . . . .	491
<a href="#">Outline</a>	An internal class that implements the main rasterization algorithm . . . . .	493
<a href="#">PainterBW</a>	A Painter that will paint using a color on a <a href="#">LCD1bpp</a> display . . . . .	496
<a href="#">PainterBWBitmap</a>	A Painter that will paint using a bitmap . . . . .	498
<a href="#">PainterGRAY2</a>	A Painter that will paint using a color and an alpha value . . . . .	501

<a href="#">PainterGRAY2Bitmap</a>	
A Painter that will paint using a bitmap . . . . .	504
<a href="#">PainterGRAY4</a>	
A Painter that will paint using a color and an alpha value . . . . .	507
<a href="#">PainterGRAY4Bitmap</a>	
A Painter that will paint using a bitmap . . . . .	510
<a href="#">PainterRGB565</a>	
A Painter that will paint using a color and an alpha value . . . . .	513
<a href="#">PainterRGB565Bitmap</a>	
A Painter that will paint using a bitmap . . . . .	517
<a href="#">PainterRGB888</a>	
A Painter that will paint using a color and an alpha value . . . . .	520
<a href="#">PainterRGB888Bitmap</a>	
A Painter that will paint using a bitmap . . . . .	523
<a href="#">Pair&lt; T1, T2 &gt;</a>	
A simple struct for holding pairs of data . . . . .	527
<a href="#">Partition&lt; ListOfTypes, NUMBER_OF_ELEMENTS &gt;</a>	
This type provides a concrete <a href="#">Partition</a> of memory-slots capable of holding any of the specified list of types . . . . .	528
<a href="#">PixelDataWidget</a>	
A widget for displaying a buffer of pixel data . . . . .	531
<a href="#">Point</a>	
A simple struct containing coordinates . . . . .	534
<a href="#">Point3D</a>	
A 3D point . . . . .	534
<a href="#">Point4</a>	
This class represents a homogeneous 3D point . . . . .	535
<a href="#">PreRenderable&lt; T &gt;</a>	
This mixin can be used on any <a href="#">Drawable</a> . . . . .	536
<a href="#">Presenter</a>	
The <a href="#">Presenter</a> base class that all application-specific presenters should derive from . . . . .	538
<a href="#">CWRUtil::Q10</a>	
Defines a number with 10 bits reserved for fraction . . . . .	539
<a href="#">CWRUtil::Q15</a>	
Defines a number with 15 bits reserved for fraction . . . . .	541
<a href="#">CWRUtil::Q5</a>	
Defines a number with 5 bits reserved for fraction . . . . .	543
<a href="#">Quadruple</a>	
Base class for homogeneous vectors and points . . . . .	548
<a href="#">RadioButton</a>	
Radio button with two states . . . . .	552
<a href="#">RadioButtonGroup&lt; CAPACITY &gt;</a>	
Class for handling a collection of RadioButtons . . . . .	558
<a href="#">Rasterizer</a>	
Polygon <a href="#">Rasterizer</a> that is used to render filled polygons with high-quality Anti- Aliasing . . . . .	563
<a href="#">Rect</a>	
Class representing a Rectangle with a few convenient methods . . . . .	567
<a href="#">Renderer</a>	
This class template is used basically for rendering scan lines . . . . .	572
<a href="#">RenderingBuffer</a>	
Rendering buffer wrapper . . . . .	573
<a href="#">RepeatButton</a>	
A button with two states . . . . .	577
<a href="#">RepeatButtonTrigger</a>	
A repeat button trigger . . . . .	580
<a href="#">ScalableImage</a>	
<a href="#">Widget</a> for representing a scaled version of a bitmap . . . . .	582

<a href="#">Scanline</a>	
This class is used to transfer data from class <a href="#">Outline</a> (or a similar one) to the rendering buffer . . .	586
<a href="#">Screen</a>	
A <a href="#">Screen</a> represents a full-screen drawable area. Applications create specific screens by subclassing this class . . . . .	590
<a href="#">ScrollableContainer</a>	
A <a href="#">ScrollableContainer</a> is a container that allows its contents to be scrolled . . . . .	596
<a href="#">ScrollBase</a>	
A scroll base class . . . . .	607
<a href="#">ScrollList</a>	
A scrolling menu of drawables . . . . .	622
<a href="#">ScrollWheel</a>	
A scroll wheel . . . . .	628
<a href="#">ScrollWheelBase</a>	
A scroll wheel base class . . . . .	629
<a href="#">ScrollWheelWithSelectionStyle</a>	
A scroll wheel with selection style . . . . .	633
<a href="#">SDL2TouchController</a>	
<a href="#">TouchController</a> for the simulator . . . . .	641
<a href="#">SDLTouchController</a>	
<a href="#">TouchController</a> for the simulator . . . . .	642
<a href="#">Shape&lt; POINTS &gt;</a>	
Simple widget capable of drawing a shape . . . . .	643
<a href="#">AbstractShape::ShapePoint&lt; T &gt;</a>	
Defines an alias representing the array of points making up the abstract shape . . . . .	647
<a href="#">SlideMenu</a>	
<a href="#">SlideMenu</a> is a container that has the functionality of being either collapsed or expanded . . .	647
<a href="#">Slider</a>	
A slider is a graphical element with which the user may set a value by moving an indicator or by clicking the slider . . . . .	657
<a href="#">SlideTransition&lt; templateDirection &gt;</a>	
A <a href="#">Transition</a> that slides from one screen to the next . . . . .	667
<a href="#">Snapper&lt; T &gt;</a>	
A mix-in that will make class T draggable and able to snap to a position . . . . .	670
<a href="#">SnapshotWidget</a>	
A widget that is able to make a snapshot of the area the <a href="#">SnapshotWidget</a> covers . . . . .	672
<a href="#">LCD::StringVisuals</a>	
The visual elements when writing a string . . . . .	675
<a href="#">SwipeContainer</a>	
A swipe container . . . . .	677
<a href="#">TextArea</a>	
This widget is capable of showing a text area on the screen . . . . .	681
<a href="#">TextAreaWithOneWildcard</a>	
<a href="#">TextArea</a> with one wildcard . . . . .	689
<a href="#">TextAreaWithTwoWildcards</a>	
<a href="#">TextArea</a> with two wildcards . . . . .	692
<a href="#">TextAreaWithWildcardBase</a>	
Base class for TextAreas displaying texts with one or more wildcards . . . . .	695
<a href="#">TextButtonStyle&lt; T &gt;</a>	
A text button style . . . . .	696
<a href="#">TextProgress</a>	
A text progress . . . . .	699
<a href="#">TextProvider</a>	
The <a href="#">TextProvider</a> is used in drawing basic and wildcard strings . . . . .	703
<a href="#">Texts</a>	
Class for setting language and getting texts . . . . .	705
<a href="#">TextureMapper</a>	
<a href="#">Widget</a> capable of drawing a transformed image . . . . .	706

<a href="#">TextureSurface</a>	
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 . . . . .	722
<a href="#">TiledImage</a>	
Simple widget capable of showing a tiled bitmap . . . . .	722
<a href="#">TiledImageButtonStyle&lt; T &gt;</a>	
A tiled image button style . . . . .	729
<a href="#">ToggleButton</a>	
A <a href="#">ToggleButton</a> is a <a href="#">Button</a> specialization that swaps the two bitmaps when clicked . . . . .	731
<a href="#">ToggleButtonTrigger</a>	
A toggle button trigger . . . . .	733
<a href="#">TouchArea</a>	
Invisible widget used to capture touch events . . . . .	735
<a href="#">TouchButtonTrigger</a>	
A touch button trigger . . . . .	737
<a href="#">TouchCalibration</a>	
Calibrates a touch coordinate . . . . .	738
<a href="#">TouchController</a>	
Basic Touch Controller interface . . . . .	739
<a href="#">Transition</a>	
Base class for Transitions . . . . .	740
<a href="#">TwoWildcardTextButtonStyle&lt; T &gt;</a>	
A wildcard text button style . . . . .	742
<a href="#">TypedText</a>	
<a href="#">TypedText</a> represents text (as in characters) and typography (as in font and alignment) . . . . .	746
<a href="#">TypedText::TypedTextData</a>	
The data structure for typed texts . . . . .	749
<a href="#">UIEventListener</a>	
This class declares a handler interface for user interface events . . . . .	750
<a href="#">Unicode</a>	
This class provides simple helper functions for working with 16-bit strings . . . . .	752
<a href="#">Vector&lt; T, capacity &gt;</a>	
A very simple container class using pre-allocated memory . . . . .	765
<a href="#">Vector4</a>	
This class represents a homogeneous 3D vector . . . . .	769
<a href="#">View&lt; T &gt;</a>	
This is a generic <a href="#">touchgfx::Screen</a> specialization for normal applications . . . . .	770
<a href="#">Widget</a>	
A <a href="#">Widget</a> is a <a href="#">Drawable</a> leaf (i.e. not a container) . . . . .	772
<a href="#">WildcardTextButtonStyle&lt; T &gt;</a>	
A wildcard text button style . . . . .	773
<a href="#">ZoomAnimationImage</a>	
Class for optimizing and wrapping move and zoom operations on ScalableImages . . . . .	777

## Chapter 6

# Namespace Documentation

### 6.1 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 [AbstractButtonContainer](#)  
*An abstract button container.*
- class [AbstractClock](#)  
*Superclass of clock widgets.*
- class [AbstractDirectionProgress](#)  
*An abstract direction progress.*
- 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 [AbstractPainterGRAY2](#)  
*A Painter that will paint using a color and an alpha value.*
- class [AbstractPainterGRAY4](#)  
*A Painter that will paint using a color and an alpha value.*
- 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 [AbstractProgressIndicator](#)  
*An abstract progress indicator.*
- class [AbstractShape](#)  
*Simple widget capable of drawing a [abstractShape](#).*
- class [AnalogClock](#)  
*An analog clock.*
- class [AnimatedImage](#)

- A widget capable of basic animation using a range of bitmaps.*

  - class [AnimatedImageButtonStyle](#)

*An animated image button style. An animated image button style. This class is supposed to be used with one of the [ButtonTrigger](#) classes to create a functional button. This class will show the first or last image of an animated image depending on the state of the button (pressed or released). When the state changes the button will show the sequence of images in forward or reversed order.*
  - class [AnimationTextureMapper](#)

*A texture mapper with animation capabilities.*
  - 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 [BoxProgress](#)

*A box progress.*
  - class [BoxWithBorder](#)

*A box with border.*
  - class [BoxWithBorderButtonStyle](#)

*A box with border button style.*
  - 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, T2, void >](#)

*A [Callback](#) is basically a wrapper of a pointer-to-member-function.*
  - struct [Callback< dest\\_type, T1, void, void >](#)

*A [Callback](#) is basically a wrapper of a pointer-to-member-function.*
  - struct [Callback< dest\\_type, void, 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 [CircleProgress](#)



- A circle progress.*
- class [ClickButtonTrigger](#)
  - A click button trigger.*
- 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 [CoverTransition](#)
  - A [Transition](#) that slides from one screen to the next.*
- struct [CWRUtil](#)
  - Helper classes and functions for [CanvasWidget](#).*
- class [DigitalClock](#)
  - A digital clock.*
- 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.*
- class [DrawableList](#)
  - A container able to display many items using only a few drawables.*
- class [DrawableListItems](#)
  - An array of drawables used by [DrawableList](#).*
- class [DrawableListItemsInterface](#)
  - A drawable list items interface.*
- 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, T2, void >](#)

[GenericCallback](#) is the base class for callbacks.
  - class [GenericCallback< T1, void, 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 [HALSDL2](#)

[HAL](#) implementation for the TouchGFX simulator.
  - class [I2C](#)

Platform independent interface for [I2C](#) drivers.
  - class [I2CTouchController](#)

Specific I2C-enabled type of Touch Controller.
  - class [IconButtonStyle](#)

An icon button style.
  - class [Image](#)

Simple widget capable of showing a bitmap.
  - class [ImageButtonStyle](#)

An image button style.
  - class [ImageProgress](#)

An image progress.
  - 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 [LCD2bpp](#)

*This class contains the various low-level drawing routines for drawing bitmaps.*
  - class [LCD4bpp](#)

*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 [LineProgress](#)

*A line progress.*
  - 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 [ModalWindow](#)

*[Container](#) for displaying a modal window and hijacking touch event to underlying view and widgets.*
  - 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 [PainterBWBitmap](#)

*A Painter that will paint using a bitmap.*
  - class [PainterGRAY2](#)

*A Painter that will paint using a color and an alpha value.*
  - class [PainterGRAY2Bitmap](#)

*A Painter that will paint using a bitmap.*
  - class [PainterGRAY4](#)

- A Painter that will paint using a color and an alpha value.*

  - class [PainterGRAY4Bitmap](#)
- A Painter that will paint using a bitmap.*

  - 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.*

  - class [RepeatButton](#)
- A button with two states.*

  - class [RepeatButtonTrigger](#)
- A repeat button trigger.*

  - 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 [ScrollBase](#)  
*A scroll base class.*
- class [ScrollList](#)  
*A scrolling menu of drawables.*
- class [ScrollWheel](#)  
*A scroll wheel.*
- class [ScrollWheelBase](#)  
*A scroll wheel base class.*
- class [ScrollWheelWithSelectionStyle](#)  
*A scroll wheel with selection style.*
- class [SDL2TouchController](#)  
*[TouchController](#) for the simulator.*
- class [SDLTouchController](#)  
*[TouchController](#) for the simulator.*
- class [Shape](#)  
*Simple widget capable of drawing a shape.*
- class [SlideMenu](#)  
*[SlideMenu](#) is a container that has the functionality of being either collapsed or expanded.*
- 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 [SwipeContainer](#)  
*A swipe container.*
- 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 [TextButtonStyle](#)  
*A text button style.*
- class [TextProgress](#)  
*A text progress.*
- 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 [TiledImage](#)  
*Simple widget capable of showing a tiled bitmap.*
- class [TiledImageButtonStyle](#)  
*A tiled image button style.*
- class [ToggleButton](#)  
*A [ToggleButton](#) is a [Button](#) specialization that swaps the two bitmaps when clicked.*
- class [ToggleButtonTrigger](#)  
*A toggle button trigger.*
- class [TouchArea](#)  
*Invisible widget used to capture touch events.*
- class [TouchButtonTrigger](#)  
*A touch button trigger.*
- class [TouchCalibration](#)  
*Calibrates a touch coordinate.*
- class [TouchController](#)  
*Basic Touch Controller interface.*
- class [Transition](#)  
*The [Transition](#) class is the base class for Transitions.*
- class [TwoWildcardTextButtonStyle](#)  
*A wildcard text button style.*
- 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 [WildcardTextButtonStyle](#)  
*A wildcard text button style.*
- 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 [BoxWithBorderButtonStyle](#)< [ClickButtonTrigger](#) > [BoxClickButton](#)  
*Defines an alias representing the box click button.*
- typedef [BoxWithBorderButtonStyle](#)< [RepeatButtonTrigger](#) > [BoxRepeatButton](#)  
*Defines an alias representing the box repeat button.*
- typedef [BoxWithBorderButtonStyle](#)< [ToggleButtonTrigger](#) > [BoxToggleButton](#)  
*Defines an alias representing the box toggle button.*
- typedef [BoxWithBorderButtonStyle](#)< [TouchButtonTrigger](#) > [BoxTouchButton](#)

- Defines an alias representing the box touch button.*

  - typedef [ImageButtonStyle](#)< [ClickButtonTrigger](#) > [ImageClickButton](#)

*Defines an alias representing the image click button.*
- typedef [ImageButtonStyle](#)< [RepeatButtonTrigger](#) > [ImageRepeatButton](#)

*Defines an alias representing the image repeat button.*
- typedef [ImageButtonStyle](#)< [TouchButtonTrigger](#) > [ImageTouchButton](#)

*Defines an alias representing the image touch button.*
- typedef [ImageButtonStyle](#)< [ToggleButtonTrigger](#) > [ImageToggleButton](#)

*Defines an alias representing the image toggle button.*
- typedef [IconButtonStyle](#)< [ClickButtonTrigger](#) > [IconClickButton](#)

*Defines an alias representing the icon click button.*
- typedef [IconButtonStyle](#)< [RepeatButtonTrigger](#) > [IconRepeatButton](#)

*Defines an alias representing the icon repeat button.*
- typedef [IconButtonStyle](#)< [TouchButtonTrigger](#) > [IconTouchButton](#)

*Defines an alias representing the icon touch button.*
- typedef [IconButtonStyle](#)< [ToggleButtonTrigger](#) > [IconToggleButton](#)

*Defines an alias representing the icon toggle button.*
- typedef [ImageButtonStyle](#)< [IconButtonStyle](#)< [ClickButtonTrigger](#) > > [IconImageClickButton](#)

*Defines an alias representing the icon image click button.*
- typedef [ImageButtonStyle](#)< [IconButtonStyle](#)< [RepeatButtonTrigger](#) > > [IconImageRepeatButton](#)

*Defines an alias representing the icon image repeat button.*
- typedef [ImageButtonStyle](#)< [IconButtonStyle](#)< [TouchButtonTrigger](#) > > [IconImageTouchButton](#)

*Defines an alias representing the icon image touch button.*
- typedef [ImageButtonStyle](#)< [IconButtonStyle](#)< [ToggleButtonTrigger](#) > > [IconImageToggleButton](#)

*Defines an alias representing the icon image toggle button.*
- typedef [TextButtonStyle](#)< [ClickButtonTrigger](#) > [TextClickButton](#)

*Defines an alias representing the text click button.*
- typedef [TextButtonStyle](#)< [RepeatButtonTrigger](#) > [TextRepeatButton](#)

*Defines an alias representing the text repeat button.*
- typedef [TextButtonStyle](#)< [TouchButtonTrigger](#) > [TextTouchButton](#)

*Defines an alias representing the text touch button.*
- typedef [TextButtonStyle](#)< [ToggleButtonTrigger](#) > [TextToggleButton](#)

*Defines an alias representing the text toggle button.*
- typedef [TiledImageButtonStyle](#)< [ClickButtonTrigger](#) > [TiledImageClickButton](#)

*Defines an alias representing the tiled image click button.*
- typedef [TiledImageButtonStyle](#)< [RepeatButtonTrigger](#) > [TiledImageRepeatButton](#)

*Defines an alias representing the tiled image repeat button.*
- typedef [TiledImageButtonStyle](#)< [TouchButtonTrigger](#) > [TiledImageTouchButton](#)

*Defines an alias representing the tiled image touch button.*
- typedef [TiledImageButtonStyle](#)< [ToggleButtonTrigger](#) > [TiledImageToggleButton](#)

*Defines an alias representing the tiled image toggle button.*
- typedef [WildcardTextButtonStyle](#)< [ClickButtonTrigger](#) > [WildcardTextClickButton](#)

*Defines an alias representing the wildcard text click button.*
- typedef [WildcardTextButtonStyle](#)< [RepeatButtonTrigger](#) > [WildcardTextRepeatButton](#)

*Defines an alias representing the wildcard text repeat button.*
- typedef [WildcardTextButtonStyle](#)< [TouchButtonTrigger](#) > [WildcardTextTouchButton](#)

*Defines an alias representing the wildcard text touch button.*
- typedef [WildcardTextButtonStyle](#)< [ToggleButtonTrigger](#) > [WildcardTextToggleButton](#)

*Defines an alias representing the wildcard text toggle button.*
- typedef [TwoWildcardTextButtonStyle](#)< [ClickButtonTrigger](#) > [TwoWildcardTextClickButton](#)

*Defines an alias representing the wildcard text click button.*

- typedef [TwoWildcardTextStyle](#)< [RepeatButtonTrigger](#) > [TwoWildcardTextRepeatButton](#)  
*Defines an alias representing the wildcard text repeat button.*
- typedef [TwoWildcardTextStyle](#)< [TouchButtonTrigger](#) > [TwoWildcardTextTouchButton](#)  
*Defines an alias representing the wildcard text touch button.*
- typedef [TwoWildcardTextStyle](#)< [ToggleButtonTrigger](#) > [TwoWildcardTextToggleButton](#)  
*Defines an alias representing the wildcard text toggle button.*
- typedef [AnimatedImageButtonStyle](#)< [ClickButtonTrigger](#) > [AnimatedImageClickButton](#)  
*Defines an alias representing the animated image click button.*
- typedef [AnimatedImageButtonStyle](#)< [RepeatButtonTrigger](#) > [AnimatedImageRepeatButton](#)  
*Defines an alias representing the animated image repeat button.*
- typedef [AnimatedImageButtonStyle](#)< [TouchButtonTrigger](#) > [AnimatedImageTouchButton](#)  
*Defines an alias representing the animated image touch button.*
- typedef [AnimatedImageButtonStyle](#)< [ToggleButtonTrigger](#) > [AnimatedImageToggleButton](#)  
*Defines an alias representing the animated image toggle button.*
- 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 uint16\_t [RenderingVariant](#)  
*Describes a combination of rendering algorithm, image format, and alpha information.*
- 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 [WideTextAction](#) { [WIDE\\_TEXT\\_NONE](#), [WIDE\\_TEXT\\_WORDWRAP](#), [WIDE\\_TEXT\\_WORDWRAP\\_ELLIPSIS\\_AFTER\\_SPACE](#), [WIDE\\_TEXT\\_CHARWRAP](#), [WIDE\\_TEXT\\_CHARWRAP\\_DOUBLE\\_ELLIPSIS](#) }  
*Defines how long text lines should be dealt with if the width exceeds that of the [TextArea](#).*

## 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.*
- FORCE\_INLINE\_FUNCTION int [LCD2shiftVal](#) (int offset)  
*Shift value to get the right pixel in a byte.*
- FORCE\_INLINE\_FUNCTION uint8\_t [LCD2getPixel](#) (const uint8\_t \*addr, int offset)  
*Get pixel from buffer/image.*
- FORCE\_INLINE\_FUNCTION uint8\_t [LCD2getPixel](#) (const uint16\_t \*addr, int offset)  
*Get pixel from buffer/image.*
- FORCE\_INLINE\_FUNCTION void [LCD2setPixel](#) (uint8\_t \*addr, int offset, uint8\_t value)  
*Set pixel in buffer.*
- FORCE\_INLINE\_FUNCTION void [LCD2setPixel](#) (uint16\_t \*addr, int offset, uint8\_t value)  
*Set pixel in buffer.*
- FORCE\_INLINE\_FUNCTION uint8\_t [LCD4getPixel](#) (const uint8\_t \*addr, int offset)  
*Get pixel from buffer/image.*
- FORCE\_INLINE\_FUNCTION uint8\_t [LCD4getPixel](#) (const uint16\_t \*addr, int offset)  
*Get pixel from buffer/image.*
- FORCE\_INLINE\_FUNCTION void [LCD4setPixel](#) (uint8\_t \*addr, int offset, uint8\_t value)  
*Set pixel in buffer.*
- FORCE\_INLINE\_FUNCTION void [LCD4setPixel](#) (uint16\_t \*addr, int offset, uint8\_t value)  
*Set pixel in buffer.*
- 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<typename T >
  - T [abs](#) (T d)
  - Simple implementation of the standard abs function.*
  - template<typename T >
  - T [gcd](#) (T a, T b)
  - Find greatest common divisor.*
  - int32\_t [clz](#) (int32\_t x)
  - Count leading zeros.*
  - int32\_t [muldiv](#) (int32\_t factor1, int32\_t factor2, int32\_t divisor, int32\_t &remainder)
  - Multiply and divide.*

## Variables

- const [BitmapId](#) [BITMAP\\_ANIMATION\\_STORAGE](#) = 0xFFFEU
- A virtual id representing animation storage.*
- 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 uint16\_t [RenderingVariant\\_NearestNeighbor](#) = 0
- The rendering variant nearest neighbor bit value.*
- static const uint16\_t [RenderingVariant\\_Bilinear](#) = 1
- The rendering variant bilinear bit value.*
- static const uint16\_t [RenderingVariant\\_NoAlpha](#) = 0

- The rendering variant no alpha bit value.*
  - static const uint16\_t [RenderingVariant\\_Alpha](#) = 2
- The rendering variant alpha bit value.*
  - static const uint16\_t [RenderingVariant\\_FormatShift](#) = 2
- The rendering variant format shift.*
  - static const float [PI](#) = 3.14159265358979323846f
- PI.*
  - const [TypedTextId](#) [TYPED\\_TEXT\\_INVALID](#) = 0xFFFFU
- The ID of an invalid text.*

## 6.1.1 Typedef Documentation

### 6.1.1.1 Alignment

uint8\_t [Alignment](#)

Defines an alignment type.

### 6.1.1.2 EasingEquation

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.

### 6.1.1.3 RenderingVariant

uint16\_t [RenderingVariant](#)

Describes a combination of rendering algorithm, image format, and alpha information. The lowest bit is 0 for "↔ Nearest neighbor", 1 for "Bilinear". The next bit is "0" for "no alpha", "2" for "alpha". The rest is the [Bitmap::Format](#) shifted up by 2.

### 6.1.1.4 TextDirection

uint8\_t [TextDirection](#)

Defines a the direction to write text.

## 6.1.2 Enumeration Type Documentation

### 6.1.2.1 BlitOperations

enum [BlitOperations](#)

The [BlitOp](#) operations.

Enumerator

<a href="#">BLIT_OP_COPY</a>	Copy the source to the destination.
------------------------------	-------------------------------------

**Enumerator**

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.

**6.1.2.2 Direction**

```
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.

**6.1.2.3 DisplayOrientation**

```
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.

**6.1.2.4 DisplayRotation**

```
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.

#### 6.1.2.5 FrameBuffer

enum enum [FrameBuffer](#)

Defines a FrameBuffer type.

#### Enumerator

FB_PRIMARY	First frame buffer.
FB_SECONDARY	Second frame buffer.
FB_TERTIARY	Third frame buffer.

#### 6.1.2.6 GlyphFlags

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".

#### 6.1.2.7 Gradient

enum enum [Gradient](#)

Defines a gradient type.

#### Enumerator

GRADIENT_HORIZONTAL	Horizontal gradient.
GRADIENT_VERTICAL	Vertical gradient.

### 6.1.2.8 TextRotation

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.

### 6.1.2.9 WideTextAction

enum enum [WideTextAction](#)

#### See also

[TextArea::setWidthTextAction](#)

#### Enumerator

WIDE_TEXT_NONE	Do nothing, simply cut the text in the middle of any character that extends beyond the width of the <a href="#">TextArea</a> .
WIDE_TEXT_WORDWRAP	Wrap between words, ellipsis anywhere "Very long t...".
WIDE_TEXT_WORDWRAP_ELLIPSIS_AFTER_SPACE	Wrap between words, ellipsis anywhere only after space "Very long ...".
WIDE_TEXT_CHARWRAP	Wrap between any two characters, ellipsis anywhere, as used in Chinese.
WIDE_TEXT_CHARWRAP_DOUBLE_ELLIPSIS	Wrap between any two characters, double ellipsis anywhere, as used in Chinese.

## 6.1.3 Function Documentation

### 6.1.3.1 abs()

```
template< class T > T abs (
    T d )
```

Simple implementation of the standard abs function.

#### Template Parameters

<i>T</i>	The type on which to perform the abs.
----------	---------------------------------------

## Parameters

<i>d</i>	The entity on which to perform the abs.
----------	-----------------------------------------

## Returns

The absolute (non-negative) value of d.

## 6.1.3.2 ceil28\_4()

```
int32_t ceil28_4 (  
    fixed28_4 value ) [inline]
```

## Parameters

<i>value</i>	The value.
--------------	------------

## Returns

The ceil result.

## 6.1.3.3 clz()

```
static int32_t clz (  
    int32_t x )
```

Count leading zeros in the binary representation of the given value.

## Parameters

<i>x</i>	The value to count the number of leading zeros in.
----------	----------------------------------------------------

## Returns

An int32\_t.

## 6.1.3.4 finalizeTransition()

```
static inline void finalizeTransition (  
    Screen * newScreen,  
    Presenter * newPresenter,  
    Transition * newTransition ) [inline], [static]
```

## Parameters

in	<i>newScreen</i>	If non-null, the new screen.
in	<i>newPresenter</i>	If non-null, the new presenter.
in	<i>newTransition</i>	If non-null, the new transition.

#### 6.1.3.5 fixed28\_4Mul()

```
fixed28_4 fixed28_4Mul (  
    fixed28_4 a,  
    fixed28_4 b ) [inline]
```

##### Parameters

<i>a</i>	The fixed28_4 to process.
<i>b</i>	The fixed28_4 to process.

##### Returns

the result.

#### 6.1.3.6 fixed28\_4ToFloat()

```
float fixed28_4ToFloat (  
    fixed28_4 value ) [inline]
```

##### Parameters

<i>value</i>	The value.
--------------	------------

##### Returns

The value as float.

#### 6.1.3.7 floatToFixed16\_16()

```
fixed16_16 floatToFixed16_16 (  
    float value ) [inline]
```

##### Parameters

<i>value</i>	The value.
--------------	------------

##### Returns

The value as fixed16\_16.

#### 6.1.3.8 floatToFixed28\_4()

```
fixed28_4 floatToFixed28_4 (  
    float value ) [inline]
```



## Parameters

<i>value</i>	The value.
--------------	------------

## Returns

The value as fixed28\_4.

## 6.1.3.9 floorDivMod()

```
void floorDivMod (
    int32_t numerator,
    int32_t denominator,
    int32_t & floor,
    int32_t & mod ) [inline]
```

## Parameters

	<i>numerator</i>	The numerator.
	<i>denominator</i>	The denominator.
in, out	<i>floor</i>	The floor.
in, out	<i>mod</i>	The modifier.

## 6.1.3.10 gcd()

```
template< typename T > T gcd (
    T a,
    T b )
```

Find greatest common divisor of two given numbers.

## Template Parameters

<i>T</i>	Generic type parameter.
----------	-------------------------

## Parameters

<i>a</i>	The first number.
<i>b</i>	The second number.

## Returns

A T.

## 6.1.3.11 hw\_init()

```
void hw_init ( )
```

Function to perform generic hardware initialization of the board. This function prototype is only provided as a convention.

#### 6.1.3.12 LCD2getPixel() [1/2]

```
FORCE_INLINE_FUNCTION uint8_t LCD2getPixel (  
    const uint8_t * addr,  
    int offset )
```

##### Parameters

<i>addr</i>	The address.
<i>offset</i>	The offset.

##### Returns

The pixel value.

#### 6.1.3.13 LCD2getPixel() [2/2]

```
FORCE_INLINE_FUNCTION uint8_t LCD2getPixel (  
    const uint16_t * addr,  
    int offset )
```

##### Parameters

<i>addr</i>	The address.
<i>offset</i>	The offset.

##### Returns

The pixel value.

#### 6.1.3.14 LCD2setPixel() [1/2]

```
FORCE_INLINE_FUNCTION void LCD2setPixel (  
    uint8_t * addr,  
    int offset,  
    uint8_t value )
```

##### Parameters

in	<i>addr</i>	The address.
	<i>offset</i>	The offset.
	<i>value</i>	The value.

**6.1.3.15 LCD2setPixel()** [2/2]

```
FORCE_INLINE_FUNCTION void LCD2setPixel (
    uint16_t * addr,
    int offset,
    uint8_t value )
```

**Parameters**

in	<i>addr</i>	The address.
	<i>offset</i>	The offset.
	<i>value</i>	The value.

**6.1.3.16 LCD2shiftVal()**

```
FORCE_INLINE_FUNCTION int LCD2shiftVal (
    int offset )
```

**Parameters**

<i>offset</i>	The offset.
---------------	-------------

**Returns**

The shift value.

**6.1.3.17 LCD4getPixel()** [1/2]

```
FORCE_INLINE_FUNCTION uint8_t LCD4getPixel (
    const uint8_t * addr,
    int offset )
```

**Parameters**

<i>addr</i>	The address.
<i>offset</i>	The offset.

**Returns**

The pixel value.

**6.1.3.18 LCD4getPixel()** [2/2]

```
FORCE_INLINE_FUNCTION uint8_t LCD4getPixel (
    const uint16_t * addr,
    int offset )
```

**Parameters**

<i>addr</i>	The address.
<i>offset</i>	The offset.

**Returns**

The pixel value.

**6.1.3.19 LCD4setPixel()** [1/2]

```
FORCE_INLINE_FUNCTION void LCD4setPixel (
    uint8_t * addr,
    int offset,
    uint8_t value )
```

**Parameters**

in	<i>addr</i>	The address.
	<i>offset</i>	The offset.
	<i>value</i>	The value.

**6.1.3.20 LCD4setPixel()** [2/2]

```
FORCE_INLINE_FUNCTION void LCD4setPixel (
    uint16_t * addr,
    int offset,
    uint8_t value )
```

**Parameters**

in	<i>addr</i>	The address.
	<i>offset</i>	The offset.
	<i>value</i>	The value.

**6.1.3.21 lookupBilinearRenderVariant()**

```
RenderingVariant lookupBilinearRenderVariant (
    const Bitmap & bitmap )
```

Returns the associated bilinear render variant based on the bitmap format.

**Parameters**

<i>bitmap</i>	The bitmap.
---------------	-------------

**Returns**

A RenderingVariant based on the bitmap format.

**6.1.3.22 lookupNearestNeighborRenderVariant()**

```
RenderingVariant lookupNearestNeighborRenderVariant (
    const Bitmap & bitmap )
```

Returns the associated nearest neighbor render variant based on the bitmap format.

**Parameters**

<i>bitmap</i>	The bitmap.
---------------	-------------

**Returns**

A RenderingVariant based on the bitmap format.

**6.1.3.23 makeTransition()**

```
template< class ScreenType, class PresenterType, class TransType, class ModelType > Presenter<
Type * 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**

<i>ScreenType</i>	Class type for the <a href="#">View</a> .
<i>PresenterType</i>	Class type for the <a href="#">Presenter</a> .
<i>TransType</i>	Class type for the <a href="#">Transition</a> .
<i>ModelType</i>	Class type for the Model.

**Parameters**

in	<i>currentScreen</i>	Pointer to pointer to the current view.
in	<i>currentPresenter</i>	Pointer to pointer to the current presenter.
in	<i>heap</i>	Reference to the heap containing the memory storage in which to allocate.
in	<i>currentTrans</i>	Pointer to pointer to the current transition.
in	<i>model</i>	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.

**6.1.3.24 memset()**

```
void memset (
    void * data,
    uint8_t c,
    uint32_t size )
```

Provides utility functions.

Simple implementation of the standard memset function.

**Parameters**

out	<i>data</i>	Address of data to set.
	<i>c</i>	Value to set.
	<i>size</i>	Number of bytes to set.

**6.1.3.25 muldiv()**

```
static int32_t muldiv (
    int32_t factor1,
    int32_t factor2,
    int32_t divisor,
    int32_t & remainder )
```

Multiply and divide without causing overflow. Multiplying two large values and subsequently dividing the result with another large value might cause an overflow in the intermediate result. The function [muldiv\(\)](#) will multiply the two first values and divide the result by the third value without causing overflow (unless the final result would overflow). The remainder from the calculation is also returned.

**Parameters**

	<i>factor1</i>	The first factor.
	<i>factor2</i>	The second factor.
	<i>divisor</i>	The divisor.
out	<i>remainder</i>	The remainder.

**Returns**

An int32\_t.

**6.1.3.26 operator\*() [1/2]**

```
Matrix4x4 operator* (
```

```
const Matrix4x4 & multiplicand,
const Matrix4x4 & multiplier )
```

Multiplication operator.

#### Parameters

<i><a href="#">multiplicand</a></i>	The first value to multiply.
<i><a href="#">multiplier</a></i>	The second value to multiply.

#### Returns

The result of the operation.

#### 6.1.3.27 operator\*() [2/2]

```
Point4 operator* (
    const Matrix4x4 & multiplicand,
    const Point4 & multiplier )
```

Multiplication operator.

#### Parameters

<i><a href="#">multiplicand</a></i>	The first value to multiply.
<i><a href="#">multiplier</a></i>	The second value to multiply.

#### Returns

The result of the operation.

#### 6.1.3.28 prepareTransition()

```
static inline void prepareTransition (
    Screen ** currentScreen,
    Presenter ** currentPresenter,
    Transition ** currentTrans ) [inline], [static]
```

#### Parameters

in	<i><a href="#">currentScreen</a></i>	If non-null, the current screen.
in	<i><a href="#">currentPresenter</a></i>	If non-null, the current presenter.
in	<i><a href="#">currentTrans</a></i>	If non-null, the current transaction.

#### 6.1.3.29 touchgfx\_generic\_init()

```
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

<i>HALType</i>	The class type of the <a href="#">HAL</a> subclass used for this port.
----------------	------------------------------------------------------------------------

## Parameters

in	<i>dma</i>	Reference to the DMA implementation object to use. Can be of type <a href="#">NoDMA</a> to disable the use of DMA for rendering.
in	<i>display</i>	Reference to the <a href="#">LCD</a> renderer implementation (subclass of <a href="#">LCD</a> ). Could be either <a href="#">LCD16bpp</a> for RGB565 UIs, or <a href="#">LCD1bpp</a> for monochrome UIs or <a href="#">LCD24bpp</a> for 24bit displays using RGB888 UIs.
in	<i>tc</i>	Reference to the touch controller driver (or <a href="#">NoTouchController</a> to disable touch input).
	<i>width</i>	The <i>native</i> 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.
	<i>height</i>	The <i>native</i> 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	<i>bitmapCache</i>	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.
	<i>bitmapCacheSize</i>	Size of bitmap cache in bytes. Pass 0 if bitmap cache is not used.
	<i>numberOfDynamicBitmaps</i>	Number of dynamic bitmaps.

## Returns

A reference to the allocated (and initialized) [HAL](#) object.

### 6.1.3.30 touchgfx\_init()

```
void touchgfx_init ( )
```

Function to perform touchgfx initialization. This function prototype is only provided as a convention.

## 6.1.4 Variable Documentation



#### 6.1.4.1 TYPED\_TEXT\_INVALID

```
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 7

# Class Documentation

### 7.1 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

##### 7.1.1 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](#)

## 7.1.2 Constructor & Destructor Documentation

### 7.1.2.1 `AbstractButton()`

```
AbstractButton ( ) [inline]
```

Constructs an [AbstractButton](#) instance in released state without an associated action.

## 7.1.3 Member Function Documentation

### 7.1.3.1 `getPressedState()`

```
bool getPressedState ( ) const [inline], [virtual]
```

Function to determine whether this [AbstractButton](#) is currently pressed.

#### Returns

true if button is pressed, false otherwise.

### 7.1.3.2 `getType()`

```
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](#).

### 7.1.3.3 `handleClickEvent()`

```
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

<i>event</i>	Information about the click.
--------------	------------------------------

See also

[Drawable::handleClickEvent\(\)](#)

Reimplemented from [Drawable](#).

Reimplemented in [RepeatButton](#), [ToggleButton](#), [RadioButton](#), and [TouchArea](#).

#### 7.1.3.4 setAction()

```
void setAction (
    GenericCallback< const AbstractButton & > & callback ) [inline]
```

Associates an action to be performed when the [AbstractButton](#) is clicked.

Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">AbstractButton</a> .
-----------------	-------------------------------------------------------------------------------------------------------------

See also

[GenericCallback](#)

## 7.2 AbstractButtonContainer Class Reference

An abstract button container.

```
#include <touchgfx/containers/buttons/AbstractButtonContainer.hpp>
```

### Public Member Functions

- [AbstractButtonContainer](#) ()  
*Default constructor.*
- virtual [~AbstractButtonContainer](#) ()  
*Destructor.*
- void [setPressed](#) (bool isPressed)  
*Sets the pressed state.*
- bool [getPressed](#) ()  
*Gets the pressed state.*
- void [setAlpha](#) (uint8\_t newAlpha)  
*Sets an alpha value.*
- uint8\_t [getAlpha](#) () const  
*Gets the alpha.*
- void [setAction](#) (GenericCallback< const [AbstractButtonContainer](#) & > &callback)  
*Sets an action callback.*

### Protected Member Functions

- virtual void [handlePressedUpdated](#) ()  
*Handles the pressed updated.*
- virtual void [handleAlphaUpdated](#) ()  
*Handles the alpha updated.*

## Protected Attributes

- bool [pressed](#)  
*True if pressed.*
- uint8\_t [alpha](#)  
*The current alpha value. 255 denotes solid, 0 denotes completely transparent.*
- [GenericCallback](#)< const [AbstractButtonContainer](#) &> \* [action](#)  
*The action.*

## Additional Inherited Members

### 7.2.1 Detailed Description

An abstract button container. It defines pressed/not pressed state, the alpha value, and the action [Callback](#) of a button. [AbstractButtonContainer](#) is used as superclass for classes defining a specific button behaviour.

See also

[ClickButtonTrigger](#), [RepeatButtonTrigger](#), [ToggleButtonTrigger](#), [TouchButtonTrigger](#)

### 7.2.2 Member Function Documentation

#### 7.2.2.1 getAlpha()

```
uint8_t getAlpha ( ) const [inline]
```

Returns

The alpha value.

#### 7.2.2.2 getPressed()

```
bool getPressed ( ) [inline]
```

Returns

True if it succeeds, false if it fails.

#### 7.2.2.3 setAction()

```
void setAction (
    GenericCallback< const AbstractButtonContainer & > & callback ) [inline]
```

Parameters

<i>callback</i>	The callback.
-----------------	---------------

## 7.2.2.4 setAlpha()

```
void setAlpha (
    uint8_t newAlpha ) [inline]
```

## Parameters

<i>newAlpha</i>	The new alpha.
-----------------	----------------

## 7.2.2.5 setPressed()

```
void setPressed (
    bool isPressed ) [inline]
```

## Parameters

<i>isPressed</i>	True if is pressed, false if not.
------------------	-----------------------------------

## 7.3 AbstractClock Class Reference

Superclass of clock widgets.

```
#include <touchgfx/containers/clock/AbstractClock.hpp>
```

## Public Member Functions

- [AbstractClock](#) ()  
*Default constructor.*
- virtual [~AbstractClock](#) ()  
*Destructor.*
- virtual void [setTime24Hour](#) (uint8\_t hour, uint8\_t minute, uint8\_t second)  
*Sets the time with input format as 24H.*
- virtual void [setTime12Hour](#) (uint8\_t hour, uint8\_t minute, uint8\_t second, bool am)  
*Sets the time with input format as 12H.*
- uint8\_t [getCurrentHour](#) () const  
*Gets the current hour.*
- uint8\_t [getCurrentMinute](#) () const  
*Gets the current minute.*
- uint8\_t [getCurrentSecond](#) () const  
*Gets the current second.*

## Protected Member Functions

- virtual void [updateClock](#) ()=0  
*Updates the visual representation of the clock.*

## Protected Attributes

- `uint8_t currentHour`  
*Local copy of the current hour.*
- `uint8_t currentMinute`  
*Local copy of the current minute.*
- `uint8_t currentSecond`  
*Local copy of the current second.*

## Additional Inherited Members

### 7.3.1 Constructor & Destructor Documentation

#### 7.3.1.1 AbstractClock()

`AbstractClock ( )`

Default constructor.

#### 7.3.1.2 ~AbstractClock()

`~AbstractClock ( )` [inline], [virtual]

Destructor.

### 7.3.2 Member Function Documentation

#### 7.3.2.1 getCurrentHour()

`uint8_t getCurrentHour ( ) const`

Gets the current hour.

##### Returns

The current hour.

#### 7.3.2.2 getCurrentMinute()

`uint8_t getCurrentMinute ( ) const`

Gets the current minute.

##### Returns

The current minute.



#### 7.3.2.3 getCurrentSecond()

```
uint8_t getCurrentSecond ( ) const
```

Gets the current second.

##### Returns

The current second.

#### 7.3.2.4 setTime12Hour()

```
void setTime12Hour (
    uint8_t hour,
    uint8_t minute,
    uint8_t second,
    bool am ) [virtual]
```

Sets the time with input format as 12H. Note that this does not affect any selected presentation formats.

##### Parameters

<i>hour</i>	The hours (in 12H format).
<i>minute</i>	The minutes (in 12H format).
<i>second</i>	The seconds (in 12H format).
<i>am</i>	AM/PM setting. True = AM, false = PM.

#### 7.3.2.5 setTime24Hour()

```
void setTime24Hour (
    uint8_t hour,
    uint8_t minute,
    uint8_t second ) [virtual]
```

Sets the time with input format as 24H. Note that this does not affect any selected presentation formats.

##### Parameters

<i>hour</i>	The hours (in 24H format).
<i>minute</i>	The minutes (in 24H format).
<i>second</i>	The seconds (in 24H format).

#### 7.3.2.6 updateClock()

```
void updateClock ( ) [protected], [pure virtual]
```

Updates the visual representation of the clock.

Implemented in [AnalogClock](#), and [DigitalClock](#).

## 7.4 AbstractDirectionProgress Class Reference

An abstract direction progress.

```
#include <touchgfx/containers/progress_indicators/AbstractDirectionProgress.h>
```

### Public Types

- enum [DirectionType](#) { **RIGHT**, **LEFT**, **DOWN**, **UP** }  
*Values that represent directions.*

### Public Member Functions

- [AbstractDirectionProgress](#) ()  
*Default constructor.*
- virtual [~AbstractDirectionProgress](#) ()  
*Destructor.*
- virtual void [setDirection](#) ([DirectionType](#) direction)  
*Sets a direction.*
- virtual [DirectionType](#) [getDirection](#) () const  
*Gets the direction.*

### Protected Attributes

- [DirectionType](#) [progressDirection](#)  
*The progress direction.*

### Additional Inherited Members

#### 7.4.1 Detailed Description

An abstract direction progress for progress indicators that need a direction to be specified.

#### 7.4.2 Member Enumeration Documentation

##### 7.4.2.1 DirectionType

```
enum enum DirectionType
```

Values that represent directions.

#### 7.4.3 Constructor & Destructor Documentation

## 7.4.3.1 AbstractDirectionProgress()

```
AbstractDirectionProgress ( )
```

Default constructor.

## 7.4.3.2 ~AbstractDirectionProgress()

```
~AbstractDirectionProgress ( ) [virtual]
```

Destructor.

## 7.4.4 Member Function Documentation

## 7.4.4.1 getDirection()

```
DirectionType getDirection ( ) const [virtual]
```

Gets the direction.

## Returns

The direction.

## 7.4.4.2 setDirection()

```
void setDirection (
    DirectionType direction ) [virtual]
```

Sets a direction.

## Parameters

<i>direction</i>	The direction.
------------------	----------------

## 7.5 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.*

- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)=0  
*Paint a designated part of the [RenderingBuffer](#).*

### Protected Member Functions

- void [setWidgetAlpha](#) (uint8\_t alpha)  
*Sets widget alpha.*

### 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.*

## 7.5.1 Detailed Description

An abstract class for creating painter classes for drawing canvas widgets.

## 7.5.2 Constructor & Destructor Documentation

### 7.5.2.1 AbstractPainter()

```
AbstractPainter ( )
```

Default constructor.

### 7.5.2.2 ~AbstractPainter()

```
~AbstractPainter ( ) [virtual]
```

Destructor.

## 7.5.3 Member Function Documentation

### 7.5.3.1 render()

```
void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels futher).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Implemented in [PainterGRAY2](#), [PainterGRAY4](#), [PainterRGB565](#), [PainterRGB888](#), [PainterGRAY2Bitmap](#), [PainterGRAY4Bitmap](#), [PainterRGB565Bitmap](#), [PainterRGB888Bitmap](#), [PainterBW](#), [PainterBWBitmap](#), [AbstractPainterRGB565](#), [AbstractPainterBW](#), [AbstractPainterGRAY2](#), [AbstractPainterGRAY4](#), and [AbstractPainterRGB888](#).

7.5.3.2 `setOffset()`

```
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

<i>offsetX</i>	The offset x coordinate of the invalidated area relative to the widget.
<i>offsetY</i>	The offset y coordinate of the invalidated area relative to the widget.

7.5.3.3 `setWidgetAlpha()`

```
void setWidgetAlpha (
    uint8_t alpha ) [protected]
```

Sets the widget alpha to allow an entire canvas widget to easily be faded without changing the painter of the widget.

## Parameters

<i>alpha</i>	The alpha.
--------------	------------

## Note

Used internally by [Canvas](#).

7.6 **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 xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

## 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.*

### 7.6.1 Detailed Description

[AbstractPainterBW](#) is used for drawing one 1bpp displays. The color is either on or off No transparency is supported.

See also

[AbstractPainter](#)

### 7.6.2 Member Function Documentation

#### 7.6.2.1 [render\(\)](#)

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels futher).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Implements [AbstractPainter](#).

Reimplemented in [PainterBW](#), and [PainterBWBitmap](#).

### 7.6.2.2 renderInit()

```
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 [PainterBWBitmap](#).

### 7.6.2.3 renderNext()

```
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](#), and [PainterBWBitmap](#).

## 7.7 AbstractPainterGRAY2 Class Reference

A Painter that will paint using a color and an alpha value.

```
#include <touchgfx/widgets/canvas/AbstractPainterGRAY2.hpp>
```

### Public Member Functions

- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

### Protected Member Functions

- virtual bool [renderInit](#) ()  
*Initialize rendering of a single scan line of pixels for the render.*
- virtual bool [renderNext](#) (uint8\_t &gray, uint8\_t &alpha)=0  
*Get the color of the next pixel in the scan line.*
- virtual void [renderPixel](#) (uint8\_t \*p, uint16\_t offset, uint8\_t gray)  
*Renders the pixel.*



## Protected Attributes

- int [currentX](#)  
*Current x coordinate relative to the widget.*
- int [currentY](#)  
*Current y coordinate relative to the widget.*

### 7.7.1 Detailed Description

The [AbstractPainterGRAY2](#) 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](#)

### 7.7.2 Member Function Documentation

#### 7.7.2.1 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Implements [AbstractPainter](#).

Reimplemented in [PainterGRAY2](#), and [PainterGRAY2Bitmap](#).

#### 7.7.2.2 renderInit()

```
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 [PainterGRAY2Bitmap](#).

**7.7.2.3 renderNext()**

```
bool renderNext (
    uint8_t & gray,
    uint8_t & alpha ) [protected], [pure virtual]
```

Get the color of the next pixel in the scan line.

**Parameters**

out	<i>gray</i>	The gray (0-15).
out	<i>alpha</i>	The alpha.

**Returns**

true if the pixel should be painted, false otherwise.

Implemented in [PainterGRAY2](#), and [PainterGRAY2Bitmap](#).

**7.7.2.4 renderPixel()**

```
void renderPixel (
    uint8_t * p,
    uint16_t offset,
    uint8_t gray ) [protected], [virtual]
```

Renders the pixel into the frame buffer.

**Parameters**

in	<i>p</i>	pointer into the frame buffer line where the given pixel should be written.
	<i>offset</i>	The offset to the pixel from the given pointer.
	<i>gray</i>	The green color.

param blue The blue color.

**7.8 AbstractPainterGRAY4 Class Reference**

A Painter that will paint using a color and an alpha value.

```
#include <touchgfx/widgets/canvas/AbstractPainterGRAY4.hpp>
```

**Public Member Functions**

- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)

Paint a designated part of the [RenderingBuffer](#).

## Protected Member Functions

- virtual bool [renderInit](#) ()  
*Initialize rendering of a single scan line of pixels for the render.*
- virtual bool [renderNext](#) (uint8\_t &gray, uint8\_t &alpha)=0  
*Get the color of the next pixel in the scan line.*
- virtual void [renderPixel](#) (uint8\_t \*p, uint16\_t offset, uint8\_t gray)  
*Renders the pixel.*

## Protected Attributes

- int [currentX](#)  
*Current x coordinate relative to the widget.*
- int [currentY](#)  
*Current y coordinate relative to the widget.*

### 7.8.1 Detailed Description

The [AbstractPainterGRAY4](#) 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](#)

### 7.8.2 Member Function Documentation

#### 7.8.2.1 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Implements [AbstractPainter](#).

Reimplemented in [PainterGRAY4](#), and [PainterGRAY4Bitmap](#).

#### 7.8.2.2 renderInit()

```
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 [PainterGRAY4Bitmap](#).

#### 7.8.2.3 renderNext()

```
bool renderNext (
    uint8_t & gray,
    uint8_t & alpha ) [protected], [pure virtual]
```

Get the color of the next pixel in the scan line.

##### Parameters

out	<i>gray</i>	The gray (0-15).
out	<i>alpha</i>	The alpha.

##### Returns

true if the pixel should be painted, false otherwise.

Implemented in [PainterGRAY4](#), and [PainterGRAY4Bitmap](#).

#### 7.8.2.4 renderPixel()

```
void renderPixel (
    uint8_t * p,
    uint16_t offset,
    uint8_t gray ) [protected], [virtual]
```

Renders the pixel into the frame buffer.

##### Parameters

in	<i>p</i>	pointer into the frame buffer line where the given pixel should be written.
	<i>offset</i>	The offset to the pixel from the given pointer.
	<i>gray</i>	The green color.

param blue The blue color.

## 7.9 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 xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*
- FORCE\_INLINE\_FUNCTION uint16\_t [mixColors](#) (uint16\_t newpix, uint16\_t bufpix, uint16\_t alpha, uint16\_t shift)  
*Mix colors.*
- FORCE\_INLINE\_FUNCTION uint16\_t [mixColors](#) (uint16\_t R, uint16\_t G, uint16\_t B, uint16\_t bufpix, uint16\_t alpha, uint16\_t shift)  
*Mix colors.*

### Static Public Attributes

- static const uint16\_t [RMASK](#) = 0xF800  
*Mask for red (1111100000000000)*
- static const uint16\_t [GMASK](#) = 0x07E0  
*Mask for green (00001111100000)*
- static const uint16\_t [BMASK](#) = 0x001F  
*Mask for blue (00000000001111)*

### 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.*

#### 7.9.1 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](#)

## 7.9.2 Member Function Documentation

### 7.9.2.1 mixColors() [1/2]

```
FORCE_INLINE_FUNCTION uint16_t mixColors (
    uint16_t newpix,
    uint16_t bufpix,
    uint16_t alpha,
    uint16_t shift ) [inline]
```

Mix colors from a new pixel and a buffer pixel with the given alpha applied to the new pixel.

#### Parameters

<i>newpix</i>	The newpix value.
<i>bufpix</i>	The bufpix value.
<i>alpha</i>	The alpha of the newpix.
<i>shift</i>	The shift, 8 if alpha is [0..255], 16 if alpha is [0..255*255].

#### Returns

The new color to write to the frame buffer.

### 7.9.2.2 mixColors() [2/2]

```
FORCE_INLINE_FUNCTION uint16_t mixColors (
    uint16_t R,
    uint16_t G,
    uint16_t B,
    uint16_t bufpix,
    uint16_t alpha,
    uint16_t shift ) [inline]
```

Mix colors from a new pixel and a buffer pixel with the given alpha applied to the new pixel.

#### Parameters

<i>R</i>	The red color (placed in 0xF800).
<i>G</i>	The green color (placed in 0x03E0).
<i>B</i>	The blue color (placed in 0x001F).
<i>bufpix</i>	The bufpix value.
<i>alpha</i>	The alpha of the newpix.
<i>shift</i>	The shift, 8 if alpha is [0..255], 16 if alpha is [0..255*255].

#### Returns

The new color to write to the frame buffer.

## 7.9.2.3 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Implements [AbstractPainter](#).

Reimplemented in [PainterRGB565](#), and [PainterRGB565Bitmap](#).

## 7.9.2.4 renderInit()

```
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](#).

## 7.9.2.5 renderNext()

```
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	<i>red</i>	The red.
out	<i>green</i>	The green.
out	<i>blue</i>	The blue.
out	<i>alpha</i>	The alpha.

**Returns**

true if the pixel should be painted, false otherwise.

Implemented in [PainterRGB565](#), and [PainterRGB565Bitmap](#).

**7.9.2.6 renderPixel()**

```
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	<i>p</i>	pointer into the frame buffer where the given color should be written.
	<i>red</i>	The red color.
	<i>green</i>	The green color.
	<i>blue</i>	The blue color.

**7.10 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 xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

**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.*



### 7.10.1 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](#)

### 7.10.2 Member Function Documentation

#### 7.10.2.1 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Implements [AbstractPainter](#).

Reimplemented in [PainterRGB888](#), and [PainterRGB888Bitmap](#).

#### 7.10.2.2 renderInit()

```
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](#).

### 7.10.2.3 renderNext()

```
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	<i>red</i>	The red.
out	<i>green</i>	The green.
out	<i>blue</i>	The blue.
out	<i>alpha</i>	The alpha.

#### Returns

true if the pixel should be painted, false otherwise.

Implemented in [PainterRGB888](#), and [PainterRGB888Bitmap](#).

### 7.10.2.4 renderPixel()

```
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	<i>p</i>	pointer into the frame buffer where the given color should be written.
	<i>red</i>	The red color.
	<i>green</i>	The green color.
	<i>blue</i>	The blue color.

## 7.11 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.*

### 7.11.1 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.

### 7.11.2 Constructor & Destructor Documentation

### 7.11.2.1 ~AbstractPartition()

```
~AbstractPartition ( ) [virtual]
```

Virtual destructor.

### 7.11.2.2 AbstractPartition()

```
AbstractPartition ( ) [protected]
```

Default constructor.

## 7.11.3 Member Function Documentation

### 7.11.3.1 allocate() [1/2]

```
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

<i>size</i>	The size.
-------------	-----------

#### Returns

The address of an empty storage slot which contains minimum 'size' bytes.

### 7.11.3.2 allocate() [2/2]

```
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

<i>T</i>	Generic type parameter.
----------	-------------------------

**Returns**

The address of an empty storage slot.

**7.11.3.3 allocateAt()** [1/2]

```
void * allocateAt (
    uint16_t index,
    uint16_t size ) [virtual]
```

Gets the address of the specified index.

**Note**

Asserts if 'size' is too large.

**Parameters**

<i>index</i>	Zero-based index of the.
<i>size</i>	The size.

**Returns**

The address of the appropriate storage slot which contains minimum 'size' bytes.

**7.11.3.4 allocateAt()** [2/2]

```
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**

<i>T</i>	Generic type parameter.
----------	-------------------------

**Parameters**

<i>index</i>	Zero-based index of the.
--------------	--------------------------

**Returns**

The address of the appropriate storage slot.

**7.11.3.5 at()** [1/2]

```
template< typename T > T & at (
    const uint16_t index ) [inline]
```

Gets the object at the specified index.

**Template Parameters**

<i>T</i>	Generic type parameter.
----------	-------------------------

**Parameters**

<i>index</i>	The index into the <a href="#">Partition</a> storage where the returned object is located.
--------------	--------------------------------------------------------------------------------------------

**Returns**

A typed reference to the object at the specified index.

**7.11.3.6 at()** [2/2]

```
template< typename T > const T & at (
    const uint16_t index ) const [inline]
```

const version of [at\(\)](#).

**Template Parameters**

<i>T</i>	Generic type parameter.
----------	-------------------------

**Parameters**

<i>index</i>	Zero-based index of the.
--------------	--------------------------

**Returns**

A T&

**7.11.3.7 capacity()**

```
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 >](#).

## 7.11.3.8 clear()

```
void clear ( ) [virtual]
```

Prepares the [Partition](#) for new allocations. Any objects present in the [Partition](#) shall not be used after invoking this method.

## 7.11.3.9 dec()

```
void dec ( ) [inline]
```

Decreases number of allocations.

## 7.11.3.10 element() [1/2]

```
void * element (
    uint16_t index ) [protected], [pure virtual]
```

Access to stored element. Used internally.

## Parameters

<i>index</i>	Zero-based index of the.
--------------	--------------------------

## Returns

null if it fails, else a void\*.

Implemented in [Partition< ListOfTypes, NUMBER\\_OF\\_ELEMENTS >](#).

## 7.11.3.11 element() [2/2]

```
const void * element (
    uint16_t index ) const [protected], [pure virtual]
```

Access to stored element, const version.

## Parameters

<i>index</i>	Zero-based index of the.
--------------	--------------------------

## Returns

null if it fails, else a void\*.

Implemented in [Partition< ListOfTypes, NUMBER\\_OF\\_ELEMENTS >](#).

## 7.11.3.12 element\_size()

```
uint32_t element_size ( ) [pure virtual]
```

Access to concrete element-size. Used internally.

**Returns**

An uint32\_t.

Implemented in [Partition< ListOfTypes, NUMBER\\_OF\\_ELEMENTS >](#).

**7.11.3.13 find()**

```
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**

<i>T</i>	Generic type parameter.
----------	-------------------------

**Parameters**

<i>pT</i>	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.

**7.11.3.14 getAllocationCount()**

```
uint16_t getAllocationCount ( ) const [virtual]
```

Gets allocation count.

**Returns**

The currently allocated storage slots.

**7.11.3.15 indexOf()**

```
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**

<i>address</i>	The location address to lookup.
----------------	---------------------------------



## Returns

An uint16\_t.

## 7.12 AbstractProgressIndicator Class Reference

An abstract progress indicator.

```
#include <touchgfx/containers/progress_indicators/AbstractProgressIndicator.h>
hpp>
```

### Public Member Functions

- [AbstractProgressIndicator](#) ()  
*Default constructor.*
- virtual [~AbstractProgressIndicator](#) ()  
*Destructor.*
- virtual void [setBackground](#) (const [Bitmap](#) &bmpBackground)  
*Sets the background image.*
- virtual void [setProgressIndicatorPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets the position and dimension of the actual progress indicator.*
- virtual int16\_t [getProgressIndicatorX](#) () const  
*Gets progress indicator x coordinate.*
- virtual int16\_t [getProgressIndicatorY](#) () const  
*Gets progress indicator y coordinate.*
- virtual int16\_t [getProgressIndicatorWidth](#) () const  
*Gets progress indicator width.*
- virtual int16\_t [getProgressIndicatorHeight](#) () const  
*Gets progress indicator height.*
- virtual void [setRange](#) (int16\_t min, int16\_t max, uint16\_t steps=0, uint16\_t minStep=0)  
*Sets the range for the progress indicator.*
- virtual void [getRange](#) (int16\_t &min, int16\_t &max, uint16\_t &steps, uint16\_t &minStep) const  
*Gets the range.*
- virtual void [getRange](#) (int16\_t &min, int16\_t &max, uint16\_t &steps) const  
*Gets the range.*
- virtual void [getRange](#) (int16\_t &min, int16\_t &max) const  
*Gets the range.*
- virtual void [setValue](#) (int value)  
*Sets a value.*
- virtual int [getValue](#) () const  
*Gets the value.*
- virtual uint16\_t [getProgress](#) (uint16\_t range=100) const  
*Gets the progress.*

### Protected Attributes

- [Image](#) background  
*The background image.*
- [Container](#) progressIndicatorContainer  
*The container that holds the actual progress indicator.*
- int16\_t rangeMin

*The range minimum.*

- `int16_t` [rangeMax](#)

*The range maximum.*

- `uint16_t` [currentValue](#)

*The current value.*

- `uint16_t` [rangeSteps](#)

*The range steps.*

- `uint16_t` [rangeStepsMin](#)

*The range steps minimum.*

## Additional Inherited Members

### 7.12.1 Detailed Description

The [AbstractProgressIndicator](#) declares methods that provides the basic mechanisms and tools to implement a progress indicator. For more specific implementations see classes that inherit from [AbstractProgressIndicator](#).

See also

[BoxProgress](#)  
[CircleProgress](#)  
[ImageProgress](#)  
[LineProgress](#)  
[TextProgress](#)

### 7.12.2 Constructor & Destructor Documentation

#### 7.12.2.1 AbstractProgressIndicator()

[AbstractProgressIndicator](#) ( )

Initialized the progress indicator with default range 0-100.

See also

[setRange](#)

#### 7.12.2.2 ~AbstractProgressIndicator()

[~AbstractProgressIndicator](#) ( ) [virtual]

Destructor.

### 7.12.3 Member Function Documentation

### 7.12.3.1 getProgress()

```
uint16_t getProgress (
    uint16_t range = 100 ) const [virtual]
```

Gets the current progress based on the range set by [setRange\(\)](#) and the value set by [setValue\(\)](#).

#### Parameters

<i>range</i>	The range.
--------------	------------

#### Returns

The progress.

#### See also

[setRange](#), [setValue](#)

### 7.12.3.2 getProgressIndicatorHeight()

```
int16_t getProgressIndicatorHeight ( ) const [virtual]
```

Gets progress indicator height.

#### Returns

The progress indicator height.

### 7.12.3.3 getProgressIndicatorWidth()

```
int16_t getProgressIndicatorWidth ( ) const [virtual]
```

Gets progress indicator width.

#### Returns

The progress indicator width.

### 7.12.3.4 getProgressIndicatorX()

```
int16_t getProgressIndicatorX ( ) const [virtual]
```

Gets progress indicator x coordinate.

#### Returns

The progress indicator x coordinate.

### 7.12.3.5 `getProgressIndicatorY()`

```
int16_t getProgressIndicatorY ( ) const [virtual]
```

Gets progress indicator y coordinate.

#### Returns

The progress indicator y coordinate.

### 7.12.3.6 `getRange()` [1/3]

```
void getRange (
    int16_t & min,
    int16_t & max,
    uint16_t & steps,
    uint16_t & minStep ) const [virtual]
```

Gets the range set by [setRange\(\)](#).

#### Parameters

out	<i>min</i>	The minimum input value.
out	<i>max</i>	The maximum input value.
out	<i>steps</i>	The steps in which to report progress.
out	<i>minStep</i>	The step which the minimum input value is mapped to.

### 7.12.3.7 `getRange()` [2/3]

```
void getRange (
    int16_t & min,
    int16_t & max,
    uint16_t & steps ) const [virtual]
```

Gets the range set by [setRange\(\)](#).

#### Parameters

out	<i>min</i>	The minimum input value.
out	<i>max</i>	The maximum input value.
out	<i>steps</i>	The steps in which to report progress.

### 7.12.3.8 `getRange()` [3/3]

```
void getRange (
    int16_t & min,
    int16_t & max ) const [virtual]
```

Gets the range set by [setRange\(\)](#).

## Parameters

out	<i>min</i>	The minimum input value.
out	<i>max</i>	The maximum input value.

## 7.12.3.9 getValue()

```
int getValue ( ) const [virtual]
```

Gets the current value set by [setValue\(\)](#).

## Returns

The value.

## 7.12.3.10 setBackground()

```
void setBackground (
    const Bitmap & bmpBackground ) [virtual]
```

Sets the background image. The width and height of the widget is updated according to the dimension of the image.

## Parameters

<i>bmpBackground</i>	The background bitmap.
----------------------	------------------------

## 7.12.3.11 setProgressIndicatorPosition()

```
void setProgressIndicatorPosition (
    int16_t x,
    int16_t y,
    int16_t width,
    int16_t height ) [virtual]
```

Sets the position and dimension of the actual progress indicator relative to the background image.

## Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>width</i>	The width of the box progress indicator.
<i>height</i>	The height of the box progress indicator.

Reimplemented in [LineProgress](#), [CircleProgress](#), [BoxProgress](#), [ImageProgress](#), and [TextProgress](#).

### 7.12.3.12 setRange()

```
void setRange (
    int16_t min,
    int16_t max,
    uint16_t steps = 0,
    uint16_t minStep = 0 ) [virtual]
```

Sets the range for the progress indicator. The range is the values that are given to the progress indicator while progressing through the task at hand. If an app needs to work through 237 items to finish a task, the range should be set to (0, 237) assuming that 0 items is the minimum. Though the minimum is often 0, it is possible to customize this.

The steps parameter, is used to specify at what granularity you want the progress indicator to report a new progress value. If the 237 items to be reported as 0%, 10%, 20%, ... 100%, the steps should be set to 10 as there are ten steps from 0% to 100%. If you want to update a widget which is 150 pixels wide, you might want to set steps to 150 to get a new progress value for every pixel. If you are updating a clock and want this to resemble an analog clock, you might want to use 12 or perhaps 60 as number of steps.

The minStep parameter is used when the value min should give a progress different from 0. For example, if progress is a clock face, you want to count from 0..1000 and you want progress per minute, but want to make sure that 0 is not a blank clock face, but instead you want 1 minute to show, the setRange(0, 1000, 60, 1) will make sure that as values progress from 0 to 1000, [getProgress\(\)](#) start from 1 and goes up to 60. Another example could be a [BoxProgress](#) with a [TextProgress](#) on top and you want to make sure that "0%" will always show in the box, use something like setRange(0, 1000, 200, 40) if your box is 200 wide and "0%" is 40 wide.

#### Parameters

<i>min</i>	The minimum input value.
<i>max</i>	The maximum input value.
<i>steps</i>	The steps in which to report progress.
<i>minStep</i>	The step which the minimum input value is mapped to.

#### See also

[setValue](#), [getProgress](#)

### 7.12.3.13 setValue()

```
void setValue (
    int value ) [virtual]
```

Sets the current value in the range (min..max) set by [setRange\(\)](#). Values lower than min are mapped to min, values higher than max are mapped to max.

#### Parameters

<i>value</i>	The value.
--------------	------------

Reimplemented in [CircleProgress](#), [LineProgress](#), [ImageProgress](#), [TextProgress](#), and [BoxProgress](#).

## 7.13 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.*
- template<typename T >  
void [setShape](#) ([ShapePoint](#)< T > \*points)  
*Sets a shape the struct Points.*
- template<typename T >  
void [setOrigin](#) (T x, T y)  
*Sets the position of (0,0).*
- template<typename T >  
void [moveOrigin](#) (T x, T y)  
*Moves the start point for this [AbstractShape](#).*
- template<typename T >  
void [getOrigin](#) (T &dx, T &dy) const  
*Gets the start coordinates for the line.*
- template<typename T >  
void [setAngle](#) (T angle)  
*Sets the angle to turn the abstractShape.*
- template<typename T >  
void [updateAngle](#) (T angle)  
*Sets the angle to turn the abstractShape.*
- int [getAngle](#) () const  
*Gets the current angle of the abstractShape.*
- template<typename T >  
void [setScale](#) (T newXScale, T newYScale)  
*Scale the [AbstractShape](#).*
- template<typename T >  
void [setScale](#) (T scale)  
*Scale the [AbstractShape](#).*
- template<typename T >  
void [getScale](#) (T &x, T &y) const  
*Gets the x scale and y scale.*
- virtual bool [drawCanvasWidget](#) (const [Rect](#) &invalidatedArea) const  
*Draws the [AbstractShape](#).*
- void [updateAbstractShapeCache](#) ()  
*Updates the abstractShape cache.*

## Protected Member Functions

- 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.*
- virtual `Rect` `getMinimalRect` () const  
*Gets minimal rectangle containing the abstractShape.*

## Additional Inherited Members

### 7.13.1 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.

### 7.13.2 Constructor & Destructor Documentation

#### 7.13.2.1 AbstractShape()

`AbstractShape` ( )

Constructs a new [AbstractShape](#).

#### 7.13.2.2 ~AbstractShape()

`~AbstractShape` ( ) [virtual]

Virtual Destructor.

### 7.13.3 Member Function Documentation

#### 7.13.3.1 drawCanvasWidget()

```
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

<code>invalidatedArea</code>	The rectangle to draw, with coordinates relative to this drawable.
------------------------------	--------------------------------------------------------------------



**Returns**

true if it succeeds, false if it fails.

Implements [CanvasWidget](#).

**7.13.3.2 getAngle()**

```
template< typename T > T getAngle ( ) const [inline]
```

Gets the current angle of the abstractShape.

**Returns**

The angle of the [AbstractShape](#).

**7.13.3.3 getCacheX()**

```
CWRUtil::Q5 getCacheX (
    int i ) const [protected], [pure virtual]
```

Gets cached x coordinate of a corner.

**Parameters**

<i>i</i>	Zero-based index of the corner.
----------	---------------------------------

**Returns**

The cached x coordinate.

Implemented in [Shape< POINTS >](#).

**7.13.3.4 getCacheY()**

```
CWRUtil::Q5 getCacheY (
    int i ) const [protected], [pure virtual]
```

Gets cached y coordinate of a corner.

**Parameters**

<i>i</i>	Zero-based index of the corner.
----------	---------------------------------

**Returns**

The cached y coordinate.

Implemented in [Shape< POINTS >](#).

#### 7.13.3.5 getCornerX()

```
CWRUtil::Q5 getCornerX (  
    int i ) const [pure virtual]
```

Gets the x coordinate of a corner.

##### Parameters

<i>i</i>	Zero-based index of the corner.
----------	---------------------------------

##### Returns

The corner x coordinate.

Implemented in [Shape< POINTS >](#).

#### 7.13.3.6 getCornerY()

```
CWRUtil::Q5 getCornerY (  
    int i ) const [pure virtual]
```

Gets the y coordinate of a corner.

##### Parameters

<i>i</i>	Zero-based index of the corner.
----------	---------------------------------

##### Returns

The corner y coordinate.

Implemented in [Shape< POINTS >](#).

#### 7.13.3.7 getMinimalRect()

```
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](#).

#### 7.13.3.8 getNumPoints()

```
int getNumPoints ( ) const [pure virtual]
```

Gets number points used to make up the shape.

**Returns**

The number points.

Implemented in [Shape< POINTS >](#).

**7.13.3.9 getOrigin()**

```
template< typename T > void getOrigin (
    T & dx,
    T & dy ) const [inline]
```

Gets the start coordinates for the line.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

out	<i>dx</i>	The x coordinate.
out	<i>dy</i>	The y coordinate.

**7.13.3.10 getScale()**

```
template< typename T > void getScale (
    T & x,
    T & y ) const [inline]
```

Gets the x scale and y scale of the shape as previously set using setScale. Default is 1 for both x scale and y scale.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

out	<i>x</i>	Scaling of x coordinates.
out	<i>y</i>	Scaling of y coordinates.

**See also**

[setScale](#)

**7.13.3.11 moveOrigin()**

```
template< typename 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.

#### Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

#### Parameters

<i>x</i>	The x coordinate of the shapes position (0,0).
<i>y</i>	The y coordinate of the shapes position (0,0).

#### Note

The area containing the [AbstractShape](#) is invalidated before and after the change.

#### See also

[setOrigin\(\)](#)

#### 7.13.3.12 setAngle()

```
template< typename T > void setAngle (
    T angle ) [inline]
```

Sets the angle to turn the abstractShape. 0 degrees is straight up and 90 degrees is 3 o'clock.

#### Template Parameters

<i>T</i>	Generic type parameter.
----------	-------------------------

#### Parameters

<i>angle</i>	The angle to turn the abstractShape to relative to 0 (straight up), not relative to the previous angle.
--------------	---------------------------------------------------------------------------------------------------------

#### Note

The area containing the [AbstractShape](#) is not invalidated.

#### See also

[updateAngle\(\)](#)

#### 7.13.3.13 setCache()

```
void setCache (
    int i,
```

```
CWRUtil::Q5 x,
CWRUtil::Q5 y ) [protected], [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>i</i>	Zero-based index of the corner.
<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

Implemented in [Shape< POINTS >](#).

#### 7.13.3.14 setCorner()

```
void setCorner (
    int i,
    CWRUtil::Q5 x,
    CWRUtil::Q5 y ) [pure virtual]
```

Sets a corner of the shape in Q5 format.

#### Parameters

<i>i</i>	Zero-based index of the corner.
<i>x</i>	The x coordinate in Q5 format.
<i>y</i>	The y coordinate in Q5 format.

#### Note

Remember to call [updateAbstractShapeCache\(\)](#) to make sure that the cached outline of the shape is correct.

#### See also

[updateAbstractShapeCache](#)

Implemented in [Shape< POINTS >](#).

#### 7.13.3.15 setOrigin()

```
template< typename 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.

#### Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

<i>x</i>	The x coordinate of the shapes position (0,0).
<i>y</i>	The y coordinate of the shapes position (0,0).

## Note

The area containing the [AbstractShape](#) is not invalidated.

## See also

[moveOrigin\(\)](#)

7.13.3.16 **setScale()** [1/2]

```
template< typename T > void setScale (
    T newXScale,
    T newYScale ) [inline]
```

Scale the [AbstractShape](#) the given amounts in the x direction and the y direction.

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

<i>newXScale</i>	The new scale in the x direction.
<i>newYScale</i>	The new scale in the y direction.

## Note

The area containing the [AbstractShape](#) is not invalidated.

## See also

[getScale](#), [updateScale](#)

7.13.3.17 **setScale()** [2/2]

```
template< typename T > void setScale (
    T scale ) [inline]
```

Scale the [AbstractShape](#) the given amount in the x direction and the y direction.

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

<i>scale</i>	The scale in the x direction.
--------------	-------------------------------

## Note

The area containing the [AbstractShape](#) is not invalidated.

## See also

[getScale](#)

## 7.13.3.18 setShape()

```
template< typename T > void setShape (
    ShapePoint< T > * points ) [inline]
```

Sets a shape the struct Points. The cached outline of the shape is automatically updated.

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

in	<i>points</i>	The points that make up the shape.
----	---------------	------------------------------------

## Note

The area containing the shape is not invalidated.

## 7.13.3.19 updateAbstractShapeCache()

```
void updateAbstractShapeCache ( )
```

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).

## 7.13.3.20 updateAngle()

```
template< typename T > void updateAngle (
    T angle ) [inline]
```

Sets the angle to turn the abstractShape. 0 degrees is straight up and 90 degrees is 3 o'clock.

## Template Parameters

<i>T</i>	Generic type parameter.
----------	-------------------------

## Parameters

<i>angle</i>	The angle to turn the abstractShape.
--------------	--------------------------------------

## Note

The area containing the [AbstractShape](#) is invalidated before and after the change.

## See also

[setAngle\(\)](#)

## 7.14 AnalogClock Class Reference

An analog clock.

```
#include <touchgfx/containers/clock/AnalogClock.hpp>
```

### Public Member Functions

- [AnalogClock](#) ()  
*Default constructor.*
- virtual [~AnalogClock](#) ()  
*Destructor.*
- virtual void [setBackground](#) (const [BitmapId](#) backgroundBitmapId)  
*Sets the background image of the clock.*
- virtual void [setBackground](#) (const [BitmapId](#) backgroundBitmapId, int16\_t rotationCenterX, int16\_t rotationCenterY)  
*Sets the background image of the clock and the rotation center of the clock.*
- virtual void [setRotationCenter](#) (int16\_t rotationCenterX, int16\_t rotationCenterY)  
*Sets the rotation center of the clock.*
- virtual void [setupHourHand](#) (const [BitmapId](#) hourHandBitmapId, int16\_t rotationCenterX, int16\_t rotationCenterY)  
*Sets up the hour hand.*
- virtual void [setupMinuteHand](#) (const [BitmapId](#) minuteHandBitmapId, int16\_t rotationCenterX, int16\_t rotationCenterY)  
*Sets up the minute hand.*
- virtual void [setupSecondHand](#) (const [BitmapId](#) secondHandBitmapId, int16\_t rotationCenterX, int16\_t rotationCenterY)  
*Sets up the second hand.*
- virtual void [setHourHandMinuteCorrection](#) (bool active)  
*Sets hour hand minute correction.*
- virtual bool [getHourHandMinuteCorrection](#) () const  
*Gets hour hand minute correction.*
- virtual void [setMinuteHandSecondCorrection](#) (bool active)  
*Sets minute hand second correction.*
- virtual bool [getMinuteHandSecondCorrection](#) () const  
*Gets minute hand second correction.*
- virtual void [setAnimation](#) (uint16\_t duration=10, [EasingEquation](#) animationProgressionEquation=[EasingEquations::backEaseInOut](#))  
*Setup the clock to use animation for hand movements.*



- virtual uint16\_t [getAnimationDuration](#) ()  
*Gets the animation duration.*
- virtual void [initializeTime24Hour](#) (uint8\_t hour, uint8\_t minute, uint8\_t second)  
*Sets the time with input format as 24H. No animations are performed.*
- virtual void [initializeTime12Hour](#) (uint8\_t hour, uint8\_t minute, uint8\_t second, bool am)  
*Sets the time with input format as 12H. No animations are performed.*

### Protected Member Functions

- virtual void [updateClock](#) ()  
*Updates the visual representation of the clock.*
- virtual void [setupHand](#) (TextureMapper &hand, const [BitmapId](#) bitmapId, int16\_t rotationCenterX, int16\_t rotationCenterY)  
*Sets up a given the hand.*
- virtual float [convertHandValueToAngle](#) (uint8\_t steps, uint8\_t handValue, uint8\_t secondHandValue=0) const  
*Convert hand value to angle.*
- virtual bool [animationEnabled](#) () const  
*Is animation enabled.*

### Protected Attributes

- [Image](#) background  
*The background image of the [AnalogClock](#).*
- [AnimationTextureMapper](#) hourHand  
*[TextureMapper](#) that represents the hourHand.*
- [AnimationTextureMapper](#) minuteHand  
*[TextureMapper](#) that represents the minuteHand.*
- [AnimationTextureMapper](#) secondHand  
*[TextureMapper](#) that represents the secondHand.*
- [EasingEquation](#) animationEquation  
*The easing equation used by hand animations.*
- uint16\_t [animationDuration](#)  
*The duration of hand animations. If 0 animations are disabled.*
- int16\_t [clockRotationCenterX](#)  
*The rotation point (X) of the hands.*
- int16\_t [clockRotationCenterY](#)  
*The rotation point (Y) of the hands.*
- uint8\_t [lastHour](#)  
*The last know hour value.*
- uint8\_t [lastMinute](#)  
*The last know minute value.*
- uint8\_t [lastSecond](#)  
*The last know second value.*
- bool [hourHandMinuteCorrectionActive](#)  
*Is hour hand minute correction active.*
- bool [minuteHandSecondCorrectionActive](#)  
*Is minute hand second correction active.*

## Additional Inherited Members

### 7.14.1 Detailed Description

An analog clock. Should be supplied with images for the background, hour hand, minute hand and the optional second hand. You setup the [AnalogClock](#) by specifying the rotation point of each hand as well as the global rotation point of the clock.

You can customize the behavior of the [AnalogClock](#) in respect to animations and relations between the hands e.g. the hour hand moves gradually towards the next hour as the minute hand progresses ([setHourHandMinuteCorrection\(\)](#))

### 7.14.2 Member Function Documentation

#### 7.14.2.1 animationEnabled()

```
bool animationEnabled ( ) const [protected], [virtual]
```

Is animation enabled.

#### Returns

true if animation is enabled.

#### 7.14.2.2 convertHandValueToAngle()

```
float convertHandValueToAngle (
    uint8_t steps,
    uint8_t handValue,
    uint8_t secondHandValue = 0 ) const [protected], [virtual]
```

Convert hand value to angle.

#### Parameters

<i>steps</i>	Number of steps the primary hand value is divided into (e.g. 60 for minutes/seconds and 12 for hour).
<i>handValue</i>	The actual value for the hand in question (in the range [0;steps]).
<i>secondHandValue</i>	(Optional) If the angle should be corrected for a secondary hand its value should be specified here (in the range [0;60]). This is the case when <a href="#">setHourHandMinuteCorrection(true)</a> or <a href="#">setMinuteHandSecondCorrection(true)</a> is selected.

#### Returns

The converted value to angle.

#### 7.14.2.3 getAnimationDuration()

```
uint16_t getAnimationDuration ( ) [inline], [virtual]
```

**Returns**

The animation duration.

**7.14.2.4 getHourHandMinuteCorrection()**

```
bool getHourHandMinuteCorrection ( ) const [virtual]
```

Gets hour hand minute correction.

**Returns**

true if hour hand minute correction is active.

**See also**

[setHourHandMinuteCorrection](#)

**7.14.2.5 getMinuteHandSecondCorrection()**

```
bool getMinuteHandSecondCorrection ( ) const [virtual]
```

Gets minute hand second correction.

**Returns**

true if minute hand second correction is active.

**See also**

[setHourHandMinuteCorrection](#)

**7.14.2.6 initializeTime12Hour()**

```
void initializeTime12Hour (
    uint8_t hour,
    uint8_t minute,
    uint8_t second,
    bool am ) [virtual]
```

Sets the time with input format as 12H. No animations are performed regardless of the animation settings. This is often useful when setting up the [AnalogClock](#) where you do not want an initial animation. Note that this does not affect any selected presentation formats.

**Parameters**

<i>hour</i>	The hours (in 12H format).
<i>minute</i>	The minutes (in 12H format).
<i>second</i>	The seconds (in 12H format).
<i>am</i>	AM/PM setting. True = AM, false = PM.

#### 7.14.2.7 initializeTime24Hour()

```
void initializeTime24Hour (
    uint8_t hour,
    uint8_t minute,
    uint8_t second ) [virtual]
```

Sets the time with input format as 24H. No animations are performed regardless of the animation settings. This is often useful when setting up the [AnalogClock](#) where you do not want an initial animation. Note that this does not affect any selected presentation formats.

##### Parameters

<i>hour</i>	The hours (in 24H format).
<i>minute</i>	The minutes (in 24H format).
<i>second</i>	The seconds (in 24H format).

#### 7.14.2.8 setAnimation()

```
void setAnimation (
    uint16_t duration = 10,
    EasingEquation animationProgressionEquation = EasingEquations::backEaseInOut )
[virtual]
```

##### Parameters

<i>duration</i>	(Optional) The animation duration.
<i>animationProgressionEquation</i>	(Optional) The animation progression equation.

#### 7.14.2.9 setBackground() [1/2]

```
void setBackground (
    const BitmapId backgroundBitmapId ) [virtual]
```

Sets the background image of the clock. The clock rotation center is set to the background image center. The clock rotation center is the point that the clock hands rotates around.

##### Parameters

<i>background</i> ↵ <i>BitmapId</i>	Identifier for the background bitmap.
----------------------------------------	---------------------------------------

#### 7.14.2.10 setBackground() [2/2]

```
void setBackground (
```

```
const BitmapId backgroundBitmapId,  
int16_t rotationCenterX,  
int16_t rotationCenterY ) [virtual]
```

Sets the background image of the clock and the rotation center of the clock. The clock rotation center is the point that the clock hands rotates around.

#### Parameters

<i>background</i> ↵ <i>BitmapId</i>	Identifier for the background bitmap.
<i>rotationCenterX</i>	The rotation center x coordinate.
<i>rotationCenterY</i>	The rotation center y coordinate.

#### 7.14.2.11 setHourHandMinuteCorrection()

```
void setHourHandMinuteCorrection (  
    bool active ) [virtual]
```

If set to true the hour hand will be positioned between the current hour and the next depending on the minute hands position.

#### Parameters

<i>active</i>	true to use hour hand correction.
---------------	-----------------------------------

#### See also

[getHourHandMinuteCorrection](#)

#### 7.14.2.12 setMinuteHandSecondCorrection()

```
void setMinuteHandSecondCorrection (  
    bool active ) [virtual]
```

If set to true the minute hand will be positioned between the current minute and the next depending on the second hands position.

#### Parameters

<i>active</i>	true to use.
---------------	--------------

#### See also

[setMinuteHandSecondCorrection](#)

#### 7.14.2.13 setRotationCenter()

```
void setRotationCenter (  

```

```

    int16_t rotationCenterX,
    int16_t rotationCenterY ) [virtual]

```

Sets the rotation center of the clock. The clock rotation center is the point that the clock hands rotates around.

#### Parameters

<i>rotationCenterX</i>	The rotation center x coordinate.
<i>rotationCenterY</i>	The rotation center y coordinate.

#### 7.14.2.14 setupHand()

```

void setupHand (
    TextureMapper & hand,
    const BitmapId bitmapId,
    int16_t rotationCenterX,
    int16_t rotationCenterY ) [protected], [virtual]

```

Sets up a given the hand.

#### Parameters

in, out	<i>hand</i>	Reference to the hand being setup.
	<i>bitmapId</i>	The bitmap identifier for the given hand.
	<i>rotationCenterX</i>	The hand rotation center x coordinate.
	<i>rotationCenterY</i>	The hand rotation center y coordinate.

#### 7.14.2.15 setupHourHand()

```

void setupHourHand (
    const BitmapId hourHandBitmapId,
    int16_t rotationCenterX,
    int16_t rotationCenterY ) [virtual]

```

Sets up the hour hand. The specified rotation center is the point of the hand that is to be placed on top of the clock rotation center. That is the point that the hand rotates around. The rotation point is relative to the supplied bitmap but can be placed outside it.

If not called the hour hand will just be omitted.

#### Parameters

<i>hourHand</i> ↔ <i>BitmapId</i>	Identifier for the hour hand bitmap.
<i>rotationCenterX</i>	The hand rotation center x coordinate.
<i>rotationCenterY</i>	The hand rotation center y coordinate.

7.14.2.16 `setupMinuteHand()`

```
void setupMinuteHand (
    const BitmapId minuteHandBitmapId,
    int16_t rotationCenterX,
    int16_t rotationCenterY ) [virtual]
```

Sets up the minute hand. The specified rotation center is the point of the hand that is to be placed on top of the clock rotation center. That is the point that the hand rotates around. The rotation point is relative to the supplied bitmap but can be placed outside it.

If not called the minute hand will just be omitted.

## Parameters

<i>minuteHandBitmapId</i>	Identifier for the minute hand bitmap.
<i>rotationCenterX</i>	The hand rotation center x coordinate.
<i>rotationCenterY</i>	The hand rotation center y coordinate.

7.14.2.17 `setupSecondHand()`

```
void setupSecondHand (
    const BitmapId secondHandBitmapId,
    int16_t rotationCenterX,
    int16_t rotationCenterY ) [virtual]
```

Sets up the second hand. The specified rotation center is the point of the hand that is to be placed on top of the clock rotation center. That is the point that the hand rotates around. The rotation point is relative to the supplied bitmap but can be placed outside it.

If not called the second hand will just be omitted.

## Parameters

<i>secondHandBitmapId</i>	Identifier for the second hand bitmap.
<i>rotationCenterX</i>	The hand rotation center x coordinate.
<i>rotationCenterY</i>	The hand rotation center y coordinate.

7.14.2.18 `updateClock()`

```
virtual void updateClock ( ) [protected], [virtual]
```

Updates the visual representation of the clock.

Implements [AbstractClock](#).

## 7.15 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 [isAnimatedImageRunning](#) ()  
*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 Member Functions

- virtual void [setBitmap](#) (const [Bitmap](#) &bmp)  
*Is not public available. Use [setBitmaps](#) instead.*

## 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

### 7.15.1 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](#) for a demonstration of how to use of this widget.

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 AnimatedImage() [1/2]

```
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

<i>start</i>	Defines the start of the range of images in the animation.
<i>end</i>	Defines the end of the range of images in the animation.
<i>updateInterval</i>	Defines the number of ticks between each animation step. Higher value results in a slower animation.

#### 7.15.2.2 AnimatedImage() [2/2]

```
AnimatedImage (
    const uint8_t & updateInterval = 1 ) [inline]
```

Constructs an [AnimatedImage](#) without initializing bitmaps.

##### Note

The bitmaps to display must be configured through `setBitmaps` function before this widget displays anything.

##### Parameters

<i>updateInterval</i>	Defines the number of ticks between each animation step. Higher value results in a slower animation.
-----------------------	------------------------------------------------------------------------------------------------------

### 7.15.3 Member Function Documentation

#### 7.15.3.1 getType()

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

##### Returns

TYPE\_ANIMATEDIMAGE.

Reimplemented from [Image](#).

#### 7.15.3.2 handleTickEvent()

```
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](#).

#### 7.15.3.3 isAnimatedImageRunning()

```
bool isAnimatedImageRunning ( ) [inline]
```

Gets the running state of the [AnimatedImage](#).

##### Returns

true if the animation is currently running, false otherwise.

#### 7.15.3.4 isReverse()

```
bool isReverse ( ) [inline]
```

Query if this object is running in reverse.

##### Returns

true if the animation is performed in reverse order.

#### 7.15.3.5 isRunning()

```
bool isRunning ( ) [inline]
```

Gets the running state of the [AnimatedImage](#).

##### Returns

true if the animation is currently running, false otherwise.

**7.15.3.6 pauseAnimation()**

```
void pauseAnimation ( ) [virtual]
```

Toggles the running state of an animation. Pauses the animation if the animation is running. Continues the animation if previously paused.

**7.15.3.7 setBitmap()**

```
void setBitmap (
    const Bitmap & bmp ) [inline], [protected], [virtual]
```

Is not public available. Use setBitmaps instead. Internally in [AnimatedImage](#) use [Image::setBitmap\(...\)](#).

**Parameters**

<i>bmp</i>	The bitmap.
------------	-------------

Reimplemented from [Image](#).

**7.15.3.8 setBitmaps()**

```
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**

<i>start</i>	Defines the start of the range of images in the animation.
<i>end</i>	Defines the end of the range of images in the animation.

**7.15.3.9 setDoneAction()**

```
void setDoneAction (
    GenericCallback< const AnimatedImage & > & callback ) [inline]
```

Associates an action to be performed when the animation of the [AnimatedImage](#) is done.

**Parameters**

<i>callback</i>	The callback is executed when done. The callback is given the animated image.
-----------------	-------------------------------------------------------------------------------

**7.15.3.10 setUpdateTicksInterval()**

```
void setUpdateTicksInterval (
    uint8_t updateInterval )
```

Sets the update interval.

#### Parameters

<i>updateInterval</i>	Defines the number of ticks between each animation step. Higher value results in a slower animation.
-----------------------	------------------------------------------------------------------------------------------------------

#### 7.15.3.11 startAnimation()

```
void startAnimation (
    const bool & rev,
    const bool & reset = false,
    const bool & loop = false ) [virtual]
```

Starts the animation.

#### Parameters

<i>rev</i>	Defines if the animation should be performed in reverse order.
<i>reset</i>	Defines if the animation should reset and start from the first (or last if reverse order) bitmap.
<i>loop</i>	Defines if the animation should loop or do a single animation.

#### 7.15.3.12 stopAnimation()

```
void stopAnimation ( ) [virtual]
```

Stops and resets the animation.

## 7.16 AnimatedImageButtonStyle< T > Class Template Reference

An animated image button style. An animated image button style. This class is supposed to be used with one of the ButtonTrigger classes to create a functional button. This class will show the first or last image of an animated image depending on the state of the button (pressed or released). When the state changes the button will show the sequence of images in forward or reversed order.

```
#include <touchgfx/containers/buttons/AnimatedImageButtonStyle.hpp>
```

### Public Member Functions

- [AnimatedImageButtonStyle \(\)](#)  
*Default constructor.*
- [virtual ~AnimatedImageButtonStyle \(\)](#)  
*Destructor.*
- void [setBitmaps](#) (const [Bitmap](#) &bmpStart, const [Bitmap](#) &bmpEnd)  
*Sets the bitmaps.*
- void [setBitmapXY](#) (uint16\_t x, uint16\_t y)  
*Sets bitmap xy.*
- void [setUpdateTicksInterval](#) (uint8\_t updateInterval)  
*Sets update ticks interval.*

## Protected Member Functions

- virtual void [handlePressedUpdated](#) ()  
*Handles the pressed updated.*
- virtual void [handleAlphaUpdated](#) ()  
*Handles the alpha updated.*

## Protected Attributes

- [AnimatedImage](#) [buttonAnimatedImage](#)  
*The button animated image.*

### 7.16.1 Detailed Description

```
template<class T>
class touchgfx::AnimatedImageButtonStyle< T >
```

The [AnimatedImageButtonStyle](#) will set the size of the enclosing container (normally [AbstractButtonContainer](#)) to the size of the first [Bitmap](#). This can be overridden by calling `setWidth/setHeight` after setting the bitmaps.

The position of the bitmap can be adjusted with `setBitmapXY` (default is upper left corner).

#### Template Parameters

<i>T</i>	Generic type parameter. Typically a <a href="#">AbstractButtonContainer</a> subclass.
----------	---------------------------------------------------------------------------------------

#### See also

[AbstractButtonContainer](#)

## 7.16.2 Member Function Documentation

### 7.16.2.1 setBitmaps()

```
void setBitmaps (
    const Bitmap & bmpStart,
    const Bitmap & bmpEnd ) [inline]
```

#### Parameters

<i>bmpStart</i>	The bitmap start.
<i>bmpEnd</i>	The bitmap end.

### 7.16.2.2 setBitmapXY()

```
void setBitmapXY (
    uint16_t x,
    uint16_t y ) [inline]
```

## Parameters

<i>x</i>	An uint16_t to process.
<i>y</i>	An uint16_t to process.

## 7.16.2.3 setUpdateTicksInterval()

```
void setUpdateTicksInterval (
    uint8_t updateInterval ) [inline]
```

## Parameters

<i>updateInterval</i>	The update interval.
-----------------------	----------------------

## 7.17 AnimationTextureMapper::AnimationSetting Struct Reference

Information about how a specific animation parameter should be animated.

```
#include <touchgfx/widgets/AnimationTextureMapper.hpp>
```

## Public Attributes

- bool [animationActive](#)  
*Should this animation be performed.*
- float [animationStart](#)  
*The animation start value.*
- float [animationEnd](#)  
*The animation end value.*
- uint16\_t [animationDelay](#)  
*A delay that is applied before animation start. Expressed in ticks.*
- uint16\_t [animationDuration](#)  
*The complete duration of the animation. Expressed in ticks.*
- [EasingEquation](#) [animationProgressionEquation](#)  
*EasingEquation expressing the development of the value during the animation.*

## 7.18 AnimationTextureMapper Class Reference

A texture mapper with animation capabilities.

```
#include <touchgfx/widgets/AnimationTextureMapper.hpp>
```

## Classes

- struct [AnimationSetting](#)  
*Information about how a specific animation parameter should be animated.*

## Public Types

- enum [AnimationParameter](#) { X\_ROTATION = 0, Y\_ROTATION, Z\_ROTATION, SCALE }  
*Values that represent different animation parameter.*

## Public Member Functions

- [AnimationTextureMapper](#) ()  
*Default constructor.*
- virtual [~AnimationTextureMapper](#) ()  
*Destructor.*
- void [setTextureMapperAnimationStepAction](#) (GenericCallback< const [AnimationTextureMapper](#) & > &callback)  
*Associates an action to be performed when the animation steps.*
- void [setTextureMapperAnimationEndedAction](#) (GenericCallback< const [AnimationTextureMapper](#) & > &callback)  
*Associates an action to be performed when the animation ends.*
- virtual bool [isTextureMapperAnimationRunning](#) () const  
*Gets whether or not the animation is running.*
- virtual void [setupAnimation](#) ([AnimationParameter](#) parameter, float endValue, uint16\_t duration, uint16\_t delay, [EasingEquation](#) progressionEquation=&[EasingEquations::linearEaseNone](#))  
*Sets up the animation for a specific parameter (angle/scale) for the next animation.*
- virtual void [startAnimation](#) ()  
*Starts the animation.*
- virtual void [cancelAnimationTextureMapperAnimation](#) ()  
*Cancel move animation.*
- virtual uint16\_t [getAnimationStep](#) ()  
*Gets the current animation step.*

## Static Public Attributes

- static const int [NUMBER\\_OF\\_ANIMATION\\_PARAMETERS](#) = SCALE + 1  
*Number of animation parameters.*

## Protected Member Functions

- virtual void [handleTickEvent](#) ()  
*The tick handler that handles the actual animation steps.*

## Protected Attributes

- [AnimationSetting](#) animations [[NUMBER\\_OF\\_ANIMATION\\_PARAMETERS](#)]  
*Descriptions of the animation of specific animation parameters.*
- [GenericCallback](#)< const [AnimationTextureMapper](#) & > \* [textureMapperAnimationStepCallback](#)  
*Animation has performed a step Callback.*
- [GenericCallback](#)< const [AnimationTextureMapper](#) & > \* [textureMapperAnimationEndedCallback](#)  
*Animation ended Callback.*
- uint16\_t [animationCounter](#)  
*Counter that is equal to the current step in the animation.*
- bool [animationRunning](#)  
*Boolean that is true if the animation is running.*

## Additional Inherited Members

### 7.18.1 Detailed Description

A texture mapper with animation capabilities. Note that the angles of the [TextureMapper](#) is moved to the  $[0; 2\text{PI}]$  range at the beginning at the animation. The end angles should be relative to this and are limited to values in the range  $[-32.7; 32.7]$ .

See also

[TextureMapper](#)

### 7.18.2 Member Enumeration Documentation

#### 7.18.2.1 AnimationParameter

enum [AnimationParameter](#)

Enumerator

X_ROTATION	Rotation around the X axis.
Y_ROTATION	Rotation around the Y axis.
Z_ROTATION	Rotation around the Z axis.
SCALE	Scaling of the image.

### 7.18.3 Constructor & Destructor Documentation

#### 7.18.3.1 AnimationTextureMapper()

[AnimationTextureMapper](#) ( )

Default constructor.

#### 7.18.3.2 ~AnimationTextureMapper()

[~AnimationTextureMapper](#) ( ) [virtual]

Destructor. Destroys the [AnimationTextureMapper](#).

### 7.18.4 Member Function Documentation

#### 7.18.4.1 cancelAnimationTextureMapperAnimation()

void [cancelAnimationTextureMapperAnimation](#) ( ) [virtual]

Cancel move animation.



#### 7.18.4.2 getAnimationStep()

```
uint16_t getAnimationStep ( ) [virtual]
```

##### Returns

The current animation step.

#### 7.18.4.3 handleTickEvent()

```
void handleTickEvent ( ) [protected], [virtual]
```

The tick handler that handles the actual animation steps.

Reimplemented from [Drawable](#).

#### 7.18.4.4 isTextureMapperAnimationRunning()

```
bool isTextureMapperAnimationRunning ( ) const [virtual]
```

Gets whether or not the animation is running.

##### Returns

true if the animation is running.

#### 7.18.4.5 setTextureMapperAnimationEndedAction()

```
void setTextureMapperAnimationEndedAction (
    GenericCallback< const AnimationTextureMapper & > & callback )
```

Associates an action to be performed when the animation ends.

##### Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">AnimationTextureMapper</a> .
-----------------	---------------------------------------------------------------------------------------------------------------------

##### See also

[GenericCallback](#)

#### 7.18.4.6 setTextureMapperAnimationStepAction()

```
void setTextureMapperAnimationStepAction (
    GenericCallback< const AnimationTextureMapper & > & callback )
```

Associates an action to be performed when the animation steps. Will not be called during delay period.

## Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">AnimationTextureMapper</a> .
-----------------	---------------------------------------------------------------------------------------------------------------------

## See also

[GenericCallback](#)

7.18.4.7 `setupAnimation()`

```
void setupAnimation (
    AnimationParameter parameter,
    float endValue,
    uint16_t duration,
    uint16_t delay,
    EasingEquation progressionEquation = &EasingEquations::linearEaseNone ) [virtual]
```

Sets up the animation for a specific parameter (angle/scale) for the next animation. The specific parameter is chosen using the AnimationType enum. AnimationTypes that are not setup using this method will keep their value during the animation.

## Parameters

<i>parameter</i>	The parameter which animation details are being specified.
<i>endValue</i>	The end value for the parameter.
<i>duration</i>	The duration for the animation of this parameter. Specified in ticks.
<i>delay</i>	The delay for the animation of this parameter. Specified in ticks.
<i>progressionEquation</i>	the progression equation for the animation of this parameter.

7.18.4.8 `startAnimation()`

```
void startAnimation ( ) [virtual]
```

Starts the animation from the current position to the specified end angles/scale. The progression of the angles/scale during the animation is described by the supplied [EasingEquations](#).

## 7.19 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.*
- uint16\_t `getNumberOfRegisteredTimerWidgets` () const  
*gets the number of timer widgets that has been registered*
- uint16\_t `getTimerWidgetCountForDrawable` (`Drawable` \*w) const  
*Gets the number of timer events registered to a widget.*

### Static Public Member Functions

- static `Application` \* `getInstance` ()  
*Gets the single instance application.*

### Static Public Attributes

- static const uint8\_t `MAX_TIMER_WIDGETS` = 32  
*Maximum number of widgets receiving ticks.*
- static const uint16\_t `TICK_INTERVAL_MS` = 10  
*Deprecated, do not use this constant. Tick interval depends on VSYNC of your target platform.*

### Protected Member Functions

- void `invalidateArea` (`Rect` area)  
*Invalidates this area.*
- `Application` ()  
*Proected 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](#)

### 7.19.1 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](#)

### 7.19.2 Constructor & Destructor Documentation

#### 7.19.2.1 Application()

```
Application ( ) [protected]
```

Protected constructor.

### 7.19.3 Member Function Documentation

## 7.19.3.1 appSwitchScreen()

```
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

<i>screenId</i>	An id that maps to the desired screen.
-----------------	----------------------------------------

## 7.19.3.2 cacheDrawOperations()

```
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

<i>enableCache</i>	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.
--------------------	---------------------------------------------------------------------------------------------------------------------------------------------------

## 7.19.3.3 clearAllTimerWidgets()

```
void clearAllTimerWidgets ( )
```

Clears all currently registered timer widgets.

## 7.19.3.4 draw() [1/2]

```
void draw ( ) [virtual]
```

Initiate a draw operation of the entire screen. Standard implementation is to delegate draw request to the current [Screen](#).

## 7.19.3.5 draw() [2/2]

```
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	<i>rect</i>	The area to draw.
----	-------------	-------------------

**7.19.3.6 getCurrentScreen()**

```
Screen * getCurrentScreen ( ) [inline]
```

Gets the current screen.

**Returns**

The current screen.

**7.19.3.7 getInstance()**

```
static Application * getInstance ( ) [static]
```

Gets the single instance application.

**Returns**

The instance of this application.

**7.19.3.8 getNumberOfRegisteredTimerWidgets()**

```
uint16_t getNumberOfRegisteredTimerWidgets ( ) const
```

gets the number of timer widgets that has been registered.

**Returns**

The size of timerWidgets.

**7.19.3.9 getTimerWidgetCountForDrawable()**

```
uint16_t getTimerWidgetCountForDrawable (
    Drawable * w ) const
```

Gets the number of timer events registered to a widget, i.e. how many times a drawable must be unregistered until it no longer receives timer ticks.

**Parameters**

in	<i>w</i>	The widget to to get count from.
----	----------	----------------------------------

**Returns**

0 if the drawable is not registered as a timer widget, otherwise returns how many times the drawable is currently registered.

**7.19.3.10 handleClickEvent()**

```
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**

<i>evt</i>	The <a href="#">ClickEvent</a> .
------------	----------------------------------

Reimplemented from [UIEventListener](#).

**7.19.3.11 handleDragEvent()**

```
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**

<i>evt</i>	The drag event, expressed in absolute coordinates.
------------	----------------------------------------------------

Reimplemented from [UIEventListener](#).

**7.19.3.12 handleGestureEvent()**

```
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**

<i>evt</i>	The gesture event.
------------	--------------------

Reimplemented from [UIEventListener](#).

#### 7.19.3.13 handleKeyEvent()

```
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

<code>c</code>	The incoming character to handle.
----------------	-----------------------------------

Reimplemented from [UIEventListener](#).

#### 7.19.3.14 handlePendingScreenTransition()

```
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](#).

#### 7.19.3.15 handleTickEvent()

```
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](#).

#### 7.19.3.16 invalidateArea()

```
void invalidateArea (
    Rect area ) [protected]
```

Invalidates this area.

##### Parameters

<code>area</code>	The area to invalidate.
-------------------	-------------------------

#### 7.19.3.17 registerTimerWidget()

```
void registerTimerWidget (
    Drawable * w )
```

Adds a widget to the list of widgets receiving ticks every frame (typically 16.67ms)



**Note**

The framework keeps track of the number of times a specific widget is registered.

**Parameters**

in	<i>w</i>	The widget to add.
----	----------	--------------------

**See also**

[unregisterTimerWidget](#)

**7.19.3.18 switchScreen()**

```
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	<i>newScreen</i>	A pointer to the new screen.
----	------------------	------------------------------

**7.19.3.19 unregisterTimerWidget()**

```
void unregisterTimerWidget (
    const Drawable * w )
```

Removes a widget from the list of widgets receiving ticks every frame (typically 16.67ms) 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	<i>w</i>	The widget to remove.
----	----------	-----------------------

**7.19.4 Member Data Documentation****7.19.4.1 instance**

```
Application* instance [static], [protected]
```

Pointer to the instance of the Application-derived subclass.

**Note**

Must be set by subclass constructor!

**7.19.4.2 MAX\_TIMER\_WIDGETS**

```
const uint8_t MAX_TIMER_WIDGETS = 32 [static]
```

**Remarks**

Memory impact:  $x * (\text{sizeof}(\text{Drawable}) + 1)$

**7.20 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, GRAY2, GRAY4 }  
*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 [cachelsCached](#) ([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.*

#### 7.20.1 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.

#### 7.20.2 Member Enumeration Documentation

### 7.20.2.1 BitmapFormat

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
GRAY2	2-bit grayscale
GRAY4	4-bit grayscale

## 7.20.3 Constructor & Destructor Documentation

### 7.20.3.1 Bitmap()

```
Bitmap (
    const BitmapId id = BITMAP\_INVALID ) [inline], [explicit]
```

Creates and binds a [Bitmap](#) instance to the corresponding entry in the [BitmapData](#) array.

#### Parameters

<i>id</i>	The unique bitmap identifier.
-----------	-------------------------------

## 7.20.4 Member Function Documentation

### 7.20.4.1 cache()

```
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

<i>id</i>	The id of the bitmap to cache.
-----------	--------------------------------

**Returns**

true if caching went well, false otherwise.

**See also**

[registerBitmapDatabase](#)

**7.20.4.2 cacheAll()**

```
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 were cached.

**See also**

[cache](#).

**7.20.4.3 cacheGetAddress()**

```
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**

<i>id</i>	The id of the bitmap in cache.
-----------	--------------------------------

**Returns**

Address if bitmap was found, zero otherwise.

**7.20.4.4 cacheIsCached()**

```
static bool cacheIsCached (
    BitmapId id ) [static]
```

Check if the [Bitmap](#) is cached.

**Parameters**

<i>id</i>	The id of the bitmap.
-----------	-----------------------

**Returns**

true if bitmap is cached.

**7.20.4.5 cacheRemoveBitmap()**

```
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**

<i>id</i>	The id of the bitmap to cache.
-----------	--------------------------------

**Returns**

true if bitmap was found and removed, false otherwise.

**See also**

[registerBitmapDatabase](#)

**7.20.4.6 cacheReplaceBitmap()**

```
static bool cacheReplaceBitmap (  
    BitmapId out,  
    BitmapId in ) [static]
```

Replace a bitmap in RAM with another [Bitmap](#). The Bitmaps must have same size.

**Parameters**

<i>out</i>	The id of the bitmap to remove from the cache.
<i>in</i>	The id of the bitmap to cache.

**Returns**

true if the replacement went well, false otherwise.

**7.20.4.7 clearCache()**

```
static void clearCache ( ) [static]
```

Clears the cached bitmaps from RAM.

**7.20.4.8 dynamicBitmapCreate()**

```
static BitmapId dynamicBitmapCreate (
    const uint16_t width,
    const uint16_t height,
    BitmapFormat format ) [static]
```

Create a dynamic bitmap.

**Parameters**

<i>width</i>	Width of the bitmap.
<i>height</i>	Height of the bitmap.
<i>format</i>	Bitmap format of the bitmap.

**Returns**

BitmapId of the new bitmap or BITMAP\_INVALID if memory full.

**Note**

Creation of a new dynamic bitmap may cause existing dynamic bitmaps to be moved in memory. Do not rely on bitmap memory addresses of dynamic bitmaps obtained from [dynamicBitmapGetAddress\(\)](#) to be valid across calls to [dynamicBitmapCreate\(\)](#).

**See also**

[dynamicBitmapAddress](#)

**7.20.4.9 dynamicBitmapDelete()**

```
static bool dynamicBitmapDelete (
    BitmapId id ) [static]
```

Delete a dynamic bitmap.

**Parameters**

<i>id</i>	The BitmapId of the dynamic bitmap.
-----------	-------------------------------------

**Returns**

true if it succeeds, false if it fails.

**7.20.4.10 dynamicBitmapGetAddress()**

```
static uint8_t * dynamicBitmapGetAddress (
    BitmapId id ) [static]
```

Get the address of the dynamic bitmap data. It is important that the address of a dynamic bitmap is not stored elsewhere as a dynamic bitmap may be moved in memory when other bitmaps are added and removed. Only store the BitmapId and ask for the address of the bitmap data when needed.

**Parameters**

<i>id</i>	The BitmapId of the dynamic bitmap.
-----------	-------------------------------------

**Returns**

null if it fails, else an uint8\_t\*.

**Note**

The address of a dynamic bitmap may change when other dynamic bitmaps are added and removed. Never store the address of dynamic images, only store the BitmapId as that will not change.

**7.20.4.11 dynamicBitmapSetSolidRect()**

```
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**

<i>id</i>	The identifier.
<i>solidRect</i>	The solid rectangle.

**Returns**

true if it succeeds, false if it fails.

**7.20.4.12 getAlphaData()**

```
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.

**7.20.4.13 getData()**

```
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.

**7.20.4.14 getFormat()**

```
BitmapFormat getFormat ( ) const
```

Gets the format of how the bitmap is stored.

**Returns**

The format of how the bitmap data is stored.

**7.20.4.15 getHeight()**

```
uint16_t getHeight ( ) const
```

Gets the height of the [Bitmap](#) in pixels.

**Returns**

The bitmap height in pixels.

**7.20.4.16 getId()**

```
BitmapId getId ( ) const [inline]
```

Gets the id of this [Bitmap](#).

**Returns**

The id of this [Bitmap](#).

#### 7.20.4.17 getRect()

```
Rect getRect ( ) const [inline]
```

Gets the rectangle describing the dimensions of the [Bitmap](#).

##### Returns

a [Rect](#) describing the dimensions of this bitmap.

#### 7.20.4.18 getSolidRect()

```
Rect getSolidRect ( ) const
```

Gets the largest solid, i.e. not transparent, rectangle in the bitmap.

##### Returns

The maximum solid rectangle of the bitmap.

#### 7.20.4.19 getWidth()

```
uint16_t getWidth ( ) const
```

Gets the width of the [Bitmap](#) in pixels.

##### Returns

The bitmap width in pixels.

#### 7.20.4.20 hasTransparentPixels()

```
bool hasTransparentPixels ( ) const
```

##### Returns

True if this bitmap has transparent pixels.

#### 7.20.4.21 isAlphaPerPixel()

```
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)

## 7.20.4.22 operator!=(())

```
bool operator!= (
    const Bitmap & other ) const [inline]
```

Inequality operator.

## Parameters

<i>other</i>	The bitmap to compare with.
--------------	-----------------------------

## Returns

True if this bitmap has a different id than the other bitmap.

## 7.20.4.23 operator==(())

```
bool operator== (
    const Bitmap & other ) const [inline]
```

Equality operator.

## Parameters

<i>other</i>	The bitmap to compare with.
--------------	-----------------------------

## Returns

True if this bitmap has the same id as the other bitmap.

## 7.20.4.24 registerBitmapDatabase()

```
static void registerBitmapDatabase (
    const BitmapData * data,
    const uint16_t n,
    uint16_t * cache = 0,
    uint32_t csize = 0,
    uint32_t numberOfDynamicBitmaps = 0 ) [static]
```

Registers an array of bitmaps. All [Bitmap](#) instances are bound to this database.

## Parameters

	<i>data</i>	A reference to the <a href="#">BitmapData</a> storage array.
	<i>n</i>	The number of bitmaps in the array.
in, out	<i>cache</i>	(Optional) Pointer to memory region in which bitmap data can be cached.
	<i>csize</i>	Size of cache memory region in bytes (0 if unused)
	<i>numberOfDynamicBitmaps</i>	Number of dynamic bitmaps to be allowed in the cache.

#### 7.20.4.25 setCache()

```
static void setCache (
    uint16_t * cachep,
    uint32_t csize,
    uint32_t numberOfDynamicBitmaps = 0 ) [static]
```

Register a memory region in which bitmap data can be cached.

##### Parameters

in, out	<i>cachep</i>	Pointer to memory region in which bitmap data can be cached.
	<i>csize</i>	Size of cache memory region in bytes.
	<i>numberOfDynamicBitmaps</i>	Number of dynamic bitmaps to be allowed in the cache.

## 7.21 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.*

#### 7.21.1 Detailed Description

Data of a bitmap.

## 7.22 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.*

### 7.22.1 Detailed Description

`BlitOp` instances carry the required information for performing operations on the `LCD` (frame buffer) using DMA.

### 7.22.2 Member Data Documentation

#### 7.22.2.1 operation

`uint32_t operation`

See also

[BlitOperations](#)

## 7.23 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 ~Box ()`  
*Destructor.*

- 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

### 7.23.1 Detailed Description

Simple widget capable of showing a rectangle of a specific color and an optional alpha.

See also

[Widget](#)

### 7.23.2 Constructor & Destructor Documentation

#### 7.23.2.1 [Box\(\)](#) [1/2]

```
Box ( ) [inline]
```

Constructs a new [Box](#) with a default alpha value of 255 (solid)

#### 7.23.2.2 [Box\(\)](#) [2/2]

```
Box (
    uint16_t width,
    uint16_t height,
    colortype color,
    uint8_t alpha = 255 ) [inline]
```

Construct a [Box](#).

## Parameters

<i>width</i>	The width of the box.
<i>height</i>	The height of the box.
<i>color</i>	The color of the box.
<i>alpha</i>	The alpha of the box. Default is 255 (solid).

## 7.23.2.3 ~Box()

```
~Box ( ) [inline], [virtual]
```

Destructor.

## 7.23.3 Member Function Documentation

## 7.23.3.1 draw()

```
void draw (
    const Rect & area ) const [virtual]
```

Draws the [Box](#).

## Parameters

<i>area</i>	The rectangle to draw, with coordinates relative to the containing layer.
-------------	---------------------------------------------------------------------------

Implements [Drawable](#).

## 7.23.3.2 forceReportAsSolid()

```
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

<i>solid</i>	true if this <a href="#">Box</a> should report as solid, even when not.
--------------	-------------------------------------------------------------------------

#### 7.23.3.3 `getAlpha()`

```
uint8_t getAlpha ( ) const [inline]
```

##### Returns

Gets the current alpha value of the [Box](#).

#### 7.23.3.4 `getColor()`

```
colortype getColor ( ) const [inline]
```

Gets the current color of the [Box](#).

##### Returns

The current color.

#### 7.23.3.5 `getSolidRect()`

```
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](#).

#### 7.23.3.6 `getType()`

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

##### Returns

TYPE\_BOX.

Reimplemented from [Widget](#).

#### 7.23.3.7 `setAlpha()`

```
void setAlpha (
    uint8_t alpha ) [inline]
```

Sets the alpha value for this [Box](#).



## Parameters

<i>alpha</i>	The alpha value. 255 = completely solid.
--------------	------------------------------------------

## 7.23.3.8 setColor()

```
void setColor (
    colortype color ) [inline]
```

Sets the color of the rectangle.

## Parameters

<i>color</i>	The color of the box.
--------------	-----------------------

## 7.23.4 Member Data Documentation

## 7.23.4.1 reportAsSolid

```
bool reportAsSolid [protected]
```

## See also

[forceReportAsSolid](#).

## 7.24 BoxProgress Class Reference

A box progress.

```
#include <touchgfx/containers/progress_indicators/BoxProgress.hpp>
```

## Public Member Functions

- [BoxProgress](#) ()  
*Default constructor.*
- virtual [~BoxProgress](#) ()  
*Destructor.*
- virtual void [setProgressIndicatorPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets the position and dimension of the box progress indicator.*
- virtual void [setColor](#) (colortype color)  
*Sets the color.*
- virtual colortype [getColor](#) () const  
*Gets the color.*
- virtual void [setAlpha](#) (uint8\_t alpha)  
*Sets the alpha.*
- virtual uint8\_t [getAlpha](#) () const  
*Gets the alpha.*

- virtual void [setValue](#) (int value)  
*Sets a value.*

## Protected Attributes

- [Box](#) box  
*The box.*

## Additional Inherited Members

### 7.24.1 Detailed Description

A [Box](#) progress which shows the current progress using a simple [Box](#). It is possible to set the color and the alpha of the box. It is also possible to control in what direction the box will progress (up, down, to the left or to the right).

See also

[Box](#)

### 7.24.2 Constructor & Destructor Documentation

#### 7.24.2.1 [BoxProgress](#)()

```
BoxProgress ( )
```

Default constructor.

#### 7.24.2.2 [~BoxProgress](#)()

```
~BoxProgress ( ) [virtual]
```

Destructor.

### 7.24.3 Member Function Documentation

#### 7.24.3.1 [getAlpha](#)()

```
uint8_t getAlpha ( ) const [virtual]
```

Gets the alpha of the [Box](#).

Returns

The alpha.

See also

[Box](#)

#### 7.24.3.2 getColor()

```
colortype getColor ( ) const [virtual]
```

Gets the color of the [Box](#).

##### Returns

The color.

##### See also

[Box](#)

#### 7.24.3.3 setAlpha()

```
void setAlpha (
    uint8_t alpha ) [virtual]
```

Sets the alpha of the [Box](#).

##### Parameters

<i>alpha</i>	The alpha.
--------------	------------

##### See also

[Box](#)

#### 7.24.3.4 setColor()

```
void setColor (
    colortype color ) [virtual]
```

Sets the color of the [Box](#).

##### Parameters

<i>color</i>	The color.
--------------	------------

##### See also

[Box](#)

#### 7.24.3.5 setProgressIndicatorPosition()

```
void setProgressIndicatorPosition (
    int16_t x,
    int16_t y,
```

```

    int16_t width,
    int16_t height ) [virtual]

```

Sets the position and dimension of the box progress indicator relative to the background image.

#### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>width</i>	The width of the box progress indicator.
<i>height</i>	The height of the box progress indicator.

Reimplemented from [AbstractProgressIndicator](#).

#### 7.24.3.6 setValue()

```

virtual void setValue (
    int value ) [virtual]

```

Sets the current value in the range (min..max) set by [setRange\(\)](#). Values lower than min are mapped to min, values higher than max are mapped to max.

#### Parameters

<i>value</i>	The value.
--------------	------------

Reimplemented from [AbstractProgressIndicator](#).

## 7.25 BoxWithBorder Class Reference

A box with border.

```
#include <touchgfx/containers/buttons/BoxWithBorder.hpp>
```

### Public Member Functions

- [BoxWithBorder](#) ()  
*Default constructor.*
- [BoxWithBorder](#) (uint16\_t width, uint16\_t height, [colortype color](#), [colortype borderColor](#), uint8\_t [borderSize](#), uint8\_t [alpha=255](#))  
*Constructor.*
- virtual [~BoxWithBorder](#) ()  
*Destructor.*
- virtual [Rect getSolidRect](#) () const  
*Gets solid rectangle.*
- void [setColor](#) ([colortype color](#))  
*Sets a color.*
- [colortype getColor](#) () const  
*Gets the color.*
- void [setBorderColor](#) ([colortype color](#))  
*Sets border color.*

- `colortype getBorderColor ()` const  
*Gets border color.*
- `void setBorderSize (uint8_t size)`  
*Sets border size.*
- `uint8_t getBorderSize ()` const  
*Gets border size.*
- `void setAlpha (uint8_t alpha)`  
*Sets an alpha.*
- `uint8_t getAlpha ()` const  
*Gets the alpha.*
- `virtual void draw (const Rect &area)` const  
*Draws the given area.*
- `virtual uint16_t getType ()` const  
*Gets the type.*

### Protected Attributes

- `uint8_t alpha`  
*The alpha.*
- `colortype color`  
*The color.*
- `colortype borderColor`  
*The border color.*
- `uint8_t borderSize`  
*Size of the border.*

### Additional Inherited Members

#### 7.25.1 Constructor & Destructor Documentation

##### 7.25.1.1 BoxWithBorder()

```
BoxWithBorder (
    uint16_t width,
    uint16_t height,
    colortype color,
    colortype borderColor,
    uint8_t borderSize,
    uint8_t alpha = 255 ) [inline]
```

#### Parameters

<i>width</i>	The width.
<i>height</i>	The height.
<i>color</i>	The color.
<i>borderColor</i>	The border color.
<i>borderSize</i>	Size of the border.
<i>alpha</i>	(Optional) The alpha.

## 7.25.2 Member Function Documentation

### 7.25.2.1 draw()

```
void draw (
    const Rect & area ) const [virtual]
```

#### Parameters

<i>area</i>	The area.
-------------	-----------

Implements [Drawable](#).

### 7.25.2.2 getAlpha()

```
uint8_t getAlpha ( ) const [inline]
```

#### Returns

The alpha.

### 7.25.2.3 getBorderColor()

```
colortype getBorderColor ( ) const [inline]
```

#### Returns

The border color.

### 7.25.2.4 getBorderSize()

```
uint8_t getBorderSize ( ) const [inline]
```

#### Returns

The border size.

### 7.25.2.5 getColor()

```
colortype getColor ( ) const [inline]
```

#### Returns

The color.

#### 7.25.2.6 getSolidRect()

```
Rect getSolidRect ( ) const [virtual]
```

##### Returns

The solid rectangle.

Implements [Drawable](#).

#### 7.25.2.7 getType()

```
uint16_t getType ( ) const [inline], [virtual]
```

##### Returns

The type.

Reimplemented from [Widget](#).

#### 7.25.2.8 setAlpha()

```
void setAlpha (
    uint8_t alpha ) [inline]
```

##### Parameters

<i>alpha</i>	The alpha.
--------------	------------

#### 7.25.2.9 setBorderColor()

```
void setBorderColor (
    colortype color ) [inline]
```

##### Parameters

<i>color</i>	The color.
--------------	------------

#### 7.25.2.10 setBorderSize()

```
void setBorderSize (
    uint8_t size ) [inline]
```

##### Parameters

<i>size</i>	The size.
-------------	-----------

### 7.25.2.11 setColor()

```
void setColor (
    colortype color ) [inline]
```

#### Parameters

<i>color</i>	The color.
--------------	------------

## 7.26 BoxWithBorderButtonStyle< T > Class Template Reference

A box with border button style.

```
#include <touchgfx/containers/buttons/BoxWithBorderButtonStyle.hpp>
```

### Public Member Functions

- [BoxWithBorderButtonStyle \(\)](#)  
*Default constructor.*
- virtual [~BoxWithBorderButtonStyle \(\)](#)  
*Destructor.*
- void [setBoxWithBorderPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets the size and position of this [BoxWithBorderButtonStyle](#).*
- void [setBoxWithBorderWidth](#) (int16\_t width)  
*Sets a width.*
- void [setBoxWithBorderHeight](#) (int16\_t height)  
*Sets a height.*
- void [setBoxWithBorderColors](#) (const colortype colorReleased, const colortype colorPressed, const colortype borderColorReleased, const colortype borderColorPressed)  
*Sets the colors.*
- void [setBorderSize](#) (uint8\_t size)  
*Sets border size.*

### Protected Member Functions

- virtual void [handlePressedUpdated](#) ()  
*Handles the pressed updated.*
- virtual void [handleAlphaUpdated](#) ()  
*Handles the alpha updated.*

### Protected Attributes

- [BoxWithBorder](#) [borderBox](#)  
*The border box.*
- colortype [up](#)  
*The up.*
- colortype [down](#)



*The down.*

- [colortype borderUp](#)

*The border up.*

- [colortype borderDown](#)

*The border down.*

### 7.26.1 Detailed Description

```
template<class T>
```

```
class touchgfx::BoxWithBorderStyle< T >
```

A box with border button style. This class is supposed to be used with one of the ButtonTrigger classes to create a functional button. This class will show a box with a border in different colors depending on the state of the button (pressed or released).

An image button style. This class is supposed to be used with one of the ButtonTrigger classes to create a functional button. This class will show one of two images depending on the state of the button (pressed or released).

#### Template Parameters

<i>T</i>	Generic type parameter. Typically a <a href="#">AbstractButtonContainer</a> subclass.
----------	---------------------------------------------------------------------------------------

#### See also

[AbstractButtonContainer](#), [BoxWithBorder](#)

### 7.26.2 Member Function Documentation

#### 7.26.2.1 setBorderSize()

```
void setBorderSize (
    uint8_t size ) [inline]
```

#### Parameters

<i>size</i>	The size.
-------------	-----------

#### 7.26.2.2 setBoxWithBorderColors()

```
void setBoxWithBorderColors (
    const colortype colorReleased,
    const colortype colorPressed,
    const colortype borderColorReleased,
    const colortype borderColorPressed ) [inline]
```

#### Parameters

<i>colorReleased</i>	The color released.
<i>colorPressed</i>	The color pressed.

## Parameters

<i>borderColorReleased</i>	The border color released.
<i>borderColorPressed</i>	The border color pressed.

**7.26.2.3 setBoxWithBorderHeight()**

```
void setBoxWithBorderHeight (
    int16_t height ) [inline]
```

## Parameters

<i>height</i>	The height.
---------------	-------------

**7.26.2.4 setBoxWithBorderPosition()**

```
void setBoxWithBorderPosition (
    int16_t x,
    int16_t y,
    int16_t width,
    int16_t height ) [inline]
```

Sets the size and position of this [BoxWithBorderButtonStyle](#), relative to its parent.

## Note

Changing this does not automatically yield a redraw.

## Parameters

<i>x</i>	The x coordinate of this <a href="#">BoxWithBorderButtonStyle</a> .
<i>y</i>	The y coordinate of this <a href="#">BoxWithBorderButtonStyle</a> .
<i>width</i>	The width of this <a href="#">BoxWithBorderButtonStyle</a> .
<i>height</i>	The height of this <a href="#">BoxWithBorderButtonStyle</a> .

**7.26.2.5 setBoxWithBorderWidth()**

```
void setBoxWithBorderWidth (
    int16_t width ) [inline]
```

## Parameters

<i>width</i>	The width.
--------------	------------

## 7.27 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

#### 7.27.1 Detailed Description

A button consists of two images, one for its normal state and one when it is pressed down.

See also

[AbstractButton](#)

#### 7.27.2 Constructor & Destructor Documentation

### 7.27.2.1 Button()

```
Button ( ) [inline]
```

Default constructor.

### 7.27.2.2 ~Button()

```
~Button ( ) [inline], [virtual]
```

Destructor.

## 7.27.3 Member Function Documentation

### 7.27.3.1 draw()

```
void draw (
    const Rect & invalidatedArea ) const [virtual]
```

#### Parameters

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

#### See also

[Drawable::draw\(\)](#)

Implements [Drawable](#).

Reimplemented in [ButtonWithLabel](#), and [ButtonWithIcon](#).

### 7.27.3.2 getAlpha()

```
uint8_t getAlpha ( ) const [inline]
```

Gets the current alpha value.

#### Returns

The current alpha value.

### 7.27.3.3 getCurrentlyDisplayedBitmap()

```
Bitmap getCurrentlyDisplayedBitmap ( ) const [inline]
```

Function to obtain the currently displayed bitmap, which depends on the button's pressed state.

#### Returns

The bitmap currently displayed.

7.27.3.4 `getSolidRect()`

```
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](#).

7.27.3.5 `getType()`

```
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](#).

7.27.3.6 `setAlpha()`

```
void setAlpha (
    uint8_t alpha ) [inline]
```

Sets the alpha value for the image.

**Parameters**

<i>alpha</i>	The alpha value. 255 = completely solid.
--------------	------------------------------------------

7.27.3.7 `setBitmaps()`

```
void setBitmaps (
    const Bitmap & bmpReleased,
    const Bitmap & bmpPressed ) [virtual]
```

Sets the bitmaps used by this button.

**Parameters**

<i>bmpReleased</i>	<a href="#">Bitmap</a> to use when button is released.
<i>bmpPressed</i>	<a href="#">Bitmap</a> to use when button is pressed.

Reimplemented in [ToggleButton](#).

## 7.28 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.*

### 7.28.1 Detailed Description

Interface for sampling external key events.

### 7.28.2 Constructor & Destructor Documentation

#### 7.28.2.1 ~ButtonController()

```
~ButtonController ( ) [inline], [virtual]
```

Destructor.

### 7.28.3 Member Function Documentation

#### 7.28.3.1 init()

```
void init ( ) [pure virtual]
```

Initializes button controller.

#### 7.28.3.2 reset()

```
void reset ( ) [inline], [virtual]
```

Resets button controller. Does nothing in the default implementation.

#### 7.28.3.3 sample()

```
bool sample (
    uint8_t & key ) [pure virtual]
```

Sample external key events.

## Parameters

<code>in, out</code>	<code>key</code>	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.

## 7.29 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.*

#### 7.29.1 Member Function Documentation

##### 7.29.1.1 `init()`

```
static void init ( ) [static]
```

Perform configuration of IO pins.

##### 7.29.1.2 `sample()`

```
static unsigned int sample ( ) [static]
```

Sample button states.

## Returns

the sampled state of the buttons.

## 7.30 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

#### 7.30.1 Detailed Description

A [Button](#) specialization that also displays an icon on top of the button bitmap.

See also

[Button](#)

#### 7.30.2 Member Function Documentation

##### 7.30.2.1 draw()

```
virtual void draw (
    const Rect & invalidatedArea ) const [virtual]
```

##### Parameters

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------



See also

[Drawable::draw\(\)](#)

Reimplemented from [Button](#).

#### 7.30.2.2 `getCurrentlyDisplayedIcon()`

```
Bitmap getCurrentlyDisplayedIcon ( ) const [inline]
```

Function to obtain the currently displayed icon, which depends on the button's pressed state.

##### Returns

The icon currently displayed.

#### 7.30.2.3 `getIconX()`

```
int16_t getIconX ( ) const [inline]
```

Gets the x coordinate of the icon bitmap.

##### Returns

The x coordinate of the icon bitmap.

#### 7.30.2.4 `getIconY()`

```
int16_t getIconY ( ) const [inline]
```

Gets the y coordinate of the icon bitmap.

##### Returns

The y coordinate of the icon bitmap.

#### 7.30.2.5 `getType()`

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

##### Returns

TYPE\_BUTTONWITHICON.

Reimplemented from [Button](#).

### 7.30.2.6 setBitmaps()

```
void setBitmaps (
    const Bitmap & newBackgroundReleased,
    const Bitmap & newBackgroundPressed,
    const Bitmap & newIconReleased,
    const Bitmap & newIconPressed ) [virtual]
```

Sets the bitmaps used by this button.

#### Parameters

<i>newBackgroundReleased</i>	<a href="#">Bitmap</a> to use when button is released.
<i>newBackgroundPressed</i>	<a href="#">Bitmap</a> to use when button is pressed.
<i>newIconReleased</i>	The bitmap for the icon in the released/unpressed button state.
<i>newIconPressed</i>	The bitmap for the icon in the pressed button state.

### 7.30.2.7 setIconX()

```
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

<i>x</i>	The new x value, relative to the background bitmap. A negative value is allowed.
----------	----------------------------------------------------------------------------------

### 7.30.2.8 setIconXY()

```
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

<i>x</i>	The new x value, relative to the background bitmap. A negative value is allowed.
<i>y</i>	The new y value, relative to the background bitmap. A negative value is allowed.

## 7.30.2.9 setIconY()

```
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.
---	----------------------------------------------------------------------------------

## 7.31 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

### 7.31.1 Detailed Description

A [Button](#) specialization that also displays a text on top of the button bitmap.

See also

[Button](#)

### 7.31.2 Constructor & Destructor Documentation

#### 7.31.2.1 ButtonWithLabel()

```
ButtonWithLabel ( )
```

Default constructor.

### 7.31.3 Member Function Documentation

#### 7.31.3.1 draw()

```
virtual void draw (
    const Rect & invalidatedArea ) const [virtual]
```

#### Parameters

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

See also

[Drawable::draw\(\)](#)

Reimplemented from [Button](#).

#### 7.31.3.2 getLabelRotation()

```
TextRotation getLabelRotation ( ) [inline]
```

Gets the current rotation of the text on the label.

##### Returns

The current rotation of the text.

#### 7.31.3.3 getLabelText()

```
TypedText getLabelText ( ) const [inline]
```

Gets the text used for the label.

##### Returns

The text used for the label.

#### 7.31.3.4 getSolidRect()

```
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](#).

#### 7.31.3.5 getType()

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

##### Returns

TYPE\_BUTTONWITHLABEL.

Reimplemented from [Button](#).

#### 7.31.3.6 setLabelColor()

```
void setLabelColor (
    color_type col,
    bool performInvalidate = false ) [inline]
```

Sets label color.

## Parameters

<i>col</i>	The color with which the text label should be drawn.
<i>performInvalidate</i>	Optional parameter. If true, performs an instant invalidation of the button.

**7.31.3.7 setLabelColorPressed()**

```
void setLabelColorPressed (
    color\_t col,
    bool performInvalidate = false ) [inline]
```

Sets label color when the button is pressed.

## Parameters

<i>col</i>	The color with which the text label should be drawn when the button is pressed.
<i>performInvalidate</i>	Optional parameter. If true, performs an instant invalidation of the button.

**7.31.3.8 setLabelRotation()**

```
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

<i>rotation</i>	the rotation of the text. Default is TEXT_ROTATE_0.
-----------------	-----------------------------------------------------

**7.31.3.9 setLabelText()**

```
void setLabelText (
    TypedText t ) [inline]
```

Sets the text to display on the button. [Texts](#) with wildcards are not supported.

## Parameters

<i>t</i>	The text to display.
----------	----------------------

**7.31.3.10 updateTextPosition()**

```
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.

## 7.32 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.*

### 7.32.1 Detailed Description

Cache bookkeeping.

## 7.33 Callback< dest\_type, T1, T2, T3 > 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\_3)(T1, T2, T3))  
*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, T3 t3)  
*Calls the member function.*
- virtual bool [isValid](#) () const  
*Function to check whether the [Callback](#) has been initialized with values.*

### 7.33.1 Detailed Description

```
template<class dest_type, typename T1 = void, typename T2 = void, typename T3 = void>
struct touchgfx::Callback< dest_type, T1, T2, T3 >
```

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 four versions, for supporting member functions with 0, 1, 2 or 3 arguments. The compiler will infer which type to use automatically.

**Note**

The member function to call must return void. The function can have zero, one, two or three arguments of any type.

**Template Parameters**

<i>dest_type</i>	The type of the class in which the member function resides.
<i>T1</i>	The type of the first argument in the member function, or void if none.
<i>T2</i>	The type of the second argument in the member function, or void if none.
<i>T3</i>	The type of the third argument in the member function, or void if none.

**7.33.2 Constructor & Destructor Documentation****7.33.2.1 Callback()** [1/2]

```
Callback ( ) [inline]
```

Initializes an empty callback.

**7.33.2.2 Callback()** [2/2]

```
Callback (
    dest_type * pobject,
    void(dest_type::*)(T1, T2, T3) pmemfun_3 ) [inline]
```

Initializes a [Callback](#) with an object and a pointer to the member function in that object to call.

**Parameters**

in	<i>pobject</i>	Pointer to the object on which the function should be called.
in	<i>pmemfun_3</i>	Address of member function. This is the version where function takes three arguments.

**7.33.3 Member Function Documentation****7.33.3.1 execute()**

```
void execute (
    T1 t1,
    T2 t2,
    T3 t3 ) [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**

<i>t1</i>	This value will be passed as the first argument in the function call.
-----------	-----------------------------------------------------------------------



## Parameters

<i>t2</i>	This value will be passed as the second argument in the function call.
<i>t3</i>	This value will be passed as the third argument in the function call.

Implements [GenericCallback< T1, T2, T3 >](#).

## 7.33.3.2 isValid()

```
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, T3 >](#).

## 7.34 Callback&lt; dest\_type, T1, T2, void &gt; 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.*

## 7.34.1 Detailed Description

```
template<class dest_type, typename T1, typename T2>
struct touchgfx::Callback< dest_type, T1, T2, 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 four versions, for supporting member functions with 0, 1, 2 or 3 arguments. The compiler will infer which type to use automatically.

**Note**

The member function to call must return void. The function can have zero, one, two or three arguments of any type.

**Template Parameters**

<i>dest_type</i>	The type of the class in which the member function resides.
<i>T1</i>	The type of the first argument in the member function, or void if none.
<i>T2</i>	The type of the second argument in the member function, or void if none.

**7.34.2 Constructor & Destructor Documentation****7.34.2.1 Callback()** [1/2]

```
Callback ( ) [inline]
```

Initializes an empty callback.

**7.34.2.2 Callback()** [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	<i>pobject</i>	Pointer to the object on which the function should be called.
in	<i>pmemfun_2</i>	Address of member function. This is the version where function takes two arguments.

**7.34.3 Member Function Documentation****7.34.3.1 execute()**

```
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**

<i>t1</i>	This value will be passed as the first argument in the function call.
<i>t2</i>	This value will be passed as the second argument in the function call.

## 7.34.3.2 isValid()

```
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 >](#).

## 7.35 Callback&lt; dest\_type, T1, void, void &gt; 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.*

## 7.35.1 Detailed Description

```
template<class dest_type, typename T1>
struct touchgfx::Callback< dest_type, T1, 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 four versions, for supporting member functions with 0, 1, 2 or 3 arguments. The compiler will infer which type to use automatically.

## Note

The member function to call must return void. The function can have zero, one, two or three arguments of any type.

## Template Parameters

<i>dest_type</i>	The type of the class in which the member function resides.
<i>T1</i>	The type of the first argument in the member function, or void if none.

## 7.35.2 Constructor & Destructor Documentation

### 7.35.2.1 Callback() [1/2]

```
Callback ( ) [inline]
```

Initializes an empty callback.

### 7.35.2.2 Callback() [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	<i>pobject</i>	Pointer to the object on which the function should be called.
in	<i>pmemfun_1</i>	Address of member function. This is the version where function takes one argument.

## 7.35.3 Member Function Documentation

### 7.35.3.1 execute()

```
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

<i>t1</i>	This value will be passed as the first argument in the function call.
-----------	-----------------------------------------------------------------------

#### See also

[isValid\(\)](#)

### 7.35.3.2 isValid()

```
bool isValid ( ) const [inline], [virtual]
```

Query if this object is valid.

**Returns**

true if valid, false if not.

Implements [GenericCallback< T1 >](#).

**7.36 Callback< dest\_type, void, 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.*

**7.36.1 Detailed Description**

```
template<class dest_type>
struct touchgfx::Callback< dest_type, void, 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 four versions, for supporting member functions with 0, 1, 2 or 3 arguments. The compiler will infer which type to use automatically.

**Note**

The member function to call must return void. The function can have zero, one, two or three arguments of any type.

**Template Parameters**

<code>dest_type</code>	The type of the class in which the member function resides.
------------------------	-------------------------------------------------------------

**7.36.2 Constructor & Destructor Documentation**

**7.36.2.1 Callback()** [1/2]

```
Callback ( ) [inline]
```

Initializes an empty callback.

**7.36.2.2 Callback()** [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	<i>pobject</i>	Pointer to the object on which the function should be called.
in	<i>pmemfun_0</i>	Address of member function. This is the version where function takes zero arguments.

**7.36.3 Member Function Documentation****7.36.3.1 execute()**

```
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).

**7.36.3.2 isValid()**

```
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<>](#).

**7.37 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".

## 7.38 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<typename T >  
void [moveTo](#) (T x, T y)  
*Move the current pen position.*
- template<typename T >  
void [lineTo](#) (T x, T y)  
*Draw line from current pen position.*
- bool [render](#) ()  
*Render the drawn shape.*

### 7.38.1 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.

### 7.38.2 Constructor & Destructor Documentation

#### 7.38.2.1 Canvas()

```
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

<code>_widget</code>	a pointer to the <a href="#">CanvasWidget</a> using this <a href="#">Canvas</a> . Used for getting the canvas dimensions.
<code>invalidatedArea</code>	the are which should be updated.

7.38.2.2 `~Canvas()`

`~Canvas ( )` [virtual]

Destructor. Takes care of unlocking the frame buffer.

## 7.38.3 Member Function Documentation

7.38.3.1 `lineTo()` [1/2]

```
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

<code>x</code>	The x coordinate for the pen position in Q5 format.
<code>y</code>	The y coordinate for the pen position in Q5 format.

## See also

[CWRUtil::Q5](#)  
[moveTo](#)

7.38.3.2 `lineTo()` [2/2]

```
template< typename 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

<code>T</code>	either int or float.
----------------	----------------------

## Parameters

<code>x</code>	The x coordinate for the pen position.
----------------	----------------------------------------



## Parameters

<i>y</i>	The y coordinate for the pen position.
----------	----------------------------------------

## 7.38.3.3 moveTo() [1/2]

```
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

<i>x</i>	The x coordinate for the pen position in Q5 format.
<i>y</i>	The y coordinate for the pen position in Q5 format.

## See also

[CWRUtil::Q5](#)  
[lineTo](#)

## 7.38.3.4 moveTo() [2/2]

```
template< typename 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

<i>T</i>	Either int or float.
----------	----------------------

## Parameters

<i>x</i>	The x coordinate for the pen position.
<i>y</i>	The y coordinate for the pen position.

## 7.38.3.5 render()

```
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.

## 7.39 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.*
- virtual void [setPainter](#) ([AbstractPainter](#) &painter)  
*Sets the painter for the [CanvasWidget](#).*
- virtual [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

#### 7.39.1 Detailed Description

Class for drawing complex polygons on the LCD using [CanvasWidgetRenderer](#).

See also

[Widget](#)

#### 7.39.2 Constructor & Destructor Documentation

## 7.39.2.1 CanvasWidget()

`CanvasWidget ( )`

Constructor.

## 7.39.2.2 ~CanvasWidget()

`~CanvasWidget ( ) [virtual]`

Destructor. Declared virtual for sub classing purposes.

## 7.39.3 Member Function Documentation

## 7.39.3.1 draw()

```
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

<code>invalidatedArea</code>	The invalidated area.
------------------------------	-----------------------

## See also

[drawCanvasWidget\(\)](#)

Implements [Drawable](#).

## 7.39.3.2 drawCanvasWidget()

```
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

<code>invalidatedArea</code>	The invalidated area.
------------------------------	-----------------------

## Returns

true the widget was drawn, false if not.

See also

[draw\(\)](#)

Implemented in [Circle](#), [Line](#), and [AbstractShape](#).

#### 7.39.3.3 getAlpha()

```
uint8_t getAlpha ( ) const [inline], [virtual]
```

Gets the current alpha value.

Returns

The current alpha value.

#### 7.39.3.4 getMinimalRect()

```
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](#), [Circle](#), and [AbstractShape](#).

#### 7.39.3.5 getPainter()

```
AbstractPainter & getPainter ( ) const [virtual]
```

Gets the current painter for the [CanvasWidget](#).

Returns

The painter.

See also

[AbstractPainter](#)

#### 7.39.3.6 getSolidRect()

```
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](#).

**7.39.3.7 invalidate()**

```
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](#).

**7.39.3.8 setAlpha()**

```
void setAlpha (
    uint8_t alpha ) [inline], [virtual]
```

Sets the alpha channel for the [CanvasWidget](#).

**Parameters**

<i>alpha</i>	The alpha value. 255 = completely solid.
--------------	------------------------------------------

**7.39.3.9 setPainter()**

```
void setPainter (
    AbstractPainter & painter ) [virtual]
```

Sets the painter for the [CanvasWidget](#).

**Note**

The area containing the [CanvasWidget](#) is not invalidated.

**Parameters**

in	<i>painter</i>	The painter for the <a href="#">CanvasWidget</a> .
----	----------------	----------------------------------------------------

**See also**

[AbstractPainter](#)

## 7.40 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 [setWriteMemoryUsageReport](#) (bool writeUsageReport)  
*Memory reporting.*
- static bool [getWriteMemoryUsageReport](#) ()  
*Gets write memory usage report flag.*
- 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

#### 7.40.1 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 [CanvasWidgetRenderer](#). 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](#)

## 7.40.2 Member Function Documentation

### 7.40.2.1 getMissingBufferSize()

```
static unsigned getMissingBufferSize ( ) [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.

### 7.40.2.2 getOutlineBuffer()

```
static Cell * getOutlineBuffer ( ) [static]
```

Gets pointer to memory used for [Cell](#) objects in [Outline](#).

#### Returns

Pointer to memory used internally by [Outline](#).

### 7.40.2.3 getOutlineBufferSize()

```
static unsigned int getOutlineBufferSize ( ) [static]
```

Gets size of memory area used for [Cell](#) objects in [Outline](#).

#### Returns

Size of memory area used internally by [Outline](#).

### 7.40.2.4 getScanlineCounts()

```
static void * getScanlineCounts ( ) [static]
```

Gets pointer to memory used for counts in [Scanline](#).

#### Returns

Pointer to memory used internally by [Scanline](#).

### 7.40.2.5 getScanlineCovers()

```
static void * getScanlineCovers ( ) [static]
```

Gets pointer to memory used for covers in [Scanline](#).

**Returns**

Pointer to memory used internally by [Scanline](#).

**7.40.2.6 getScanlineStartIndices()**

```
static void * getScanlineStartIndices ( ) [static]
```

Gets pointer to memory used for indices in [Scanline](#).

**Returns**

Pointer to memory used internally by [Scanline](#).

**7.40.2.7 getScanlineWidth()**

```
static unsigned getScanlineWidth ( ) [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](#)).

**7.40.2.8 getUsedBufferSize()**

```
static unsigned getUsedBufferSize ( ) [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.

**7.40.2.9 getWriteMemoryUsageReport()**

```
static bool getWriteMemoryUsageReport ( ) [static]
```

Gets write memory usage report flag.

**Returns**

true if it CWR writes memory reports, false if not.



#### 7.40.2.10 hasBuffer()

```
static bool hasBuffer ( ) [static]
```

Query if [CanvasWidgetRenderer](#) has been initialized with a buffer.

##### Returns

True if a buffer has been setup.

#### 7.40.2.11 numCellsMissing()

```
static void numCellsMissing (
    unsigned missing ) [static]
```

Called by the destructor in [Outline](#) to help keep track of the memory requirements of CanvasWidgets.

##### Parameters

<i>missing</i>	Number of <a href="#">Cell</a> objects required, but not available, to <a href="#">Outline</a> .
----------------	--------------------------------------------------------------------------------------------------

#### 7.40.2.12 numCellsUsed()

```
static void numCellsUsed (
    unsigned used ) [static]
```

Called by the destructor in [Outline](#) to help keep track of the memory requirements of CanvasWidgets.

##### Parameters

<i>used</i>	Number of <a href="#">Cell</a> objects used from the dedicated buffer.
-------------	------------------------------------------------------------------------

#### 7.40.2.13 setScanlineWidth()

```
static bool setScanlineWidth (
    unsigned width ) [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

<i>width</i>	The width of the scanline on screen.
--------------	--------------------------------------

##### Returns

true if it succeeds, false if it fails.

#### 7.40.2.14 `setupBuffer()`

```
static void setupBuffer (
    uint8_t * buffer,
    unsigned bufsize ) [static]
```

Setup the buffers used by [CanvasWidget](#).

##### Parameters

in	<i>buffer</i>	Buffer reserved for <a href="#">CanvasWidget</a> .
	<i>bufsize</i>	The size of the buffer.

#### 7.40.2.15 `setWriteMemoryUsageReport()`

```
static void setWriteMemoryUsageReport (
    bool writeUsageReport ) [static]
```

Memory reporting can be turned on (and off) using this method. CWR will try to work with the given amount of memory passed when calling [setupBuffer\(\)](#). If the outline of the figure is too complex, this will be reported.

"CWR requires X bytes" means that X bytes is the highest number of bytes required by CWR so far, but since the size of the invalidated area and the shape of things draw can influence this, this may be reported several times with a higher and higher number. Leave your app running for a long time to find out what the memory requirements are.

"CWR requires X bytes (Y bytes missing)" means the same as the report above, but there as was not enough memory to render the entire shape. To get around this, CWR will split the shape into two separate drawings of half size. This means that less memory is required, but drawing will be (somewhat) slower. After you see this message all future draws will be split into smaller chunks, so memory requirements might not get as high. This is followed by:

"CWR will split draw into multiple draws due to limited memory." actually just means that CWR will try to work with a smaller amount of memory.

In general, if there is enough memory available to run the simulation and never see the message "CWR will split draw ...", this is preferred. The size of the buffer required will be the highest number X reported as "CWR requires X bytes". Good numbers can also be around half of X.

##### Parameters

<i>writeUsageReport</i>	true to write report.
-------------------------	-----------------------

See also

[setupBuffer](#)

## 7.41 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).*

### 7.41.1 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.

### 7.41.2 Member Function Documentation

#### 7.41.2.1 `addCover()`

```
void addCover (
    int _cover,
    int _area ) [inline]
```

Adds a cover to a [Cell](#).

#### Parameters

<code>_cover</code>	The cover to add to the <a href="#">Cell</a> .
<code>_area</code>	The area to add to the <a href="#">Cell</a> .

#### 7.41.2.2 `packedCoord()`

```
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.

**7.41.2.3 set()**

```
void set (
    int _x,
    int _y,
    int _cover,
    int _area ) [inline]
```

Sets all the [Cell](#) parameters.

**Parameters**

<code>_x</code>	The x coordinate.
<code>_y</code>	The y coordinate.
<code>_cover</code>	The cover.
<code>_area</code>	The area.

**7.41.2.4 setCoord()**

```
void setCoord (
    int _x,
    int _y ) [inline]
```

Sets the coordinate of the [Cell](#).

**Parameters**

↔ <code>_x</code>	The <a href="#">Cell</a> 's x coordinate.
↔ <code>_y</code>	The <a href="#">Cell</a> 's y coordinate.

**7.41.2.5 setCover()**

```
void setCover (
    int _cover,
    int _area ) [inline]
```

Sets the cover of area cell.

**Parameters**

<code>_cover</code>	The cover.
<code>_area</code>	The area.

## 7.42 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<typename T >  
void [setCircle](#) (const T x, const T y, const T r)  
*Sets the center and radius of the [Circle](#).*
- void [setCircle](#) (const int16\_t x, const int16\_t y, const int16\_t r)  
*Sets the center and radius of the [Circle](#).*
- template<typename T >  
void [setCenter](#) (const T x, const T y)  
*Sets the center of the [Circle](#).*
- void [setCenter](#) (const int16\_t x, const int16\_t y)  
*Sets the center of the [Circle](#).*
- template<typename T >  
void [getCenter](#) (T &x, T &y) const  
*Gets the center coordinates of the [Circle](#).*
- template<typename T >  
void [setRadius](#) (const T r)  
*Sets the radius of the [Circle](#).*
- template<typename T >  
void [getRadius](#) (T &r) const  
*Gets the radius of the [Circle](#).*
- template<typename T >  
void [setArc](#) (const T startAngle, const T endAngle)  
*Sets the start and end angles in degrees of the [Circle](#) arc.*
- void [setArc](#) (const int16\_t startAngle, const int16\_t endAngle)  
*Sets the start and end angles in degrees of the [Circle](#) arc.*
- template<typename T >  
void [getArc](#) (T &startAngle, T &endAngle) const  
*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.*
- template<typename T >  
void [getArcStart](#) (T &angle) const  
*Gets the start angle in degrees for the arc.*
- int16\_t [getArcEnd](#) () const  
*Gets the end angle in degrees for the arc.*
- template<typename T >  
void [getArcEnd](#) (T &angle) const  
*Gets the end angle in degrees for the arc.*
- template<typename T >  
void [updateArcStart](#) (const T startAngle)  
*Updates the start angle in degrees for this [Circle](#) arc.*
- template<typename T >  
void [updateArcEnd](#) (const T endAngle)

*Updates the end angle in degrees for this [Circle](#) arc.*

- `template<typename T >`  
`void setLineWidth (const T width)`  
*Sets the line width for this [Circle](#).*
- `template<typename T >`  
`void getLineWidth (T &width) const`  
*Gets line width.*
- `void setPrecision (const int precision)`  
*Sets a precision of the [Circle](#) drawing function.*
- `int getPrecision () const`  
*Gets the precision of the circle drawing function.*
- `void setCapPrecision (const int precision)`  
*Sets the precision of the ends of the [Circle](#) arc.*
- `int getCapPrecision () const`  
*Sets the 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.*
- `Rect getMinimalRect (CWRUtil::Q5 arcStart, CWRUtil::Q5 arcEnd) const`  
*Gets minimal rectangle containing a given circle arc.*

## Additional Inherited Members

### 7.42.1 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](#)

### 7.42.2 Constructor & Destructor Documentation

#### 7.42.2.1 [Circle](#)()

[Circle](#) ( )

Constructs a new [Circle](#).

### 7.42.3 Member Function Documentation

## 7.42.3.1 drawCanvasWidget()

```
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

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

## Returns

true if it succeeds, false if it fails.

Implements [CanvasWidget](#).

## 7.42.3.2 getArc()

```
template< typename T > void getArc (
    T & startAngle,
    T & endAngle ) const [inline]
```

Gets the start and end angles in degrees for the circle arc.

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

out	<i>startAngle</i>	The start.
out	<i>endAngle</i>	The end.

## See also

[setArc](#)

## 7.42.3.3 getArcEnd() [1/2]

```
template< typename T > T getArcEnd ( ) const [inline]
```

Gets the end angle in degrees for the arc.

## Template Parameters

<i>T</i>	Generic type parameter, either int (default) or float.
----------	--------------------------------------------------------

**Returns**

The finishing angle for the arc.

**See also**

[getArc](#)  
[setArc](#)

**7.42.3.4 getArcEnd()** [2/2]

```
template< typename T > void getArcEnd (
    T & angle ) const [inline]
```

Gets the end angle in degrees for the arc.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>in, out</i>	<i>angle</i>	The angle.
----------------	--------------	------------

**7.42.3.5 getArcStart()** [1/2]

```
int16_t getArcStart ( ) const [inline]
```

Gets the start angle in degrees for the arc.

**Returns**

The starting angle for the arc.

**See also**

[getArc](#)  
[setArc](#)

**7.42.3.6 getArcStart()** [2/2]

```
template< typename T > void getArcStart (
    T & angle ) const [inline]
```

Gets the start angle in degrees for the arc.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------



**Parameters**

<i>in, out</i>	<i>angle</i>	The angle.
----------------	--------------	------------

**See also**

[getArc](#)  
[setArc](#)

**7.42.3.7 getCapPrecision()**

```
int getCapPrecision ( ) const
```

Gets the precision of the ends of the [Circle](#) arc.

**Returns**

The cap precision in degrees.

**See also**

[getCapPrecision](#)

**7.42.3.8 getCenter()**

```
template< typename T > void getCenter (
    T & x,
    T & y ) const [inline]
```

Gets the center coordinates of the [Circle](#).

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>out</i>	<i>x</i>	The x coordinate of the center.
<i>out</i>	<i>y</i>	The y coordinate of the center.

**7.42.3.9 getLineWidth()**

```
template< typename T > void getLineWidth (
    T & width ) const [inline]
```

Gets line width.

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

out	<i>width</i>	The width.
-----	--------------	------------

## See also

[setLineWidth](#)7.42.3.10 `getMinimalRect()` [1/3]

```
Rect getMinimalRect ( ) const [virtual]
```

Gets minimal rectangle for the current shape of the circle.

## Returns

The minimal rectangle.

Reimplemented from [CanvasWidget](#).

7.42.3.11 `getMinimalRect()` [2/3]

```
Rect getMinimalRect (
    int16_t arcStart,
    int16_t arcEnd ) const
```

Gets minimal rectangle containing a given circle arc.

## Parameters

<i>arcStart</i>	The arc start.
<i>arcEnd</i>	The arc end.

## Returns

The minimal rectangle.

7.42.3.12 `getMinimalRect()` [3/3]

```
Rect getMinimalRect (
    CWRUtil::Q5 arcStart,
    CWRUtil::Q5 arcEnd ) const
```

Gets minimal rectangle containing a given circle arc.

## Parameters

<i>arcStart</i>	The arc start.
<i>arcEnd</i>	The arc end.

## Returns

The minimal rectangle.

7.42.3.13 `getPrecision()`

```
int getPrecision ( ) const
```

Gets the precision of the circle drawing function. The precision is the number of degrees used as step counter when drawing smaller line fragments around the circumference of the circle, the default being being 5.

## Returns

The precision.

## See also

[setPrecision](#)

7.42.3.14 `getRadius()`

```
template< typename T > void getRadius (
    T & r ) const [inline]
```

Gets the radius of the [Circle](#).

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

out	<i>r</i>	The radius.
-----	----------	-------------

7.42.3.15 `setArc()` [1/2]

```
template< typename T > void setArc (
    const T startAngle,
    const T endAngle ) [inline]
```

Sets the start and end angles in degrees of the [Circle](#) arc. 0 degrees is straight up (12 o'clock) and 90 degrees is to the left (3 o'clock). Any positive or negative degrees can be used to specify the part of the [Circle](#) to draw.

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

<i>startAngle</i>	The start degrees.
<i>endAngle</i>	The end degrees.

## Note

The area containing the [Circle](#) is not invalidated.

## See also

[getArc](#)  
[updateArcStart](#)  
[updateArcEnd](#)

7.42.3.16 `setArc()` [2/2]

```
void setArc (  
    const int16_t startAngle,  
    const int16_t endAngle ) [inline]
```

Sets the start and end angles in degrees of the [Circle](#) arc. 0 degrees is straight up (12 o'clock) and 90 degrees is to the left (3 o'clock). Any positive or negative degrees can be used to specify the part of the [Circle](#) to draw.

## Parameters

<i>startAngle</i>	The start degrees.
<i>endAngle</i>	The end degrees.

## Note

The area containing the [Circle](#) is not invalidated.

## See also

[getArc](#)  
[updateArcStart](#)  
[updateArcEnd](#)

7.42.3.17 `setCapPrecision()`

```
void setCapPrecision (  
    const 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.

## Parameters

<i>precision</i>	The new cap precision.
------------------	------------------------

## Note

The circle is not invalidated.

7.42.3.18 `setCenter()` [1/2]

```
template< typename T > void setCenter (
    const T x,
    const T y ) [inline]
```

Sets the center of the [Circle](#).

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

<i>x</i>	The x coordinate of center.
<i>y</i>	The y coordinate of center.

## Note

The area containing the [Circle](#) is not invalidated.

7.42.3.19 `setCenter()` [2/2]

```
void setCenter (
    const int16_t x,
    const int16_t y ) [inline]
```

Sets the center of the [Circle](#).

## Parameters

<i>x</i>	The x coordinate of center.
<i>y</i>	The y coordinate of center.

## Note

The area containing the [Circle](#) is not invalidated.

**7.42.3.20 setCircle()** [1/2]

```
template< typename T > void setCircle (
    const T x,
    const T y,
    const T r ) [inline]
```

Sets the center and radius of the [Circle](#).

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>x</i>	The x coordinate of center.
<i>y</i>	The y coordinate of center.
<i>r</i>	The radius.

**Note**

The area containing the [Circle](#) is not invalidated.

**7.42.3.21 setCircle()** [2/2]

```
void setCircle (
    const int16_t x,
    const int16_t y,
    const int16_t r ) [inline]
```

Sets the center and radius of the [Circle](#).

**Parameters**

<i>x</i>	The x coordinate of center.
<i>y</i>	The y coordinate of center.
<i>r</i>	The radius.

**Note**

The area containing the [Circle](#) is not invalidated.

**7.42.3.22 setLineWidth()**

```
template< typename T > void setLineWidth (
    const T width ) [inline]
```

Sets the line width for this [Circle](#). If the line width is set to zero, the circle will be filled.

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

<i>width</i>	The width of the line measured in pixels.
--------------	-------------------------------------------

## Note

The area containing the [Circle](#) is not invalidated.  
if the new line width is smaller than the old width, the circle should be invalidated before updating the width to ensure that the old circle is completely erased.

## 7.42.3.23 setPrecision()

```
void setPrecision (
    const int precision )
```

Sets a precision of the [Circle](#) drawing function. The number given as precision is the number of degrees used as step counter when drawing smaller line fragments around the circumference of the circle, five being a sensible value. Higher values results in less nice circles but faster rendering. Large circles might need a precision smaller than five.

## Parameters

<i>precision</i>	The precision measured in degrees.
------------------	------------------------------------

## Note

The circle is not invalidated.

## 7.42.3.24 setRadius()

```
template< typename T > void setRadius (
    const T r ) [inline]
```

Sets the radius of the [Circle](#).

## Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

## Parameters

<i>r</i>	The radius.
----------	-------------

**Note**

The area containing the [Circle](#) is not invalidated.

**7.42.3.25 updateArcEnd()**

```
template< typename T > void updateArcEnd (
    const T endAngle ) [inline]
```

Updates the end angle in degrees for this [Circle](#) arc.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>endAngle</i>	The end angle in degrees.
-----------------	---------------------------

**Note**

The area containing the updated [Circle](#) arc is invalidated.

**See also**

[setArc](#)  
[updateArcStart](#)

**7.42.3.26 updateArcStart()**

```
template< typename T > void updateArcStart (
    const T startAngle ) [inline]
```

Updates the start angle in degrees for this [Circle](#) arc.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>startAngle</i>	The start angle in degrees.
-------------------	-----------------------------

**Note**

The area containing the updated [Circle](#) arc is invalidated.



See also

[setArc](#)  
[updateArcEnd](#)

## 7.43 CircleProgress Class Reference

A circle progress.

```
#include <touchgfx/containers/progress_indicators/CircleProgress.hpp>
```

### Public Member Functions

- [CircleProgress](#) ()  
*Default constructor.*
- virtual [~CircleProgress](#) ()  
*Destructor.*
- virtual void [setProgressIndicatorPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets the position and dimension of the circle progress indicator.*
- virtual void [setPainter](#) ([AbstractPainter](#) &painter)  
*Sets the painter.*
- virtual void [setCenter](#) (int x, int y)  
*Sets the center.*
- virtual void [getCenter](#) (int &x, int &y) const  
*Gets the center coordinates.*
- virtual void [setRadius](#) (int r)  
*Sets the radius.*
- virtual int [getRadius](#) () const  
*Gets the radius.*
- virtual void [setLineWidth](#) (int width)  
*Sets line width.*
- virtual int [getLineWidth](#) () const  
*Gets line width.*
- virtual void [setCapPrecision](#) (int precision)  
*Sets the cap precision.*
- virtual int [getCapPrecision](#) () const  
*Gets the cap precision.*
- virtual void [setStartEndAngle](#) (int startAngle, int endAngle)  
*Sets start and end angle.*
- virtual int [getStartAngle](#) () const  
*Gets start angle.*
- virtual int [getEndAngle](#) () const  
*Gets end angle.*
- virtual void [setAlpha](#) (uint8\_t alpha)  
*Sets the alpha.*
- virtual uint8\_t [getAlpha](#) () const  
*Gets the alpha.*
- virtual void [setValue](#) (int value)  
*Sets a value.*

## Protected Attributes

- [Circle circle](#)  
*The circle.*
- `int` [circleEndAngle](#)  
*The circle end angle.*

## Additional Inherited Members

### 7.43.1 Detailed Description

A circle progress indicator uses CWR for drawing the arc of a circle to show progress, and so the user must create a painter for painting the circle. The circle progress is defined by setting the minimum and maximum angle of the arc.

See also

[Circle](#)

### 7.43.2 Constructor & Destructor Documentation

#### 7.43.2.1 CircleProgress()

```
CircleProgress ( )
```

Default constructor.

#### 7.43.2.2 ~CircleProgress()

```
~CircleProgress ( ) [virtual]
```

Destructor.

### 7.43.3 Member Function Documentation

#### 7.43.3.1 getAlpha()

```
uint8_t getAlpha ( ) const [virtual]
```

Gets the alpha of the circle.

Returns

The alpha.

See also

[setAlpha](#)

#### 7.43.3.2 getCapPrecision()

```
int getCapPrecision ( ) const [inline], [virtual]
```

Gets the cap precision.

##### Returns

The cap precision.

#### 7.43.3.3 getCenter()

```
void getCenter (
    int & x,
    int & y ) const [virtual]
```

Gets the center coordinates.

##### Parameters

out	<i>x</i>	The x coordinate.
out	<i>y</i>	The y coordinate.

#### 7.43.3.4 getEndAngle()

```
int getEndAngle ( ) const [virtual]
```

Gets end angle. Beware that the value returned is not related to the current progress of the circle but rather the end point of the circle when it is at 100%.

##### Returns

The end angle.

##### See also

[setStartEndAngle](#)

[setEndAngle](#)

#### 7.43.3.5 getLineWidth()

```
int getLineWidth ( ) const [virtual]
```

Gets line width.

##### Returns

The line width.

##### See also

[setLineWidth](#)

#### 7.43.3.6 `getRadius()`

```
int getRadius ( ) const [virtual]
```

Gets the radius.

##### Returns

The radius.

#### 7.43.3.7 `getStartAngle()`

```
int getStartAngle ( ) const [virtual]
```

Gets start angle.

##### Returns

The start angle.

##### See also

[setStartEndAngle](#)  
[getEndAngle](#)

#### 7.43.3.8 `setAlpha()`

```
void setAlpha (
    uint8_t alpha ) [virtual]
```

Sets the alpha of the [Circle](#). Please note, that the alpha can also be set on the Painter, but this can be controlled directly from the user app, setting alpha for the [CircleProgress](#) will set the alpha of the actual circle.

##### Parameters

<i>alpha</i>	The alpha.
--------------	------------

##### See also

[getAlpha](#)

#### 7.43.3.9 `setCapPrecision()`

```
void setCapPrecision (
    int precision ) [virtual]
```

Sets the cap precision of the circle arc. This is not used if line width is zero.

##### Parameters

<i>precision</i>	The cap precision.
------------------	--------------------

See also

[Circle::setCapPrecision](#)

#### 7.43.3.10 setCenter()

```
void setCenter (
    int x,
    int y ) [virtual]
```

Sets the center of the circle / arc.

Parameters

<i>x</i>	The int to process.
<i>y</i>	The int to process.

#### 7.43.3.11 setLineWidth()

```
void setLineWidth (
    int width ) [virtual]
```

Sets line width of the circle. If a line width of zero is specified, it has a special meaning of drawing a filled circle instead of just the circle arc.

Parameters

<i>width</i>	The width.
--------------	------------

See also

[Circle::setLineWidth](#)

#### 7.43.3.12 setPainter()

```
void setPainter (
    AbstractPainter & painter ) [virtual]
```

Sets the painter to use for drawing.

Parameters

<i>in, out</i>	<i>painter</i>	The painter.
----------------	----------------	--------------

See also

[Circle::setPainter](#)  
[AbstractPainter](#)

#### 7.43.3.13 `setProgressIndicatorPosition()`

```
void setProgressIndicatorPosition (
    int16_t x,
    int16_t y,
    int16_t width,
    int16_t height ) [virtual]
```

Sets the position and dimension of the circle progress indicator relative to the background image.

##### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>width</i>	The width of the circle progress indicator.
<i>height</i>	The height of the circle progress indicator.

Reimplemented from [AbstractProgressIndicator](#).

#### 7.43.3.14 `setRadius()`

```
void setRadius (
    int r ) [virtual]
```

Sets the radius of the circle.

##### Parameters

<i>r</i>	The int to process.
----------	---------------------

##### See also

[Circle::setRadius](#)

#### 7.43.3.15 `setStartEndAngle()`

```
void setStartEndAngle (
    int startAngle,
    int endAngle ) [virtual]
```

Sets start and end angle. By swapping end and start angles, circles can be drawn backwards.

##### Parameters

<i>startAngle</i>	The start angle.
<i>endAngle</i>	The end angle.

## 7.43.3.16 setValue()

```
virtual void setValue (
    int value ) [virtual]
```

Sets the current value in the range (min..max) set by [setRange\(\)](#). Values lower than min are mapped to min, values higher than max are mapped to max.

## Parameters

<i>value</i>	The value.
--------------	------------

Reimplemented from [AbstractProgressIndicator](#).

## 7.44 ClickButtonTrigger Class Reference

A click button trigger.

```
#include <touchgfx/containers/buttons/ClickButtonTrigger.hpp>
```

### Public Member Functions

- [ClickButtonTrigger](#) ()  
*Default constructor.*
- virtual [~ClickButtonTrigger](#) ()  
*Destructor.*
- virtual void [handleClickEvent](#) (const [ClickEvent](#) &event)  
*Handles the click event described by event.*

### Additional Inherited Members

#### 7.44.1 Detailed Description

A click button trigger. This trigger will create a button that reacts on clicks. This means it will call the action when it gets a touch released event.

The [ClickButtonTrigger](#) can be combined with one or more of the [ButtonStyle](#) classes to create a functional button.

#### 7.44.2 Member Function Documentation

## 7.44.2.1 handleClickEvent()

```
void handleClickEvent (
    const ClickEvent & event ) [inline], [virtual]
```

Handles the click event described by event. The action callback is called when receiving a RELEASED event in PRESSED state.

## Parameters

<i>event</i>	The event.
--------------	------------

Reimplemented from [Drawable](#).

## 7.45 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.*

#### 7.45.1 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](#)

#### 7.45.2 Member Enumeration Documentation



#### 7.45.2.1 ClickEventType

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.

## 7.45.3 Constructor &amp; Destructor Documentation

## 7.45.3.1 ClickEvent()

```
ClickEvent (
    ClickEventType type,
    int16_t x,
    int16_t y,
    int16_t force = 0 ) [inline]
```

Constructor.

## Parameters

<i>type</i>	The type of the click event.
<i>x</i>	The x coordinate of the click event.
<i>y</i>	The y coordinate of the click event.
<i>force</i>	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.

## 7.45.3.2 ~ClickEvent()

```
~ClickEvent ( ) [inline], [virtual]
```

Destructor.

## 7.45.4 Member Function Documentation

## 7.45.4.1 getEventType()

```
Event::EventType getEventType ( ) [inline], [virtual]
```

Gets event type.

## Returns

The type of this event.

Implements [Event](#).

#### 7.45.4.2 getForce()

```
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.

#### 7.45.4.3 getType()

```
ClickEventType getType ( ) const [inline]
```

Gets the click type of this event.

##### Returns

The click type of this event.

#### 7.45.4.4 getX()

```
int16_t getX ( ) const [inline]
```

Gets the x coordinate of this event.

##### Returns

The x coordinate of this event.

#### 7.45.4.5 getY()

```
int16_t getY ( ) const [inline]
```

Gets the y coordinate of this event.

##### Returns

The y coordinate of this event.

#### 7.45.4.6 setType()

```
void setType (
    ClickEventType type ) [inline]
```

Sets the click type of this event.

##### Parameters

<i>type</i>	The type to set.
-------------	------------------

#### 7.45.4.7 setX()

```
void setX (
    int16_t x ) [inline]
```

Sets the x coordinate of this event.

##### Parameters

<i>x</i>	The x coordinate of this event.
----------	---------------------------------

#### 7.45.4.8 setY()

```
void setY (
    int16_t y ) [inline]
```

Sets the y coordinate of this event.

##### Parameters

<i>y</i>	The y coordinate of this event.
----------	---------------------------------

## 7.46 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.*

### 7.46.1 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

<i>T</i>	specifies the type to extend with the <a href="#">ClickListener</a> behavior.
----------	-------------------------------------------------------------------------------

### 7.46.2 Constructor & Destructor Documentation

#### 7.46.2.1 ClickListener()

```
ClickListener ( ) [inline]
```

Default constructor.

### 7.46.3 Member Function Documentation

#### 7.46.3.1 handleClickEvent()

```
void handleClickEvent (
    const ClickEvent & event ) [inline], [virtual]
```

Ensures that the clickEvent is propagated to the super class T and to the clickAction listener.

#### Parameters

<i>event</i>	Information about the click.
--------------	------------------------------

#### See also

[Drawable::handleClickEvent\(\)](#)

#### 7.46.3.2 setClickAction()

```
void setClickAction (
    GenericCallback< const T &, const ClickEvent & > & callback ) [inline]
```

Associates an action to be performed when the class T is clicked.

#### Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to T.
-----------------	---------------------------------------------------------------------------

See also

[GenericCallback](#)

## 7.47 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.*

#### 7.47.1 Detailed Description

Contains functionality for color conversion.

#### 7.47.2 Member Function Documentation

##### 7.47.2.1 [getBlueColor\(\)](#)

```
static inline uint8_t getBlueColor (
    colortype color ) [inline], [static]
```

Gets the blue color part of a color. As this function must work for all color depths, it can be somewhat slow if used in speed critical sections. Consider finding the color in another way, if possible.

##### Parameters

<i>color</i>	The 16 bit color value.
--------------	-------------------------

##### Returns

The blue part of the color.

##### 7.47.2.2 [getColorFrom24BitRGB\(\)](#)

```
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

<i>red</i>	Value of the red part (0-255).
<i>green</i>	Value of the green part (0-255).
<i>blue</i>	Value of the blue part (0-255).

#### Returns

The color, encoded in a 16- or 24-bit representation depending on [LCD](#) color depth.

#### 7.47.2.3 getGreenColor()

```
static inline uint8_t getGreenColor (  
    colortype color ) [inline], [static]
```

Gets the green color part of a color. As this function must work for all color depths, it can be somewhat slow if used in speed critical sections. Consider finding the color in another way, if possible.

#### Parameters

<i>color</i>	The 16 bit color value.
--------------	-------------------------

#### Returns

The green part of the color.

#### 7.47.2.4 getRedColor()

```
static inline uint8_t getRedColor (  
    colortype color ) [inline], [static]
```

Gets the red color part of a color. As this function must work for all color depths, it can be somewhat slow if used in speed critical sections. Consider finding the color in another way, if possible.

#### Parameters

<i>color</i>	The color value.
--------------	------------------

**Returns**

The red part of the color.

## 7.48 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::getBlueColor\(\)](#) 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.*

### 7.48.1 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.

### 7.48.2 Constructor & Destructor Documentation

#### 7.48.2.1 colortype() [1/2]

```
colortype ( ) [inline]
```

Default constructor. Creates a black (0) color.

#### 7.48.2.2 colortype() [2/2]

```
colortype (
    uint32_t col ) [inline]
```

Constructor 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**

<i>col</i>	The color.
------------	------------



See also

[Color::getColorFrom24BitRGB\(\)](#)

### 7.48.3 Member Function Documentation

#### 7.48.3.1 getColor32()

```
uint32_t getColor32 ( ) const [inline]
```

Returns

The color 32.

See also

[Color::getRedColor\(\)](#)  
[Color::getGreenColor\(\)](#)  
[Color::getBlueColor\(\)](#)

#### 7.48.3.2 operator uint16\_t()

```
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.

## 7.49 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, const [Unicode::UnicodeChar](#) fallbackChar, const [Unicode::UnicodeChar](#) ellipsisChar)  
*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 data 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

### 7.49.1 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](#)

### 7.49.2 Constructor & Destructor Documentation

#### 7.49.2.1 ConstFont()

```
ConstFont (
    const GlyphNode * list,
    uint16\_t size,
    uint16\_t height,
    uint8\_t pixBelowBase,
    uint8\_t bitsPerPixel,
    uint8\_t maxLeft,
    uint8\_t maxRight,
    const Unicode::UnicodeChar fallbackChar,
    const Unicode::UnicodeChar ellipsisChar )
```

Creates a font instance.

#### Parameters

<i>list</i>	The array of glyphs known to this font.
<i>size</i>	The number of glyphs in list.
<i>height</i>	The height in pixels of the highest character in this font.
<i>pixBelowBase</i>	The maximum number of pixels that can be drawn below the baseline in this font.
<i>bitsPerPixel</i>	The number of bits per pixel in this font.
<i>maxLeft</i>	The maximum a character extends to the left.

## Parameters

<i>maxRight</i>	The maximum a character extends to the right.
<i>fallbackChar</i>	The fallback character for the typography in case no glyph is available.
<i>ellipsisChar</i>	The ellipsis character used for truncating long texts.

## 7.49.3 Member Function Documentation

## 7.49.3.1 find()

```
const GlyphNode * find (
    Unicode::UnicodeChar unicode ) const [protected]
```

Finds the glyph data associated with the specified unicode.

## Parameters

<i>unicode</i>	The character to look up.
----------------	---------------------------

## Returns

A pointer to the glyph node or null if the glyph was not found.

## 7.49.3.2 getGlyph()

```
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

	<i>unicode</i>	The character to look up.
	<i>pixelData</i>	Pointer to the pixel data for the glyph if the glyph is found. This is set by this method.
out	<i>bitsPerPixel</i>	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](#).

### 7.49.3.3 getKerning()

```
int8_t getKerning (
    Unicode::UnicodeChar prevChar,
    const GlyphNode * glyph ) const [pure virtual]
```

Gets the kerning distance between two characters.

#### Parameters

<i>prevChar</i>	The unicode value of the previous character.
<i>glyph</i>	the glyph object for the current character.

#### Returns

The kerning distance between prevChar and glyph char.

Reimplemented from [Font](#).

Implemented in [InternalFlashFont](#).

### 7.49.3.4 getPixelData()

```
const uint8_t * getPixelData (
    const GlyphNode * glyph ) const [pure virtual]
```

Gets the pixel date associated with this glyph.

#### Parameters

<i>glyph</i>	The glyph to get the pixels data from.
--------------	----------------------------------------

#### Returns

Pointer to the pixel data of this glyph.

Implemented in [InternalFlashFont](#).

## 7.50 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.*
- [Drawable](#) \* [getFirstChild](#) ()  
*Obtain a pointer to the first child of this container.*

### 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](#).*

### Additional Inherited Members

#### 7.50.1 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](#)

## 7.50.2 Constructor & Destructor Documentation

### 7.50.2.1 Container()

```
Container ( ) [inline]
```

Default constructor.

### 7.50.2.2 ~Container()

```
~Container ( ) [inline], [virtual]
```

Destructor.

## 7.50.3 Member Function Documentation

### 7.50.3.1 add()

```
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	<i>d</i>	The <a href="#">Drawable</a> to add.
----	----------	--------------------------------------

Reimplemented in [SlideMenu](#), [ScrollableContainer](#), [ModalWindow](#), [ListLayout](#), and [SwipeContainer](#).

### 7.50.3.2 contains()

```
bool contains (
    const Drawable & d ) [virtual]
```

Query if this object contains the given drawable.

#### Parameters

<i>d</i>	The <a href="#">Drawable</a> to look for.
----------	-------------------------------------------

#### Returns

True if the specified [Drawable](#) instance is direct child of this container, false otherwise.

## 7.50.3.3 draw()

```
void draw (
    const Rect & invalidatedArea ) const [virtual]
```

Draws the children of this container. Only used when JSMOC is disabled.

## Parameters

<i>invalidatedArea</i>	The area to draw.
------------------------	-------------------

Implements [Drawable](#).

Reimplemented in [Keyboard](#).

## 7.50.3.4 forEachChild()

```
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

<i>in</i>	<i>function</i>	The function to be executed for each child.
-----------	-----------------	---------------------------------------------

## See also

[ListLayout::insert](#) for a usage example.

## 7.50.3.5 getContainedArea()

```
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](#).

## 7.50.3.6 getFirstChild()

```
Drawable * getFirstChild ( ) [inline]
```

Useful if you want to manually iterate the children added to this container.

## Returns

Pointer to the first drawable added to this container. If nothing has been added return zero.

### 7.50.3.7 getLastChild()

```
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

	<i>x</i>	The x coordinate of the intersection.
	<i>y</i>	The y coordinate of the intersection.
out	<i>last</i>	out parameter in which the result is placed.

#### See also

[Drawable::getLastChild](#)

Implements [Drawable](#).

Reimplemented in [ScrollableContainer](#).

### 7.50.3.8 getSolidRect()

```
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](#).

### 7.50.3.9 getType()

```
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](#), [Slider](#), [Keyboard](#), and [ListLayout](#).

### 7.50.3.10 insert()

```
void insert (
    Drawable * previous,
    Drawable & d ) [virtual]
```



Inserts a [Drawable](#) instance after the specified child node. If previous is null, the drawable will be inserted as the first element in the list.

**Note**

As with add, do not add the same drawable twice.

**Parameters**

in	<i>previous</i>	The <a href="#">Drawable</a> to insert after. If null, insert as header.
in	<i>d</i>	The <a href="#">Drawable</a> to insert.

Reimplemented in [ListLayout](#).

**7.50.3.11 moveChildrenRelative()**

```
void moveChildrenRelative (
    int16_t deltaX,
    int16_t deltaY ) [protected], [virtual]
```

Calls moveRelative on all children.

**Parameters**

<i>deltaX</i>	Horizontal displacement.
<i>deltaY</i>	Vertical displacement.

Reimplemented in [ScrollableContainer](#).

**7.50.3.12 remove()**

```
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	<i>d</i>	The <a href="#">Drawable</a> to remove.
----	----------	-----------------------------------------

Reimplemented in [SlideMenu](#), [ModalWindow](#), [ListLayout](#), and [SwipeContainer](#).

**7.50.3.13 removeAll()**

```
void removeAll ( ) [virtual]
```

Removes all children by resetting their parent and sibling pointers.

Reimplemented in [ListLayout](#).

#### 7.50.3.14 setupDrawChain()

```
virtual void setupDrawChain (
    const Rect & invalidatedArea,
    Drawable ** nextPreviousElement ) [protected], [virtual]
```

Configure linked list for draw chain.

#### Note

For TouchGFX internal use only.

#### Parameters

	<i>invalidatedArea</i>	Include drawables that intersect with this area only.
<i>in, out</i>	<i>nextPreviousElement</i>	Modifiable element in linked list.

Reimplemented from [Drawable](#).

Reimplemented in [Keyboard](#).

#### 7.50.3.15 unlink()

```
void unlink ( ) [virtual]
```

Removes all children by unlinking the first child. The parent and sibling pointers of the children are not reset.

## 7.51 CoverTransition< templateDirection > Class Template Reference

A [Transition](#) that slides from one screen to the next.

```
#include <touchgfx/transitions/CoverTransition.hpp>
```

### Classes

- class [FullSolidRect](#)

A [Widget](#) that returns a solid rect of the same size as the application.

### Public Member Functions

- [CoverTransition](#) (const uint8\_t transitionSteps=20)  
*Constructor.*
- virtual [~CoverTransition](#) ()  
*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`.*

## Additional Inherited Members

### 7.51.1 Detailed Description

```
template<Direction templateDirection>
class touchgfx::CoverTransition< templateDirection >
```

A [Transition](#) that slides the new screen over the previous.

#### Template Parameters

<code>templateDirection</code>	Type of the template direction.
--------------------------------	---------------------------------

See also

[Transition](#)

### 7.51.2 Constructor & Destructor Documentation

#### 7.51.2.1 CoverTransition()

```
CoverTransition (
    const uint8_t transitionSteps = 20 ) [inline]
```

Constructor.

#### Parameters

<code>transitionSteps</code>	Number of steps in the transition animation.
------------------------------	----------------------------------------------

#### 7.51.2.2 ~CoverTransition()

```
~CoverTransition ( ) [inline], [virtual]
```

Destructor.

### 7.51.3 Member Function Documentation

#### 7.51.3.1 handleTickEvent()

```
void handleTickEvent ( ) [inline], [virtual]
```

Handles the tick event when transitioning. It moves the contents of the [Screen](#)'s container. The direction of the transition determines the direction the contents of the container moves.

Reimplemented from [Transition](#).

#### 7.51.3.2 init()

```
void init ( ) [inline], [virtual]
```

Initializes this object.

See also

[Transition::init\(\)](#)

Reimplemented from [Transition](#).

#### 7.51.3.3 initMoveDrawable()

```
void initMoveDrawable (
    Drawable & d ) [inline], [protected], [virtual]
```

Moves the [Drawable](#) to its initial position outside of the visible area.

Parameters

in	<i>d</i>	The <a href="#">Drawable</a> to move.
----	----------	---------------------------------------

#### 7.51.3.4 tearDown()

```
void tearDown ( ) [inline], [virtual]
```

Tear down.

See also

[Transition::tearDown\(\)](#)

Reimplemented from [Transition](#).

#### 7.51.3.5 tickMoveDrawable()

```
void tickMoveDrawable (
```

```
Drawable & d) [inline], [protected], [virtual]
```

Moves the [Drawable](#).

#### Parameters

in	d	The <a href="#">Drawable</a> to move.
----	---	---------------------------------------

## 7.52 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<typename T >  
static FORCE\_INLINE\_FUNCTION [Q5 toQ5](#) (T value)  
*Convert an integer to a fixed point number.*
- template<typename T >  
static FORCE\_INLINE\_FUNCTION [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 sine](#) ([Q5](#) 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 [Q15 cosine](#) ([Q5](#) 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<typename T >  
static int [angle](#) (T x, T y)  
*Find angle of a coordinate.*
- template<typename 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.*
- static [Q5 muldivQ5](#) ([Q5](#) factor1, [Q5](#) factor2, [Q5](#) divisor)  
*Multiply two [Q5](#)'s and divide by a [Q5](#) without overflowing the multiplication.*
- static [Q5 mulQ5](#) ([Q5](#) factor1, [Q5](#) factor2)  
*Multiply two [Q5](#)'s returning a new [Q5](#).*

### 7.52.1 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%29Widget](http://en.wikipedia.org/wiki/Q_%28number_format%29Widget)

### 7.52.2 Member Function Documentation

#### 7.52.2.1 [angle\(\)](#) [1/4]

```
template< typename T > static int angle (
    T x,
    T y ) [inline], [static]
```

##### Template Parameters

<i>T</i>	Generic type parameter (int or float).
----------	----------------------------------------

##### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

##### Returns

The angle of the coordinate.

#### 7.52.2.2 [angle\(\)](#) [2/4]

```
template< typename T > static int angle (
    T x,
    T y,
    T & d ) [inline], [static]
```

## Template Parameters

<i>T</i>	Generic type parameter (int or float).
----------	----------------------------------------

## Parameters

	<i>x</i>	The x coordinate.
	<i>y</i>	The y coordinate.
out	<i>d</i>	The distance from (0,0) to (x,y).

## Returns

The angle of the coordinate.

7.52.2.3 `angle()` [3/4]

```
static int angle (
    Q5 x,
    Q5 y ) [inline], [static]
```

## Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

## Returns

The angle of the coordinate.

7.52.2.4 `angle()` [4/4]

```
static int angle (
    Q5 x,
    Q5 y,
    Q5 & d ) [inline], [static]
```

## Parameters

	<i>x</i>	The x coordinate.
	<i>y</i>	The y coordinate.
out	<i>d</i>	The distance from (0,0) to (x,y).

## Returns

The angle.

### 7.52.2.5 arcsine()

```
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

<i>q10</i>	The 10.
------------	---------

#### Returns

An int8\_t.

### 7.52.2.6 cosine() [1/2]

```
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>i</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\(\)](#)

### 7.52.2.7 cosine() [2/2]

```
static Q15 cosine (
    Q5 i ) [inline], [static]
```

Find the value of cos(i) with 15 bits precision using the fact that  $\cos(i) = \sin(90 - i)$ .

#### Parameters

<i>i</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\(\)](#)

#### 7.52.2.8 muldivQ5()

```
static Q5 muldivQ5 (
    Q5 factor1,
    Q5 factor2,
    Q5 divisor ) [inline], [static]
```

Multiply two Q5's and divide by a Q5 without overflowing the multiplication (assuming that the final result can be stored in a Q5).

##### Parameters

<i>factor1</i>	The first factor.
<i>factor2</i>	The second factor.
<i>divisor</i>	The divisor.

##### Returns

$\text{factor1} * \text{factor2} / \text{divisor}$ .

#### 7.52.2.9 mulQ5()

```
static Q5 mulQ5 (
    Q5 factor1,
    Q5 factor2 ) [inline], [static]
```

Multiply two Q5's returning a new Q5 without overflowing.

##### Parameters

<i>factor1</i>	The first factor.
<i>factor2</i>	The second factor.

##### Returns

$\text{factor1} * \text{factor2}$ .

#### 7.52.2.10 sine() [1/2]

```
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  $(1 \ll 15)$  to get the rounded value of  $\sin(i)$ . By using this function, a complete circle can be drawn without the need for using floating point math.

**Parameters**

<i>i</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.

**7.52.2.11 sine()** [2/2]

```
static Q15 sine (
    Q5 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 sin(i). By using this function, a complete circle can be drawn without the need for using floating point math.

If the given degree is not an integer, the value is approximated by interpolation between sin(floor(i)) and sin(ceil(i)).

**Parameters**

<i>i</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.

**7.52.2.12 sqrtQ10()**

```
static Q5 sqrtQ10 (
    Q10 value ) [inline], [static]
```

**Parameters**

<i>value</i>	The value to find the square root of.
--------------	---------------------------------------

**Returns**

The square root of the given value.

**7.52.2.13 toQ10()**

```
template< typename T > FORCE_INLINE_FUNCTION 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

<i>T</i>	Should be either int or float.
----------	--------------------------------

## Parameters

<i>value</i>	the integer to convert.
--------------	-------------------------

## Returns

the converted integer.

## 7.52.2.14 toQ5()

```
template< typename T > FORCE_INLINE_FUNCTION 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

<i>T</i>	Should be either int or float.
----------	--------------------------------

## Parameters

<i>value</i>	the integer to convert.
--------------	-------------------------

## Returns

the converted integer.

## 7.53 DigitalClock Class Reference

A digital clock.

```
#include <touchgfx/containers/clock/DigitalClock.hpp>
```

## Public Types

- enum [DisplayMode](#) { [DISPLAY\\_12\\_HOUR\\_NO\\_SECONDS](#), [DISPLAY\\_24\\_HOUR\\_NO\\_SECONDS](#), [DISPLAY\\_12\\_HOUR](#), [DISPLAY\\_24\\_HOUR](#) }
- Values that represent different display modes.*

## Public Member Functions

- [DigitalClock](#) ()  
*Default constructor.*
- virtual [~DigitalClock](#) ()

*Destructor.*

- virtual void [setWidth](#) (int16\_t width)  
*Sets the width of the [DigitalClock](#).*
- virtual void [setHeight](#) (int16\_t height)  
*Sets the height of the [DigitalClock](#).*
- virtual void [setBaselineY](#) (int16\_t baselineY)  
*Adjusts the digital clocks y coordinate to place the text at the specified baseline.*
- virtual void [setTypedText](#) (TypedText typedText)  
*Sets the typed text of the [DigitalClock](#).*
- virtual void [setColor](#) (color\_type color)  
*Sets the color of the text.*
- virtual void [setDisplayMode](#) (DisplayMode dm)  
*Sets the display mode.*
- virtual DisplayMode [getDisplayMode](#) () const  
*Gets the display mode.*
- void [displayLeadingZeroForHourIndicator](#) (bool displayLeadingZero)  
*Sets whether to display a leading zero for the hour indicator or not.*
- virtual void [setAlpha](#) (uint8\_t alpha)  
*Sets the alpha value of the digital clock.*
- virtual uint8\_t [getAlpha](#) () const  
*Gets the alpha value of the digital clock.*
- virtual uint16\_t [getTextWidth](#) () const  
*Gets text width.*

### Protected Member Functions

- virtual void [updateClock](#) ()  
*Updates the visual representation of the clock.*

### Protected Attributes

- DisplayMode [displayMode](#)  
*The current display mode.*
- bool [useLeadingZeroForHourIndicator](#)  
*Print a leading zero if the hour is less than 10.*
- TextAreaWithOneWildcard [text](#)  
*The clock text.*
- Unicode::UnicodeChar [buffer](#) [BUFFER\_SIZE]  
*Wild card buffer for the clock text.*

### Static Protected Attributes

- static const int [BUFFER\\_SIZE](#) = 16  
*Buffer size of the wild card.*

### 7.53.1 Detailed Description

A digital clock. Can be set in either 12 or 24 hour mode. Seconds are optional. Width and height must be set manually to match the typography and alignment specified in the text database. The Digital Clock requires a typedText with one wildcard and uses the following characters (not including quotes) "AMP :0123456789" These must be present in the text database with the same typography as the wildcard text. Leading zero for the hour indicator can be enabled/disable by the displayLeadingZeroForHourIndicator method.

## 7.53.2 Member Enumeration Documentation

### 7.53.2.1 DisplayMode

enum `DisplayMode`

#### Enumerator

DISPLAY_12_HOUR_NO_SECONDS	12 Hour clock. Seconds are not displayed
DISPLAY_24_HOUR_NO_SECONDS	24 Hour clock. Seconds are not displayed
DISPLAY_12_HOUR	12 Hour clock. Seconds are displayed
DISPLAY_24_HOUR	24 Hour clock. Seconds are displayed

## 7.53.3 Constructor & Destructor Documentation

### 7.53.3.1 DigitalClock()

`DigitalClock ( )`

Default constructor.

### 7.53.3.2 ~DigitalClock()

`~DigitalClock ( )` [virtual]

Destructor.

## 7.53.4 Member Function Documentation

### 7.53.4.1 displayLeadingZeroForHourIndicator()

```
void displayLeadingZeroForHourIndicator (
    bool displayLeadingZero )
```

Sets whether to display a leading zero for the hour indicator or not. That is the if an hour value less than 10 will be displayed as "8:" or "08:".

Default value for this setting is false.

#### Parameters

<i>displayLeadingZero</i>	true = show leading zero. false = do not show leading zero.
---------------------------	-------------------------------------------------------------

#### 7.53.4.2 getAlpha()

```
uint8_t getAlpha ( ) const [virtual]
```

Gets the alpha value of the digital clock.

##### Returns

The alpha value. 255 = completely solid. 0 = invisible.

#### 7.53.4.3 getDisplayMode()

```
DisplayMode getDisplayMode ( ) const [inline], [virtual]
```

##### Returns

The display mode.

#### 7.53.4.4 getTextWidth()

```
uint16_t getTextWidth ( ) const [inline], [virtual]
```

##### Returns

The text width of the current content of the digital clock.

#### 7.53.4.5 setAlpha()

```
void setAlpha (
    uint8_t alpha ) [virtual]
```

Sets the alpha value of the digital clock.

##### Parameters

<i>alpha</i>	The alpha value. 255 = completely solid. 0 = invisible.
--------------	---------------------------------------------------------

#### 7.53.4.6 setBaselineY()

```
void setBaselineY (
    int16_t baselineY ) [virtual]
```

Adjusts the digital clocks 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

## Parameters

<i>baselineY</i>	The y coordinate of the baseline.
------------------	-----------------------------------

## 7.53.4.7 setColor()

```
void setColor (
    colortype color ) [virtual]
```

Sets the color of the text. Invalidates the [DigitalClock](#).

## Parameters

<i>color</i>	The new text color.
--------------	---------------------

## 7.53.4.8 setDisplayMode()

```
void setDisplayMode (
    DisplayMode dm ) [inline], [virtual]
```

## Parameters

<i>dm</i>	The new DisplayMode.
-----------	----------------------

## 7.53.4.9 setHeight()

```
void setHeight (
    int16_t height ) [virtual]
```

Sets the height of the [DigitalClock](#). The text area that displays the clock is expanded to match the dimension of the [DigitalClock](#).

## Parameters

<i>height</i>	The height.
---------------	-------------

Reimplemented from [Drawable](#).

## 7.53.4.10 setTypedText()

```
void setTypedText (
    TypedText typedText ) [virtual]
```

Sets the typed text of the [DigitalClock](#). Expects a typed text with one wildcard and that the following characters are

defined in the text spreadsheet (for the same typography):

AMP :0123456789

Invalidates the [DigitalClock](#).

#### Parameters

<i>typedText</i>	Describes the typed text to use.
------------------	----------------------------------

#### 7.53.4.11 `setWidth()`

```
void setWidth (
    int16_t width ) [virtual]
```

Sets the width of the [DigitalClock](#). The text area that displays the clock is expanded to match the dimension of the [DigitalClock](#).

#### Parameters

<i>width</i>	The width.
--------------	------------

Reimplemented from [Drawable](#).

#### 7.53.4.12 `updateClock()`

```
virtual void updateClock ( ) [protected], [virtual]
```

Updates the visual representation of the clock.

Implements [AbstractClock](#).

## 7.54 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.*

### 7.54.1 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.

### 7.54.2 Member Function Documentation

#### 7.54.2.1 [transformDisplayToFrameBuffer\(\)](#) [1/6]

```
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.

#### 7.54.2.2 [transformDisplayToFrameBuffer\(\)](#) [2/6]

```
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.

#### 7.54.2.3 [transformDisplayToFrameBuffer\(\)](#) [3/6]

```
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	<i>x</i>	the x part to translate.
in, out	<i>y</i>	the y part to translate.
	<i>in</i>	the rectangle defining the coordinate space.

**7.54.2.4 transformDisplayToFrameBuffer()** [4/6]

```
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	<i>x</i>	the x part to translate.
in, out	<i>y</i>	the y part to translate.
	<i>in</i>	the rectangle defining the coordinate space.

**7.54.2.5 transformDisplayToFrameBuffer()** [5/6]

```
static void transformDisplayToFrameBuffer (
    Rect & r ) [static]
```

Transform rectangle from display to frame buffer coordinates.

**Parameters**

in, out	<i>r</i>	the rectangle to translate.
---------	----------	-----------------------------

**7.54.2.6 transformDisplayToFrameBuffer()** [6/6]

```
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	<i>r</i>	the rectangle to translate.
	<i>in</i>	the rectangle defining the coordinate space.

## 7.54.2.7 transformFrameBufferToDisplay()

```
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.

## 7.55 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.*

### 7.55.1 Detailed Description

[DMA\\_Interface](#) provides basic functionality and structure for processing "blit" operations using DMA.

### 7.55.2 Constructor & Destructor Documentation

#### 7.55.2.1 ~DMA\_Interface()

```
~DMA_Interface ( ) [inline], [virtual]
```

Destructor.

#### 7.55.2.2 DMA\_Interface()

```
DMA_Interface (
    DMA_Queue & dmaQueue ) [inline], [protected]
```

Constructs a DMA Interface object.

## Parameters

in	<i>dmaQueue</i>	Reference to the queue of DMA operations.
----	-----------------	-------------------------------------------

### 7.55.3 Member Function Documentation

#### 7.55.3.1 addToQueue()

```
void addToQueue (
    const BlitOp & op ) [virtual]
```

Inserts a [BlitOp](#) for processing. This also potentially starts the DMA controller, if not already running.

## Parameters

<i>op</i>	The operation to add.
-----------	-----------------------

#### 7.55.3.2 disableAlpha()

```
void disableAlpha ( ) [protected], [virtual]
```

Configures blit-op hardware for solid operation (no alpha-blending)

#### 7.55.3.3 enableAlpha()

```
void enableAlpha (
    uint8_t alpha ) [protected], [virtual]
```

Configures blit-op hardware for alpha-blending.

## Parameters

<i>alpha</i>	The alpha-blending value to apply.
--------------	------------------------------------

#### 7.55.3.4 enableCopyWithTransparentPixels()

```
void enableCopyWithTransparentPixels (
    uint8_t alpha ) [protected], [virtual]
```

Configures blit-op hardware for alpha-blending while simultaneously skipping transparent pixels.

## Parameters

<i>alpha</i>	The alpha-blending value to apply.
--------------	------------------------------------

#### 7.55.3.5 execute()

```
void execute ( ) [protected], [virtual]
```

Performs a queued blit-op.

#### 7.55.3.6 executeCompleted()

```
void executeCompleted ( ) [protected], [virtual]
```

To be called when blit-op has been performed.

#### 7.55.3.7 flush()

```
void flush ( ) [inline], [virtual]
```

This function blocks until all DMA transfers in the queue have been completed.

Reimplemented in [NoDMA](#).

#### 7.55.3.8 getAllowed()

```
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.

#### 7.55.3.9 getBlitCaps()

```
BlitOperations getBlitCaps ( ) [pure virtual]
```

Gets the blit capabilities of this DMA.

##### Returns

The blit operations supported by this DMA implementation.

Implemented in [NoDMA](#).

#### 7.55.3.10 initialize()

```
void initialize ( ) [inline], [virtual]
```

Perform initialization. Does nothing in this base class.

#### 7.55.3.11 isDmaQueueEmpty()

```
uint8_t isDmaQueueEmpty ( )
```

Query if the DMA queue is empty.

**Returns**

1 if DMA queue is empty, else 0.

**7.55.3.12 isDmaQueueFull()**

```
uint8_t isDmaQueueFull ( )
```

Query if the DMA queue is full.

**Returns**

1 if DMA queue is full, else 0.

**7.55.3.13 isDMARunning()**

```
bool isDMARunning ( ) [inline]
```

Query if the DMA is running.

**Returns**

true if a DMA operation is currently in progress.

**7.55.3.14 seedExecution()**

```
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.

**7.55.3.15 setAllowed()**

```
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**

<i>allowed</i>	true if DMA transfers are allowed.
----------------	------------------------------------

**7.55.3.16 setupDataCopy()**

```
void setupDataCopy (
    const BlitOp & blitOp ) [protected], [pure virtual]
```

Configures blit-op hardware for a 2D copy as specified by blitOp.

**Parameters**

<i>blitOp</i>	The operation to execute.
---------------	---------------------------

Implemented in [NoDMA](#).

**7.55.3.17 setupDataFill()**

```
void setupDataFill (
    const BlitOp & blitOp ) [protected], [pure virtual]
```

Configures blit-op hardware for a 2D fill as specified by blitOp.

**Parameters**

<i>blitOp</i>	The operation to execute.
---------------	---------------------------

Implemented in [NoDMA](#).

**7.55.3.18 signalDMAInterrupt()**

```
void signalDMAInterrupt ( ) [pure virtual]
```

This function is called automatically by the framework when a DMA interrupt has been received.

Implemented in [NoDMA](#).

**7.55.3.19 start()**

```
void start ( ) [virtual]
```

Signals that DMA transfers can start. If any elements are in the queue, start it.

**7.55.3.20 waitForFrameBufferSemaphore()**

```
void waitForFrameBufferSemaphore ( ) [protected], [virtual]
```

Waits until frame buffer semaphore is available (i.e. neither DMA or application is accessing the frame buffer).

**7.56 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.*

### 7.56.1 Detailed Description

This class provides an interface for a FIFO (circular) list used by `DMA_Interface` and descendants for storing `BlitOp`'s.

### 7.56.2 Constructor & Destructor Documentation

#### 7.56.2.1 `~DMA_Queue()`

```
~DMA_Queue ( ) [inline], [virtual]
```

Destructor.

#### 7.56.2.2 `DMA_Queue()`

```
DMA_Queue ( ) [inline], [protected]
```

Default constructor.

### 7.56.3 Member Function Documentation

#### 7.56.3.1 first()

```
const BlitOp * first ( ) [protected], [pure virtual]
```

Gets the first element in the queue.

##### Returns

The first element in the queue.

Implemented in [LockFreeDMA\\_Queue](#).

#### 7.56.3.2 isEmpty()

```
bool isEmpty ( ) [pure virtual]
```

Query if this object is empty.

##### Returns

true if the queue is empty.

Implemented in [LockFreeDMA\\_Queue](#).

#### 7.56.3.3 isFull()

```
bool isFull ( ) [pure virtual]
```

Query if this object is full.

##### Returns

true if the queue is full.

Implemented in [LockFreeDMA\\_Queue](#).

#### 7.56.3.4 pop()

```
void pop ( ) [protected], [pure virtual]
```

Pops an element from the queue.

Implemented in [LockFreeDMA\\_Queue](#).

#### 7.56.3.5 pushCopyOf()

```
void pushCopyOf (
    const BlitOp & op ) [pure virtual]
```

Adds the specified blitop to the queue.

## Parameters

<i>op</i>	The blitop to add.
-----------	--------------------

Implemented in [LockFreeDMA\\_Queue](#).

## 7.57 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 y 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.*

#### 7.57.1 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](#)

## 7.57.2 Constructor & Destructor Documentation

### 7.57.2.1 DragEvent()

```
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

<i>type</i>	The type of the drag event.
<i>oldX</i>	The x coordinate of the drag start position (dragged from)
<i>oldY</i>	The y coordinate of the drag start position (dragged from)
<i>newX</i>	The x coordinate of the new position (dragged to)
<i>newY</i>	The y coordinate of the new position (dragged to)

### 7.57.2.2 ~DragEvent()

```
~DragEvent ( ) [inline], [virtual]
```

Destructor.

## 7.57.3 Member Function Documentation

### 7.57.3.1 getDeltaX()

```
int16_t getDeltaX ( ) const [inline]
```

Gets the distance in x coordinates (how long was the drag).

#### Returns

The distance of this drag event.

### 7.57.3.2 getDeltaY()

```
int16_t getDeltaY ( ) const [inline]
```

Gets the distance in y coordinates (how long was the drag).

**Returns**

The distance of this drag event.

**7.57.3.3 getEventType()**

```
Event::EventType getEventType ( ) [inline], [virtual]
```

Gets event type.

**Returns**

The type of this event.

Implements [Event](#).

**7.57.3.4 getNewX()**

```
int16_t getNewX ( ) const [inline]
```

Gets the new x coordinate (dragged to).

**Returns**

The new x coordinate (dragged to).

**7.57.3.5 getNewY()**

```
int16_t getNewY ( ) const [inline]
```

Gets the new y coordinate (dragged to).

**Returns**

The new y coordinate (dragged to).

**7.57.3.6 getOldX()**

```
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).

### 7.57.3.7 getOldY()

```
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).

### 7.57.3.8 getType()

```
DragEventType getType ( ) const [inline]
```

Gets the type of this drag event.

#### Returns

The type of this drag event.

## 7.58 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.*

### 7.58.1 Detailed Description

```
template<class T>
class touchgfx::Draggable< T >
```

Mix-in class that extends a class to become draggable.

#### Template Parameters

<i>T</i>	specifies the type to extend with the draggable behavior.
----------	-----------------------------------------------------------

### 7.58.2 Constructor & Destructor Documentation

## 7.58.2.1 Draggable()

```
Drawable ( ) [inline]
```

Default constructor.

## 7.58.2.2 ~Drawable()

```
~Drawable ( ) [inline], [virtual]
```

Destructor.

## 7.58.3 Member Function Documentation

## 7.58.3.1 handleDragEvent()

```
void handleDragEvent (
    const DragEvent & evt ) [inline], [virtual]
```

Called when dragging the draggable object. The object is moved according to the drag event.

## Parameters

<i>evt</i>	The drag event.
------------	-----------------

Reimplemented in [Snapper< T >](#).

## 7.59 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\_BUTTON, TYPE\_BUTTONWITHICON, TYPE\_BUTTONWITHLABEL, TYPE\_IMAGE, TYPE\_TILEDIMAGE, 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, TYPE\_CLICKABLECONTAINER }  
 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.*

### 7.59.1 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](#)

### 7.59.2 Member Enumeration Documentation

#### 7.59.2.1 DrawableType

```
enum enum DrawableType
```

Enum defining [Drawable](#) types. To be used by automated GUI testing to determine class type of a [Drawable](#) object.

### 7.59.3 Constructor & Destructor Documentation

#### 7.59.3.1 Drawable()

```
Drawable ( ) [inline]
```

Default constructor.

#### 7.59.3.2 ~Drawable()

```
~Drawable ( ) [inline], [virtual]
```

Destructor.

### 7.59.4 Member Function Documentation

## 7.59.4.1 childGeometryChanged()

```
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](#).

## 7.59.4.2 draw()

```
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

<i>invalidatedArea</i>	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](#), [BoxWithBorder](#), [ButtonWithLabel](#), [ScalableImage](#), [Container](#), [TiledImage](#), [ButtonWithIcon](#), [Box](#), [CanvasWidget](#), [TextureMapper](#), [TextAreaWithOneWildcard](#), [Image](#), [RadioButton](#), [SnapshotWidget](#), [CoverTransition< templateDirection >::FullSolidRect](#), [Button](#), [PixelDataWidget](#), and [TouchArea](#).

## 7.59.4.3 getAbsoluteRect()

```
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.

## 7.59.4.4 getCachedAbsX()

```
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.

#### 7.59.4.5 getCachedAbsY()

```
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.

#### 7.59.4.6 getCachedVisibleRect()

```
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.

#### 7.59.4.7 getHeight()

```
int16_t getHeight ( ) const [inline]
```

Gets the height of this drawable.

##### Returns

The height of this drawable.

#### 7.59.4.8 getLastChild()

```
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

	<i>x</i>	The point of intersection expressed in coordinates relative to the parent.
	<i>y</i>	The point of intersection expressed in coordinates relative to the parent.
out	<i>last</i>	Result will be placed here.

Implemented in [ScrollableContainer](#), [Container](#), and [Widget](#).

#### 7.59.4.9 getNextSibling()

```
Drawable * getNextSibling ( ) [inline]
```

Returns the next sibling node. This will be the next [Drawable](#) that has been added to the same [Container](#) as this [Drawable](#).

##### Returns

The next sibling. If this is the last sibling, the return value is 0.

#### 7.59.4.10 getParent()

```
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.

#### 7.59.4.11 getRect()

```
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.

#### 7.59.4.12 getSolidRect()

```
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](#), [TiledImage](#), [ButtonWithLabel](#), [TextureMapper](#), [Image](#), [RadioButton](#), [TouchArea](#), [Button](#), [SnapshotWidget](#), [Box](#), [BoxWithBorder](#), [PixelDataWidget](#), [TextArea](#), and [CoverTransition< templateDirection >::FullSolidRect](#).

#### 7.59.4.13 `getSolidRectAbsolute()`

```
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.

#### 7.59.4.14 `getType()`

```
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`, `TextArea`, `ZoomAnimationImage`, `ScrollableContainer`, `Slider`, `TextAreaWithTwoWildcards`, `Keyboard`, `RadioButton`, `AnimatedImage`, `Container`, `ScalableImage`, `BoxWithBorder`, `TiledImage`, `TextAreaWithOneWildcard`, `ButtonWithLabel`, `Box`, `ButtonWithIcon`, `ListLayout`, `Image`, `SnapshotWidget`, `Button`, `TouchArea`, `ToggleButton`, `AbstractButton`, and `Widget`.

#### 7.59.4.15 `getVisibleRect()`

```
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.
-----	------	------------------------------------------------------------------

#### 7.59.4.16 `getWidth()`

```
int16_t getWidth ( ) const [inline]
```

Gets the width of this drawable.

**Returns**

The width of this drawable.

**7.59.4.17 getX()**

```
int16_t getX ( ) const [inline]
```

Gets the x coordinate of this drawable.

**Returns**

The x value, relative to the parent.

**7.59.4.18 getY()**

```
int16_t getY ( ) const [inline]
```

Gets the y coordinate of this drawable.

**Returns**

The y value, relative to the parent.

**7.59.4.19 handleClickEvent()**

```
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**

<i>evt</i>	The <a href="#">ClickEvent</a> received from the <a href="#">HAL</a> .
------------	------------------------------------------------------------------------

Reimplemented in [Slider](#), [Keyboard](#), [ScrollableContainer](#), [ScrollList](#), [ScrollWheelBase](#), [RepeatButton](#), [RepeatButtonTrigger](#), [ToggleButton](#), [ToggleButtonTrigger](#), [RadioButton](#), [TouchArea](#), [AbstractButton](#), [ClickButtonTrigger](#), [TouchButtonTrigger](#), and [SwipeContainer](#).

**7.59.4.20 handleDragEvent()**

```
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**

<i>evt</i>	The <a href="#">DragEvent</a> received from the <a href="#">HAL</a> .
------------	-----------------------------------------------------------------------

Reimplemented in [ScrollBase](#), [Slider](#), [Keyboard](#), [ScrollableContainer](#), [ScrollWheelBase](#), [TouchArea](#), and [SwipeContainer](#).

#### 7.59.4.21 handleGestureEvent()

```
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

<i>evt</i>	The <a href="#">GestureEvent</a> received from the <a href="#">HAL</a> .
------------	--------------------------------------------------------------------------

Reimplemented in [ScrollBase](#), [ScrollableContainer](#), [ScrollWheelBase](#), and [SwipeContainer](#).

#### 7.59.4.22 handleTickEvent()

```
void handleTickEvent ( ) [inline], [virtual]
```

Called periodically by the framework if the [Drawable](#) instance has subscribed to timer ticks.

##### See also

[Application::registerTimerWidget](#)

Reimplemented in [ScrollBase](#), [SlideMenu](#), [MoveAnimator< touchgfx::Container >](#), [ScrollableContainer](#), [ZoomAnimationImage](#), [AnimationTextureMapper](#), [RepeatButtonTrigger](#), [RepeatButton](#), [AnimatedImage](#), and [SwipeContainer](#).

#### 7.59.4.23 invalidate()

```
void invalidate ( ) const [virtual]
```

Tell the framework that this entire drawable needs to be redrawn.

##### See also

[invalidateRect](#)

Reimplemented in [CanvasWidget](#).

#### 7.59.4.24 invalidateRect()

```
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	<i>invalidatedArea</i>	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).
----	------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## 7.59.4.25 isTouchable()

```
bool isTouchable ( ) const [inline]
```

Gets whether this [Drawable](#) receives touch events or not.

## Returns

True if touch events are received.

## See also

[setTouchable](#)

## 7.59.4.26 isVisible()

```
bool isVisible ( ) const [inline]
```

Gets whether this [Drawable](#) is visible.

## Returns

true if the [Drawable](#) is visible.  
True if visible.

## See also

[setVisible](#)

## 7.59.4.27 moveRelative()

```
void moveRelative (
    int16_t x,
    int16_t y ) [virtual]
```

Moves the drawable.

## Note

Will redraw the appropriate areas of the screen.

## Parameters

<i>x</i>	The relative position to move to.
<i>y</i>	The relative position to move to.

#### 7.59.4.28 moveTo()

```
void moveTo (
    int16_t x,
    int16_t y ) [inline], [virtual]
```

Moves the drawable.

##### Note

Will redraw the appropriate areas of the screen.

##### Parameters

<i>x</i>	The absolute position to move to.
<i>y</i>	The absolute position to move to.

#### 7.59.4.29 resetDrawChainCache()

```
void resetDrawChainCache ( ) [inline], [protected]
```

Reset cached coordinate data.

##### Note

For TouchGFX internal use only.

#### 7.59.4.30 setHeight()

```
void setHeight (
    int16_t height ) [inline], [virtual]
```

Sets the height of this drawable.

##### Note

Changing this does not automatically yield a redraw.

##### Parameters

<i>height</i>	The new height.
---------------	-----------------

Reimplemented in [ZoomAnimationImage](#), [DrawableList](#), [DigitalClock](#), [ScrollWheelWithSelectionStyle](#), and [ScrollBase](#).

## 7.59.4.31 setPosition()

```
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**

<i>x</i>	The x coordinate of this <a href="#">Drawable</a> .
<i>y</i>	The y coordinate of this <a href="#">Drawable</a> .
<i>width</i>	The width of this <a href="#">Drawable</a> .
<i>height</i>	The height of this <a href="#">Drawable</a> .

Reimplemented in [ZoomAnimationImage](#).

## 7.59.4.32 setTouchable()

```
void setTouchable (
    bool touch ) [inline]
```

Controls whether this [Drawable](#) receives touch events or not.

**Parameters**

<i>touch</i>	If true it will receive touch events, if false it will not.
--------------	-------------------------------------------------------------

## 7.59.4.33 setupDrawChain()

```
void setupDrawChain (
    const Rect & invalidatedArea,
    Drawable ** nextPreviousElement ) [inline], [protected], [virtual]
```

Configure linked list for draw chain.

**Note**

For TouchGFX internal use only.

**Parameters**

	<i>invalidatedArea</i>	Include drawables that intersect with this area only.
<i>in, out</i>	<i>nextPreviousElement</i>	Modifiable element in linked list.

Reimplemented in [Keyboard](#), and [Container](#).

#### 7.59.4.34 setVisible()

```
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

<i>vis</i>	true if this <a href="#">Drawable</a> should be visible. By default, drawables are visible unless this function has been called with false as argument.
------------	---------------------------------------------------------------------------------------------------------------------------------------------------------

#### 7.59.4.35 setWidth()

```
void setWidth (
    int16_t width ) [inline], [virtual]
```

Sets the width of this drawable.

##### Note

Changing this does not automatically yield a redraw.

##### Parameters

<i>width</i>	The new width.
--------------	----------------

Reimplemented in [ZoomAnimationImage](#), [DrawableList](#), [DigitalClock](#), [ScrollBase](#), and [ScrollWheelWithSelectionStyle](#).

#### 7.59.4.36 setX()

```
void setX (
    int16_t x ) [inline], [virtual]
```

Sets the x coordinate of this drawable.

##### Note

Changing this does not automatically yield a redraw.

##### Parameters

<i>x</i>	The new x value, relative to the parent. A negative value is allowed.
----------	-----------------------------------------------------------------------

**7.59.4.37 setXY()**

```
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**

<i>x</i>	The new x value, relative to the parent. A negative value is allowed.
<i>y</i>	The new y value, relative to the parent. A negative value is allowed.

**7.59.4.38 setY()**

```
void setY (
    int16_t y ) [inline], [virtual]
```

Sets the y coordinate of this drawable.

**Note**

Changing this does not automatically yield a redraw.

**Parameters**

<i>y</i>	The new y value, relative to the parent. A negative value is allowed.
----------	-----------------------------------------------------------------------

**7.59.4.39 translateRectToAbsolute()**

```
void translateRectToAbsolute (
    Rect & r ) const [virtual]
```

Helper function for converting a specified subregion of this [Drawable](#) to absolute coordinates.

**Parameters**

<i>in, out</i>	<i>r</i>	The <a href="#">Rect</a> to translate.
----------------	----------	----------------------------------------

**7.60 DrawableList Class Reference**

A container able to display many items using only a few drawables.

```
#include <touchgfx/containers/DrawableList.hpp>
```

## Public Member Functions

- [DrawableList](#) ()  
*Default constructor.*
- virtual [~DrawableList](#) ()  
*Destructor.*
- virtual void [setWidth](#) (int16\_t width)  
*Sets the width of the [DrawableList](#).*
- virtual void [setHeight](#) (int16\_t height)  
*Sets the height of the [DrawableList](#). If the list is horizontal, the height is also propagated to all drawables in the list.*
- virtual void [setHorizontal](#) (bool horizontal)  
*Sets a horizontal layout.*
- virtual bool [getHorizontal](#) () const  
*Gets the orientation of the drawables.*
- virtual void [setCircular](#) (bool circular)  
*Sets whether the list is circular or not.*
- virtual bool [getCircular](#) () const  
*Gets the circular setting.*
- void [setDrawableSize](#) (int16\_t drawableSize, int16\_t drawableMargin)  
*Sets drawable size.*
- virtual int16\_t [getItemSize](#) () const  
*Gets size of each item.*
- virtual int16\_t [getDrawableSize](#) () const  
*Gets drawable size.*
- virtual int16\_t [getDrawableMargin](#) () const  
*Gets drawable margin.*
- virtual void [setDrawables](#) ([DrawableListItemsInterface](#) &drawableListItems, int16\_t drawableItemIndexOffset, [GenericCallback](#)< [DrawableListItemsInterface](#) \*, int16\_t, int16\_t > &updateDrawableCallback)  
*Sets the drawables parameters.*
- int16\_t [getNumberOfDrawables](#) () const  
*Gets number of drawables.*
- void [setNumberOfItems](#) (int16\_t numberOfItems)  
*Sets number of items in the list.*
- int16\_t [getNumberOfItems](#) () const  
*Gets number of items in the [DrawableList](#).*
- int16\_t [getRequiredNumberOfDrawables](#) () const  
*Gets required number of drawables.*
- void [setOffset](#) (int32\_t ofs)  
*Sets virtual coordinate.*
- int32\_t [getOffset](#) () const  
*Gets offset.*
- int16\_t [getItemIndex](#) (int16\_t drawableIndex)  
*Gets item stored in a given [Drawable](#).*
- int16\_t [getDrawableIndices](#) (int16\_t itemIndex, int16\_t \*drawableIndexArray, int16\_t arraySize)  
*Gets drawable indices.*
- int16\_t [getDrawableIndex](#) (int16\_t itemIndex, int16\_t prevDrawableIndex=-1)  
*Gets the drawable index of an item.*
- void [refreshDrawables](#) ()  
*Refresh drawables.*
- void [itemChanged](#) (int16\_t itemIndex)  
*Item changed.*

## Additional Inherited Members

### 7.60.1 Detailed Description

A container able to display many items using only a few drawables. This is done by only having drawables for visible items, and populating these drawables with new content when the drawable becomes visible.

This means that all drawables must have an identical structure in some way, for example an [Image](#) or a [Container](#) with a button and a text.

### 7.60.2 Constructor & Destructor Documentation

#### 7.60.2.1 DrawableList()

```
DrawableList ( )
```

Default constructor.

#### 7.60.2.2 ~DrawableList()

```
~DrawableList ( ) [inline], [virtual]
```

Destructor.

### 7.60.3 Member Function Documentation

#### 7.60.3.1 getCircular()

```
bool getCircular ( ) const [virtual]
```

Gets the circular setting, previously set using [setCircular\(\)](#).

##### Returns

True if the list is circular (infinite), false if the list is not circular.

##### See also

[setCircular](#)

#### 7.60.3.2 getDrawableIndex()

```
int16_t getDrawableIndex (
    int16_t itemIndex,
    int16_t prevDrawableIndex = -1 )
```

Gets the drawable index of an item. If the number of items is smaller than the number of drawables and the [DrawableList](#) is circular, the same item can be in more than one drawable. In that case, calling this function again with the previously returned index as second parameter, the index of the next drawable containing the item will be returned.

**Parameters**

<i>itemIndex</i>	Index of the item.
<i>prevDrawableIndex</i>	(Optional) Index of the previous drawable. If given, search starts after this index.

**Returns**

The first drawable index with the given item. Returns -1 if the item is not in a drawable.

**See also**

[getDrawableIndices](#)

**7.60.3.3 getDrawableIndices()**

```
int16_t getDrawableIndices (
    int16_t itemIndex,
    int16_t * drawableIndices,
    int16_t num )
```

Gets drawable indices. Useful when the number of items is smaller than the number of drawables as the same item might be in more than one drawable on the screen (if the [DrawableList](#) is circular). The passed array will be filled with the drawable indices and the number of indices found is returned.

**Parameters**

	<i>itemIndex</i>	Zero-based index of the item.
out	<i>drawableIndices</i>	Array where the drawable indices are stored.
	<i>num</i>	Size of drawable array.

**Returns**

The number of drawable indices found.

**See also**

[getFirstDrawableIndex](#)  
[setCircular](#)  
[getDrawableIndex](#)

**7.60.3.4 getDrawableMargin()**

```
int16_t getDrawableMargin ( ) const [virtual]
```

Gets drawable margin as set by [setDrawables](#).

**Returns**

The drawable margin.



### 7.60.3.5 getDrawableSize()

```
int16_t getDrawableSize ( ) const [virtual]
```

Gets drawable size as set by setDrawables.

#### Returns

The drawable size.

#### See also

[setDrawables](#)

### 7.60.3.6 getHorizontal()

```
bool getHorizontal ( ) const [virtual]
```

Gets the orientation of the drawables, previously set using [setHorizontal\(\)](#).

#### Returns

True if it horizontal, false if it is vertical.

#### See also

[setHorizontal](#)

### 7.60.3.7 getItemIndex()

```
int16_t getItemIndex (
    int16_t drawableIndex )
```

Gets item stored in a given [Drawable](#).

#### Parameters

<i>drawableIndex</i>	Zero-based index of the drawable.
----------------------	-----------------------------------

#### Returns

The item index.

### 7.60.3.8 getItemSize()

```
int16_t getItemSize ( ) const [virtual]
```

Gets size of each item. This equals the drawable size plus the drawable margin as set in [setDrawables\(\)](#). Equals [getDrawableSize\(\)](#) + [getDrawableMargin\(\)](#).

**Returns**

The item size.

**Note**

Not the same as [getDrawableSize\(\)](#).

**See also**

[setDrawables](#)  
[getDrawableSize](#)  
[getDrawableMargin](#)

**7.60.3.9 getNumberOfDrawables()**

```
int16_t getNumberOfDrawables ( ) const
```

Gets number of drawables, as set using [setDrawables\(\)](#).

**Returns**

The number of drawables.

**See also**

[setDrawables](#)

**7.60.3.10 getNumberOfItems()**

```
int16_t getNumberOfItems ( ) const
```

Gets number of items in the [DrawableList](#), as previously set using [setNumberOfItems\(\)](#).

**Returns**

The number of items.

**See also**

[setNumberOfItems](#)

**7.60.3.11 getOffset()**

```
int32_t getOffset ( ) const
```

Gets offset, as previously set using [setOffset\(\)](#).

**Returns**

The virtual offset.

**See also**

[setOffset](#)

#### 7.60.3.12 getRequiredNumberOfDrawables()

```
int16_t getRequiredNumberOfDrawables ( ) const
```

Gets required number of drawables. After setting up the [DrawableList](#) it is possible to request how many drawables are needed to ensure that the list can always be drawn properly. If the [DrawableList](#) has been setup with fewer Drawables than the required number of drawables, part of the lower part of the [DrawableList](#) will look wrong.

The number of required drawables depend on the size of the widget and the size of the drawables and the margin around drawables. If there are fewer drawables than required, the widget will not display correctly. If there are more drawables than required, some will be left unused.

##### Returns

The required number of drawables.

##### See also

[setDrawables](#)

#### 7.60.3.13 itemChanged()

```
void itemChanged (
    int16_t itemIndex )
```

Item changed and drawables containing this item must be updated. This function can be called when an item has changed and needs to be updated on screen. If the given item is displayed on screen, possibly more than once for cyclic lists, each drawable is request to refresh its content to reflect the new value.

##### Parameters

<i>itemIndex</i>	Zero-based index of the item.
------------------	-------------------------------

#### 7.60.3.14 refreshDrawables()

```
void refreshDrawables ( )
```

Refresh drawables. Useful to call if the number or items, their size or other properties have changed.

#### 7.60.3.15 setCircular()

```
void setCircular (
    bool circular ) [virtual]
```

Sets whether the list is circular (infinite) or not. A circular list is a list where the first drawable re-appears after the last item in the list.

##### Parameters

<i>circular</i>	True if the list should be circular, false if the list should not be circular.
-----------------	--------------------------------------------------------------------------------

See also

[getCircular](#)

#### 7.60.3.16 setDrawables()

```
void setDrawables (
    DrawableListItemsInterface & drawableListItems,
    int16_t drawableItemIndexOffset,
    GenericCallback< DrawableListItemsInterface *, int16_t, int16_t > & update↵
    DrawableCallback ) [virtual]
```

Sets the drawables parameters. These parameters are

- The access class to the array of drawables
- The offset in the drawableListItems array to start using drawable and
- [Callback](#) to update the contents of a drawable.

##### Parameters

in, out	<i>drawableListItems</i>	Number of drawables allocated.
in	<i>drawableItemIndexOffset</i>	A callback to get access to a drawable.
in	<i>updateDrawableCallback</i>	A callback to update the contents of a drawable.

See also

[getRequiredNumberOfDrawables](#)

#### 7.60.3.17 setDrawableSize()

```
void setDrawableSize (
    int16_t drawableSize,
    int16_t drawableMargin )
```

Sets drawable size. The total size of each drawable is the drawableSize + 2\*drawableMargin as margin will be added before and after each drawable.

##### Parameters

<i>drawableSize</i>	The size of the drawable.
<i>drawableMargin</i>	The margin around drawables (before as well as after the drawable).

#### 7.60.3.18 setHeight()

```
void setHeight (
    int16_t height ) [virtual]
```

## Parameters

<i>height</i>	The height.
---------------	-------------

Reimplemented from [Drawable](#).

## 7.60.3.19 setHorizontal()

```
void setHorizontal (
    bool horizontal ) [virtual]
```

Sets a horizontal layout. If horizontal is set true, all drawables are arranged side by side. If horizontal is set false, the drawables are arranged above and below each other (vertically).

## Parameters

<i>horizontal</i>	True to align drawables horizontal, false to align drawables vertically.
-------------------	--------------------------------------------------------------------------

## Note

Default value is false, i.e. vertical layout.

## See also

[getHorizontal](#)

## 7.60.3.20 setNumberOfItems()

```
void setNumberOfItems (
    int16_t numberOfItems )
```

Sets number of items in the list. This forces all drawables to be updated to ensure that the content is correct.

## Parameters

<i>numberOfItems</i>	Number of items.
----------------------	------------------

## Note

The [DrawableList](#) is refreshed to reflect the change.

## 7.60.3.21 setOffset()

```
void setOffset (
    int32_t ofs )
```

Sets virtual coordinate. Does not move to the given coordinate, but places the drawables and fill correct content into the drawables to give the impression that everything has been scrolled to the given coordinate.

Setting a value of 0 means that item 0 is at the start of the [DrawableList](#). Setting a value of "-getItemSize()" places item 0 outside the start of the [DrawableList](#) and item 1 at the start of it.

Items that are completely outside of view, will be updated with new content using the provided callback from [setDrawables\(\)](#). Care is taken to not fill drawables more than strictly required.

#### Parameters

<i>ofs</i>	The virtual coordinate.
------------	-------------------------

#### See also

[getOffset](#)  
[setDrawables](#)

#### 7.60.3.22 setWidth()

```
void setWidth (
    int16_t width ) [virtual]
```

Sets the width of the [DrawableList](#). If the list is vertical, the width is also propagated to all drawables in the list.

#### Parameters

<i>width</i>	The width.
--------------	------------

Reimplemented from [Drawable](#).

## 7.61 DrawableListItems< TYPE, SIZE > Class Template Reference

An array of drawables used by [DrawableList](#).

```
#include <touchgfx/containers/DrawableList.hpp>
```

### Public Member Functions

- [DrawableListItems](#) ()  
*Default constructor.*
- virtual [~DrawableListItems](#) ()  
*Destructor.*
- virtual [Drawable](#) \* [getDrawable](#) (int16\_t index)  
*Gets the address of an element.*
- TYPE & [operator\[\]](#) (int index)  
*Array indexer operator.*
- virtual int16\_t [getNumberOfDrawables](#) ()  
*Gets number of drawables.*

### Public Attributes

- TYPE [element](#) [SIZE]  
*The array of drawables.*

### 7.61.1 Detailed Description

```
template<class TYPE, int SIZE>
class touchgfx::DrawableListItems< TYPE, SIZE >
```

An array of drawables used by [DrawableList](#). This class is primarily used to ease the setup of a callback function to get access to a specific drawable in the array.

Example usage:

```
static const int NUMBER_OF_DRAWABLES = 5;
DrawableListItems<TextAreaWithOneWildcardContainer, NUMBER_OF_DRAWABLES> menuItems;
```

#### Template Parameters

<i>TYPE</i>	Type of the drawables. Can be a simple drawable, such as <a href="#">Image</a> or a more complex container.
<i>SIZE</i>	Size of the array. This is the number of drawables to allocate and should be all visible drawables on the screen at any given time.

### 7.61.2 Constructor & Destructor Documentation

#### 7.61.2.1 DrawableListItems()

```
DrawableListItems ( ) [inline]
```

Default constructor.

#### 7.61.2.2 ~DrawableListItems()

```
~DrawableListItems ( ) [inline], [virtual]
```

Destructor.

### 7.61.3 Member Function Documentation

#### 7.61.3.1 getDrawable()

```
Drawable * getDrawable (
    int16_t index ) [inline], [virtual]
```

Gets the address of an element.

#### Parameters

<i>index</i>	Zero-based index of the drawable.
--------------	-----------------------------------

**Returns**

The drawable for the given index.

Implements [DrawableListItemsInterface](#).

**7.61.3.2 getNumberOfDrawables()**

```
virtual int16_t getNumberOfDrawables ( ) [inline], [virtual]
```

Gets number of drawables.

**Returns**

The number of drawables.

Implements [DrawableListItemsInterface](#).

**7.61.3.3 operator[]()**

```
TYPE & operator[] (
    int index ) [inline]
```

Array indexer operator.

**Parameters**

<i>index</i>	Zero-based index of elements to access.
--------------	-----------------------------------------

**Returns**

The indexed value.

**7.62 DrawableListItemsInterface Class Reference**

A drawable list items interface.

```
#include <touchgfx/containers/DrawableList.hpp>
```

**Public Member Functions**

- virtual [~DrawableListItemsInterface](#) ()  
*Destructor.*
- virtual [Drawable](#) \* [getDrawable](#) (int16\_t index)=0  
*Gets a drawable.*
- virtual int16\_t [getNumberOfDrawables](#) ()=0  
*Gets number of drawables.*

**7.62.1 Detailed Description**

A drawable list items interface. Used to pass the allocated array of drawable elements to setDrawables function in either [ScrollList](#), [ScrollWheel](#) or [ScrollWheelWithSelectionStyle](#). Provides easy access to each element in the array



as well as the size of the array.

See also

[ScrollList::setDrawables](#)  
[ScrollWheel::setDrawables](#)  
[ScrollWheelWithSelectionMode::setDrawables](#)

## 7.62.2 Constructor & Destructor Documentation

### 7.62.2.1 ~DrawableListItemsInterface()

```
~DrawableListItemsInterface ( ) [inline], [virtual]
```

Destructor.

## 7.62.3 Member Function Documentation

### 7.62.3.1 getDrawable()

```
Drawable * getDrawable (
    int16_t index ) [pure virtual]
```

Gets a drawable at a given index.

Parameters

<i>index</i>	Zero-based index of the drawable.
--------------	-----------------------------------

Returns

Null if it fails, else the drawable.

Implemented in [DrawableListItems< TYPE, SIZE >](#).

### 7.62.3.2 getNumberOfDrawables()

```
int16_t getNumberOfDrawables ( ) [pure virtual]
```

Gets number of drawables.

Returns

The number of drawables.

Implemented in [DrawableListItems< TYPE, SIZE >](#).

## 7.63 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.*

## 7.64 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)*

### 7.64.1 Detailed Description

Data of a dynamic bitmap.

## 7.65 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:  $\sin(t)$*

### 7.65.1 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/>

### 7.65.2 Member Function Documentation

#### 7.65.2.1 backEaseIn()

```
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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

#### 7.65.2.2 backEaseInOut()

```
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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.3 backEaseOut()

```
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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.4 bounceEaseIn()

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.5 bounceEaseInOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.6 bounceEaseOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.7 circEaseIn()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.8 circEaseInOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.9 circEaseOut()**

```
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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.10 cubicEaseIn()

```

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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.11 cubicEaseInOut()

```

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

<i>t</i>	Time. The current time or step.
----------	---------------------------------



**Parameters**

<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.12 cubicEaseOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.13 elasticEaseIn()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.14 elasticEaseInOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.15 elasticEaseOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.16 expoEaseIn()**

```
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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.17 expoEaseInOut()

```

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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.18 expoEaseOut()

```

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

<i>t</i>	Time. The current time or step.
----------	---------------------------------

**Parameters**

<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.19 linearEaseIn()**

```
static int16_t linearEaseIn (
    uint16_t t,
    int16_t b,
    int16_t c,
    uint16_t d ) [static]
```

Simple linear tweening - no easing.

**Parameters**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.20 linearEaseInOut()**

```
static int16_t linearEaseInOut (
    uint16_t t,
    int16_t b,
    int16_t c,
    uint16_t d ) [static]
```

Simple linear tweening - no easing.

**Parameters**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.21 linearEaseNone()**

```
static int16_t linearEaseNone (
    uint16_t t,
    int16_t b,
    int16_t c,
    uint16_t d ) [static]
```

Simple linear tweening - no easing.

**Parameters**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.22 linearEaseOut()**

```
static int16_t linearEaseOut (
    uint16_t t,
    int16_t b,
    int16_t c,
    uint16_t d ) [static]
```

**Parameters**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.23 quadEaseIn()**

```
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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.24 quadEaseInOut()

```

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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.25 quadEaseOut()

```

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

<i>t</i>	Time. The current time or step.
----------	---------------------------------

**Parameters**

<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.26 quartEaseIn()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.27 quartEaseInOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.28 quartEaseOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.29 quintEaseIn()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.30 quintEaseInOut()**

```
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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.31 quintEaseOut()

```

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

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

#### Returns

The current value as a function of the current time or step.

#### 7.65.2.32 sineEaseIn()

```

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

<i>t</i>	Time. The current time or step.
----------	---------------------------------

**Parameters**

<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.33 sineEaseInOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

**Returns**

The current value as a function of the current time or step.

**7.65.2.34 sineEaseOut()**

```
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**

<i>t</i>	Time. The current time or step.
<i>b</i>	Beginning. The beginning value.
<i>c</i>	Change. The change between the beginning value and the destination value.
<i>d</i>	Duration. The total time or total number of steps.

## Returns

The current value as a function of the current time or step.

## 7.66 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.*

## 7.66.1 Constructor & Destructor Documentation

### 7.66.1.1 Edge()

```
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

<i>gradients</i>	The gradients for the triangle.
<i>vertices</i>	The vertices for the triangle.
<i>top</i>	The index in the vertices array of the top vertex of this edge.
<i>bottom</i>	The index in the vertices array of the bottom vertex of this edge.

## 7.66.2 Member Function Documentation

### 7.66.2.1 step() [1/2]

```
int step ( ) [inline]
```

Perform a step along the edge.

#### Returns

the Height.

### 7.66.2.2 step() [2/2]

```
int step (
    int steps ) [inline]
```

Performs a number of steps along the edge.

#### Parameters

<i>steps</i>	The number of steps the perform.
--------------	----------------------------------

**Returns**

height.

## 7.67 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.*

### 7.67.1 Detailed Description

Simple base class for events.

### 7.67.2 Member Enumeration Documentation

#### 7.67.2.1 EventType

```
enum enum EventType
```

The events types.

**Enumerator**

<code>EVENT_CLICK</code>	A click.
<code>EVENT_DRAG</code>	A drag.
<code>EVENT_GESTURE</code>	A gesture.

### 7.67.3 Constructor & Destructor Documentation

#### 7.67.3.1 ~Event()

```
~Event ( ) [inline], [virtual]
```

Destructor.

## 7.67.4 Member Function Documentation

### 7.67.4.1 `getEventType()`

```
EventType getEventType ( ) [pure virtual]
```

Gets event type.

#### Returns

The type of this event.

Implemented in [ClickEvent](#), [DragEvent](#), and [GestureEvent](#).

## 7.68 `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.*
- void [clearFadeAnimationEndedAction](#) ()  
*Clears the fade animation ended action previously set by [setFadeAnimationEndedAction](#).*
- 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.*
- virtual bool [isFadeAnimationRunning](#) () 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.*
- void [nextFadeAnimationStep](#) ()  
*Execute next step in fade animation.*

## 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](#).*

### 7.68.1 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

<i>T</i>	Specifies the type should have the fade animation capability.
----------	---------------------------------------------------------------

### 7.68.2 Constructor & Destructor Documentation

#### 7.68.2.1 FadeAnimator()

```
FadeAnimator ( ) [inline]
```

Default constructor. Creates and initialize the [FadeAnimator](#).

#### 7.68.2.2 ~FadeAnimator()

```
~FadeAnimator ( ) [inline], [virtual]
```

Destructor. Destroys the [FadeAnimator](#).

### 7.68.3 Member Function Documentation

#### 7.68.3.1 clearFadeAnimationEndedAction()

```
void clearFadeAnimationEndedAction ( ) [inline]
```

Clears the fade animation ended action previously set by setFadeAnimationEndedAction.

See also

[setFadeAnimationEndedAction](#)

#### 7.68.3.2 getFadeAnimationDelay()

```
uint16_t getFadeAnimationDelay ( ) const [inline], [virtual]
```

Gets the current animation delay.

Returns

The current animation delay.

#### 7.68.3.3 handleTickEvent()

```
void handleTickEvent ( ) [inline], [protected], [virtual]
```

The tick handler that handles the actual animation steps.

#### 7.68.3.4 isFadeAnimationRunning()

```
bool isFadeAnimationRunning ( ) const [inline], [virtual]
```

Gets whether or not the fade animation is running.

Returns

true if the fade animation is running.

#### 7.68.3.5 isRunning()

```
bool isRunning ( ) const [inline], [virtual]
```

Gets whether or not the fade animation is running.

Returns

true if the fade animation is running.



## 7.68.3.6 nextFadeAnimationStep()

```
void nextFadeAnimationStep ( ) [inline], [protected]
```

Execute next step in fade animation and stop the timer if necessary.

## 7.68.3.7 setFadeAnimationDelay()

```
void setFadeAnimationDelay (
    uint16_t delay ) [inline], [virtual]
```

Sets a delay on animations done by the [FadeAnimator](#).

## Parameters

<i>delay</i>	The delay in ticks.
--------------	---------------------

## 7.68.3.8 setFadeAnimationEndedAction()

```
void setFadeAnimationEndedAction (
    GenericCallback< const FadeAnimator< T > & > & callback ) [inline]
```

Associates an action to be performed when the animation ends.

## Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">FadeAnimator</a> .
-----------------	-----------------------------------------------------------------------------------------------------------

## See also

[GenericCallback](#)

## 7.68.3.9 startFadeAnimation()

```
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

<i>endAlpha</i>	The alpha value of T at animation end.
<i>duration</i>	The duration of the animation measured in ticks.
<i>alphaProgressionEquation</i>	The equation that describes the development of the alpha value during the animation. Default = <a href="#">EasingEquations::linearEaseNone</a> .

## 7.69 Font Class Reference

The font base class.

```
#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 [Unicode::UnicodeChar](#) [getFallbackChar](#) () const  
*Gets fallback character.*
- virtual [Unicode::UnicodeChar](#) [getEllipsisChar](#) () const  
*Gets ellipsis character.*
- 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, const `Unicode::UnicodeChar` fallbackChar, const `Unicode::UnicodeChar` ellipsisChar)  
*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.*
- `Unicode::UnicodeChar` `fallbackCharacter`  
*The fallback character to use when no glyph exists for the wanted character.*
- `Unicode::UnicodeChar` `ellipsisCharacter`  
*The ellipsis character used for truncating long texts.*

## Static Protected Attributes

- static `StringWidthFunctionPointer` `getStringWidthFunction`  
*The getStringWidth function, either LTR (supporting LTR only) or RTL (supporting RTL and LTR)*

### 7.69.1 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.

### 7.69.2 Constructor & Destructor Documentation

#### 7.69.2.1 `Font()`

```
Font (
    uint16_t height,
    uint8_t pixBelowBase,
    uint8_t bitsPerPixel,
    uint8_t maxLeft,
```

```
uint8_t maxRight,
const Unicode::UnicodeChar fallbackChar,
const Unicode::UnicodeChar ellipsisChar ) [inline], [protected]
```

The protected constructor of a [Font](#).

#### Parameters

<i>height</i>	The font height in pixels.
<i>pixBelowBase</i>	The number of pixels below the base line.
<i>bitsPerPixel</i>	The number of bits per pixel.
<i>maxLeft</i>	The maximum left extend for a glyph in the font.
<i>maxRight</i>	The maximum right extend for a glyph in the font.
<i>fallbackChar</i>	The fallback character for the typography in case no glyph is available.
<i>ellipsisChar</i>	The ellipsis character used for truncating long texts.

## 7.69.3 Member Function Documentation

### 7.69.3.1 getBitsPerPixel()

```
uint8_t getBitsPerPixel ( ) const [inline], [virtual]
```

Gets bits per pixel for this font.

#### Returns

The number of bits used per pixel in this font.

### 7.69.3.2 getCharWidth()

```
uint16_t getCharWidth (
    const Unicode::UnicodeChar c ) const [virtual]
```

Gets the width in pixels of the specified character.

#### Parameters

<i>c</i>	The unicode character.
----------	------------------------

#### Returns

The width in pixels of the specified character.

### 7.69.3.3 getEllipsisChar()

```
Unicode::UnicodeChar getEllipsisChar ( ) const [inline], [virtual]
```

Gets ellipsis character for the given font. This is the character which is used when truncating long lines.

**Returns**

The ellipsis character for the typography.

**7.69.3.4 getFallbackChar()**

```
Unicode::UnicodeChar getFallbackChar ( ) const [inline], [virtual]
```

Gets fallback character for the given font. In case there is no glyph for a character, use the glyph for the character returned by this function. If 0 (zero) is returned, there is no default character.

**Returns**

The default character for the typography in case no glyph is available.

**7.69.3.5 getFontHeight()**

```
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.

**7.69.3.6 getGlyph() [1/2]**

```
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**

	<i>unicode</i>	The character to look up.
	<i>pixelData</i>	Pointer to the pixel data for the glyph if the glyph is found. This is set by this method.
out	<i>bitsPerPixel</i>	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](#).

### 7.69.3.7 `getGlyph()` [2/2]

```
const GlyphNode * getGlyph (
    Unicode::UnicodeChar unicode ) const [inline], [virtual]
```

Gets the glyph data associated with the specified unicode.

#### Parameters

<i>unicode</i>	The character to look up.
----------------	---------------------------

#### Returns

A pointer to the glyph node or null if the glyph was not found.

### 7.69.3.8 `getKerning()`

```
int8_t getKerning (
    Unicode::UnicodeChar prevChar,
    const GlyphNode * glyph ) const [inline], [virtual]
```

Gets the kerning distance between two characters.

#### Parameters

<i>prevChar</i>	The unicode value of the previous character.
<i>glyph</i>	the glyph object for the current character.

#### Returns

The kerning distance between prevChar and glyph char.

Reimplemented in [ConstFont](#), and [InternalFlashFont](#).

### 7.69.3.9 `getMaxPixelsLeft()`

```
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.

### 7.69.3.10 `getMaxPixelsRight()`

```
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.

**7.69.3.11 getMaxTextHeight()**

```
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**

<i>text</i>	A zero-terminated unicode string.
...	Variable arguments providing additional information.

**Returns**

The height if the given text.

**7.69.3.12 getMinimumTextHeight()**

```
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.

**7.69.3.13 getNumberOfLines()**

```
uint16_t getNumberOfLines (
    const Unicode::UnicodeChar * text,
    ... ) const [virtual]
```

Count the number of lines in a given text.

**Parameters**

<i>text</i>	The text.
...	Variable arguments providing additional information.

**Returns**

The number of lines.

#### 7.69.3.14 getSpacingAbove()

```
uint8_t getSpacingAbove (
    const Unicode::UnicodeChar * text,
    ... ) const [virtual]
```

Gets the number of blank pixels at the top of the given text.

##### Parameters

<i>text</i>	A zero-terminated unicode string.
...	Variable arguments providing additional information.

##### Returns

The number of blank pixels above the text.

#### 7.69.3.15 getStringWidth() [1/2]

```
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

<i>text</i>	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.

#### 7.69.3.16 getStringWidth() [2/2]

```
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 width 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**

<i>textDirection</i>	The text direction.
<i>text</i>	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.

**7.69.3.17 getStringWidthLTR()**

```
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**

<i>textDirection</i>	The text direction.
<i>text</i>	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
<i>pArg</i>	Variable arguments providing additional information.

**Returns**

The width in pixels of the longest line of the specified string.

**7.69.3.18 getStringWidthRTL()**

```
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**

<i>textDirection</i>	The text direction.
<i>text</i>	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
<i>pArg</i>	The argument.

**Returns**

The string width RTL.

## 7.70 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.*

#### 7.70.1 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.

#### 7.70.2 Member Function Documentation

##### 7.70.2.1 getFont()

```
static Font * getFont (
    FontId fontId ) [static]
```

Gets a font.

##### Parameters

<i>fontId</i>	The font id of the font to get.
---------------	---------------------------------

**Returns**

The font with a font id of fontId.

##### 7.70.2.2 setFontProvider()

```
static void setFontProvider (
    FontProvider * fontProvider ) [static]
```

Sets the font provider. Must be initialized with the appropriate [FontProvider](#) by the application.

##### Parameters

in	<i>fontProvider</i>	Sets the font provider. Must be initialized with the appropriate <a href="#">FontProvider</a> by the application.
----	---------------------	-------------------------------------------------------------------------------------------------------------------

## 7.71 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.*

### 7.71.1 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.

### 7.71.2 Constructor & Destructor Documentation

#### 7.71.2.1 [~FontProvider](#)()

```
~FontProvider ( ) [inline], [virtual]
```

Destructor.

### 7.71.3 Member Function Documentation

#### 7.71.3.1 [getFont](#)()

```
Font * getFont (
    FontId fontId ) [pure virtual]
```

Gets a font.

#### Parameters

<i>fontId</i>	The font id of the font to get.
---------------	---------------------------------

#### Returns

The font with a font id of fontId.

## 7.72 CoverTransition< templateDirection >::FullSolidRect Class Reference

A [Widget](#) that returns a solid rect of the same size as the application.

```
#include <CoverTransition.hpp>
```

## Public Member Functions

- virtual [Rect](#) [getSolidRect](#) ( ) const  
*Pure virtual function for obtaining the largest possible rectangle that is guaranteed to be solid (non-transparent).*
- virtual void [draw](#) (const [Rect](#) &area) const  
*Pure virtual function for drawing this drawable.*

## Additional Inherited Members

### 7.72.1 Member Function Documentation

#### 7.72.1.1 [draw](#)()

```
virtual void draw (
    const Rect & invalidatedArea ) const [inline], [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

<i>invalidatedArea</i>	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).
------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Implements [Drawable](#).

#### 7.72.1.2 [getSolidRect](#)()

```
virtual Rect getSolidRect ( ) const [inline], [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](#).

## 7.73 [GenericCallback](#)< T1, T2, T3 > 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, T3 val3)=0  
*Calls the member function.*
- virtual bool [isValid](#) () const =0  
*Function to check whether the [Callback](#) has been initialized with values.*

### 7.73.1 Detailed Description

```
template<class T1 = void, class T2 = void, class T3 = void>
class touchgfx::GenericCallback< T1, T2, T3 >
```

[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 four versions to support callback functions taking zero, one, two or three arguments.

#### Template Parameters

<i>T1</i>	The type of the first argument in the member function, or void if none.
<i>T2</i>	The type of the second argument in the member function, or void if none.
<i>T3</i>	The type of the third argument in the member function, or void if none.

### 7.73.2 Constructor & Destructor Documentation

#### 7.73.2.1 ~GenericCallback()

```
~GenericCallback ( ) [inline], [virtual]
```

Empty virtual destructor.

### 7.73.3 Member Function Documentation

### 7.73.3.1 execute()

```
virtual void execute (
    T1 val1,
    T2 val2,
    T3 val3 ) [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

<i>val1</i>	This value will be passed as the first argument in the function call.
<i>val2</i>	This value will be passed as the second argument in the function call.
<i>val3</i>	This value will be passed as the third argument in the function call.

Implemented in [Callback< dest\\_type, T1, T2, T3 >](#), [Callback< touchgfx::SlideTransition, touchgfx::Drawable &>](#), [Callback< touchgfx::RadioButtonGroup, const touchgfx::AbstractButton &>](#), [Callback< touchgfx::CoverTransition, touchgfx::Drawable &>](#), [Callback< touchgfx::SlideMenu, const touchgfx::AbstractButton &>](#), and [Callback< touchgfx::SlideMenu, const touchgfx::MoveAnimator< touchgfx::Container > &>](#).

### 7.73.3.2 isValid()

```
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, void >](#), [Callback< dest\\_type, T1, void, void >](#), [Callback< dest\\_type, T1, T2, void >](#), [Callback< dest\\_type, T1, T2, T3 >](#), [Callback< touchgfx::SlideTransition, touchgfx::Drawable &>](#), [Callback< touchgfx::RadioButtonGroup, const touchgfx::AbstractButton &>](#), [Callback< touchgfx::CoverTransition, touchgfx::Drawable &>](#), [Callback< touchgfx::SlideMenu, const touchgfx::AbstractButton &>](#), and [Callback< touchgfx::SlideMenu, const touchgfx::MoveAnimator< touchgfx::Container > &>](#).

## 7.74 GenericCallback< T1, T2, 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, T2 val2)=0  
*Calls the member function.*
- virtual bool [isValid](#) () const =0  
*Function to check whether the [Callback](#) has been initialized with values.*

### 7.74.1 Detailed Description

```
template<class T1, class T2>
class touchgfx::GenericCallback< T1, T2, 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 four versions to support callback functions taking zero, one, two or three arguments.

#### Template Parameters

<i>T1</i>	The type of the first argument in the member function, or void if none.
<i>T2</i>	The type of the second argument in the member function, or void if none.

### 7.74.2 Constructor & Destructor Documentation

#### 7.74.2.1 ~GenericCallback()

```
~GenericCallback ( ) [inline], [virtual]
```

Empty virtual destructor.

### 7.74.3 Member Function Documentation

#### 7.74.3.1 execute()

```
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

<i>val1</i>	This value will be passed as the first argument in the function call.
<i>val2</i>	This value will be passed as the second argument in the function call.

### 7.74.3.2 isValid()

```
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).

## 7.75 GenericCallback< T1, void, 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.*

### 7.75.1 Detailed Description

```
template<class T1>
class touchgfx::GenericCallback< T1, void, 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 four versions to support callback functions taking zero, one, two or three arguments.

#### Template Parameters

<i>T1</i>	The type of the first argument in the member function, or void if none.
-----------	-------------------------------------------------------------------------



## 7.75.2 Constructor & Destructor Documentation

### 7.75.2.1 ~GenericCallback()

```
~GenericCallback ( ) [inline], [virtual]
```

Empty virtual destructor.

## 7.75.3 Member Function Documentation

### 7.75.3.1 execute()

```
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

<i>val1</i>	This value will be passed as the first argument in the function call.
-------------	-----------------------------------------------------------------------

### 7.75.3.2 isValid()

```
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).

## 7.76 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.*

### 7.76.1 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 four versions to support callback functions taking zero, one, two or three arguments.

### 7.76.2 Constructor & Destructor Documentation

#### 7.76.2.1 ~GenericCallback()

```
~GenericCallback ( ) [inline], [virtual]
```

Empty virtual destructor.

### 7.76.3 Member Function Documentation

#### 7.76.3.1 execute()

```
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).

#### 7.76.3.2 isValid()

```
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).

## 7.77 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.*

#### 7.77.1 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](#)

#### 7.77.2 Member Enumeration Documentation

##### 7.77.2.1 GestureType

```
enum enum GestureType
```

##### Enumerator

<a href="#">SWIPE_HORIZONTAL</a>	An enum constant representing a horizontal swipe.
<a href="#">SWIPE_VERTICAL</a>	An enum constant representing a vertical swipe.

### 7.77.3 Constructor & Destructor Documentation

#### 7.77.3.1 GestureEvent()

```
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

<i>t</i>	The type of the gesture event.
<i>v</i>	The velocity of this gesture (swipe)
<i>x_coord</i>	The x coordinate of the gesture.
<i>y_coord</i>	The y coordinate of the gesture.

### 7.77.4 Member Function Documentation

#### 7.77.4.1 getEventType()

```
Event::EventType getEventType ( ) [inline], [virtual]
```

##### Returns

The type of this event.

Implements [Event](#).

#### 7.77.4.2 getType()

```
GestureType getType ( ) const [inline]
```

Gets the type of this gesture event.

##### Returns

The type of this gesture event.

#### 7.77.4.3 getVelocity()

```
int16_t getVelocity ( ) const [inline]
```

Gets the velocity of this gesture event.

**Returns**

The velocity of this gesture event.

**7.77.4.4 getX()**

```
int16_t getX ( ) const [inline]
```

Gets the x coordinate of this gesture event.

**Returns**

The x coordinate of this gesture event.

**7.77.4.5 getY()**

```
int16_t getY ( ) const [inline]
```

Gets the y coordinate of this gesture event.

**Returns**

The y coordinate of this gesture event.

## 7.78 Gestures Class Reference

This class implements the detection of gestures.

```
#include <touchgfx/hal/Gestures.hpp>
```

### Public Member Functions

- [Gestures](#) ()  
*Default constructor.*
- void [registerEventListener](#) (UEventListener &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.*

### 7.78.1 Detailed Description

This class implements the detection of gestures.

## 7.78.2 Constructor & Destructor Documentation

### 7.78.2.1 Gestures()

```
Gestures ( ) [inline]
```

Default constructor. Does nothing.

## 7.78.3 Member Function Documentation

### 7.78.3.1 registerClickEvent()

```
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

<i>evt</i>	The type of the click event.
<i>x</i>	The x coordinate of the click event.
<i>y</i>	The y coordinate of the click event.

### 7.78.3.2 registerDragEvent()

```
bool registerDragEvent (
    uint16_t oldX,
    uint16_t oldY,
    uint16_t newX,
    uint16_t newY )
```

Register a drag event.

#### Parameters

<i>oldX</i>	The x coordinate of the drag start position (dragged from)
<i>oldY</i>	The y coordinate of the drag start position (dragged from)
<i>newX</i>	The x coordinate of the new position (dragged to)
<i>newY</i>	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).

## 7.78.3.3 registerEventListener()

```
void registerEventListener (
    UIEventListener & l )
```

Register the event listener.

## Parameters

in	/	The EventListener to register.
----	---	--------------------------------

## 7.78.3.4 setDragThreshold()

```
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.
-----	----------------------

## 7.78.3.5 tick()

```
void tick ( )
```

Has to be called during the timer tick.

## 7.79 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.*
- [int\\_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)*

### 7.79.1 Detailed Description

struct providing information about a glyph. Used by [LCD](#) when rendering.

### 7.79.2 Member Function Documentation

#### 7.79.2.1 advance()

```
uint16_t advance ( ) const [inline]
```

Gets the "advance" value where the 9th bit is stored in flags.

#### Returns

the right value of "advance".

#### 7.79.2.2 height()

```
uint16_t height ( ) const [inline]
```

Gets the "height" value where the 9th bit is stored in flags.

#### Returns

the right value of "height".



## 7.79.2.3 top()

```
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".

## 7.79.2.4 width()

```
uint16_t width ( ) const [inline]
```

Gets the "width" value where the 9th bit is stored in flags.

## Returns

the right value of "width".

## 7.80 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.*

## 7.80.1 Detailed Description

Interface class for manipulating GPIOs in order to do performance measurements on target. Not used on the PC simulator.

## 7.80.2 Member Enumeration Documentation

### 7.80.2.1 GPIO\_ID

```
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).

## 7.80.3 Member Function Documentation

### 7.80.3.1 clear()

```
static void clear (  
    GPIO_ID id ) [static]
```

Sets a pin low.

#### Parameters

<i>id</i>	the pin to set.
-----------	-----------------

### 7.80.3.2 get()

```
static bool get (  
    GPIO_ID id ) [static]
```

Gets the state of a pin.

#### Parameters

<i>id</i>	the pin to get.
-----------	-----------------

#### Returns

true if the pin is high, false otherwise.

### 7.80.3.3 init()

```
static void init ( ) [static]
```

Perform configuration of IO pins.

## 7.80.3.4 set()

```
static void set (
    GPIO_ID id ) [static]
```

Sets a pin high.

## Parameters

<i>id</i>	the pin to set.
-----------	-----------------

## 7.80.3.5 toggle()

```
static void toggle (
    GPIO_ID id ) [static]
```

Toggles a pin.

## Parameters

<i>id</i>	the pin to toggle.
-----------	--------------------

## 7.81 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.*

## 7.81.1 Constructor & Destructor Documentation

### 7.81.1.1 Gradients()

```
Gradients (
    const Point3D * vertices )
```

#### Parameters

<i>vertices</i>	The vertices.
-----------------	---------------

#### See also

[Point3D](#)

## 7.82 HAL Class Reference

Hardware Abstraction Layer.

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

### Public Types

- enum [FrameRefreshStrategy](#) { [REFRESH\\_STRATEGY\\_DEFAULT](#), [REFRESH\\_STRATEGY\\_OPTIM\\_SIN](#)↵  
[GLE\\_BUFFER\\_TFT\\_CTRL](#) }

*A list of available frame refresh strategies.*

### 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()`).*
- virtual uint16\_t \* `getTFTFrameBuffer ()` const =0

*Gets the frame buffer address used by the TFT controller.*
- 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.*
- virtual uint16\_t \* `copyFBRegionToMemory (Rect meAbs, uint16_t *dst, uint32_t stride)`

*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 routine 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.*

  - virtual void [setFrameBufferStartAddresses](#) (void \*frameBuffer, void \*doubleBuffer, void \*animationStorage)
- Sets frame buffer start addresses.*

  - 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.*

- bool [setFrameRefreshStrategy](#) (FrameRefreshStrategy s)

*Set a specific strategy for handling timing and mechanism of frame buffer drawing.*

- [FrameRefreshStrategy](#) [getFrameRefreshStrategy](#) () const

*Used internally by TouchGFX core to manage the timing and process of drawing into the frame buffer.*

- void [registerTaskDelayFunction](#) (void(\*delayF)(uint16\_t))

*Registers a function capable of delaying GUI task execution.*

- virtual void [taskDelay](#) (uint16\_t ms)

*Delay GUI task execution by number of milliseconds.*

- virtual uint16\_t [getTFTCurrentLine](#) ()

*Get the current line (Y) of the TFT controller.*

### 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 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.*
- void(\* [taskDelayFunc](#) )(uint16\_t)  
*Pointer to a function that can delay GUI task for a number of milliseconds.*
- 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.*
- [FrameRefreshStrategy](#) [refreshStrategy](#)  
*The selected display refresh strategy.*
- 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.*

## 7.82.1 Detailed Description

Contains functions that are specific to the hardware platform the code is running on.



## 7.82.2 Member Enumeration Documentation

### 7.82.2.1 FrameRefreshStrategy

```
enum typedef enum FrameRefreshStrategy
```

See also

```
bool setFrameRefreshStrategy(FrameRefreshStrategy s)
```

Enumerator

REFRESH_STRATEGY_DEFAULT	If not explicitly set, this strategy is used.
REFRESH_STRATEGY_OPTIM_SINGLE_BUFFER_TFT_CTRL	Strategy optimized for single frame buffer on systems with TFT controller.

## 7.82.3 Constructor & Destructor Documentation

### 7.82.3.1 HAL()

```
HAL (
    DMA_Interface & dmaInterface,
    LCD & display,
    TouchController & touchCtrl,
    uint16_t width,
    uint16_t height ) [inline]
```

Creates a [HAL](#) instance.

Parameters

in	<i>dmaInterface</i>	Reference to the DMA interface.
in	<i>display</i>	Reference to the <a href="#">LCD</a> .
in	<i>touchCtrl</i>	Reference to the touch controller.
	<i>width</i>	The width of the <a href="#">LCD</a> display, in pixels.
	<i>height</i>	The height of the <a href="#">LCD</a> display, in pixels.

### 7.82.3.2 ~HAL()

```
~HAL ( ) [inline], [virtual]
```

Destructor.

## 7.82.4 Member Function Documentation

#### 7.82.4.1 allowDMATransfers()

```
void allowDMATransfers ( ) [virtual]
```

Allow the DMA to start transfers. Front Porch Entry is a good place to call this.

#### 7.82.4.2 backPorchExited()

```
void backPorchExited ( ) [inline], [virtual]
```

Has to be called from within the [LCD](#) IRQ routine when the Back Porch Exit is reached.

#### 7.82.4.3 beginFrame()

```
bool beginFrame ( ) [protected], [virtual]
```

Called when beginning to rendering a frame.

##### Returns

true if rendering can begin, false otherwise.

#### 7.82.4.4 blitCopy()

```
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

<i>pSrc</i>	The source-array pointer (points to first value to copy)
<i>x</i>	The destination x coordinate on the frame buffer.
<i>y</i>	The destination y coordinate on the frame buffer.
<i>width</i>	The width desired area of the source 2D array.
<i>height</i>	The height of desired area of the source 2D array.
<i>srcWidth</i>	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

## 7.82.4.5 blitCopyARGB8888()

```
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

<i>pSrc</i>	The source-array pointer (points to first value to copy)
<i>x</i>	The destination x coordinate on the frame buffer.
<i>y</i>	The destination y coordinate on the frame buffer.
<i>width</i>	The width desired area of the source 2D array.
<i>height</i>	The height of desired area of the source 2D array.
<i>srcWidth</i>	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
<i>alpha</i>	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)

## 7.82.4.6 blitCopyGlyph()

```
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

<i>pSrc</i>	The source-array pointer (points to first value to copy)
<i>x</i>	The destination x coordinate on the frame buffer.
<i>y</i>	The destination y coordinate on the frame buffer.
<i>width</i>	The width desired area of the source 2D array.
<i>height</i>	The height of desired area of the source 2D array.
<i>srcWidth</i>	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
<i>color</i>	Color of the text.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>operation</i>	The operation type to use for blit copy.

#### 7.82.4.7 blitFill()

```
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

<i>color</i>	The desired fill-color.
<i>x</i>	The destination x coordinate on the frame buffer.
<i>y</i>	The destination y coordinate on the frame buffer.
<i>width</i>	The width desired area of the source 2D array.
<i>height</i>	The height of desired area of the source 2D array.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)

#### 7.82.4.8 blitSetTransparencyKey()

```
void blitSetTransparencyKey (
    uint16_t key ) [virtual]
```

Only present for backwards compatibility in TouchGFX 4.x. Will be removed in TouchGFX 5.

##### Parameters

<i>key</i>	The "transparent" color value.
------------	--------------------------------

Reimplemented in [HALSDL2](#).

#### 7.82.4.9 blockCopy()

```
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

<i>out</i>	<i>dest</i>	Pointer to destination memory.
------------	-------------	--------------------------------

**Parameters**

in	<i>src</i>	Pointer to source memory.
	<i>numBytes</i>	Number of bytes to copy.

**Returns**

true if the copy succeeded, false if copy was not performed.

Reimplemented in [HALSDL2](#).

**7.82.4.10 cacheTextString()**

```
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**

<i>str</i>	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).

**7.82.4.11 configureInterrupts()**

```
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 [HALSDL2](#).

**7.82.4.12 copyFBRegionToMemory()** [1/2]

```
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**

<i>meAbs</i>	The frame buffer region to copy.
--------------	----------------------------------

**Returns**

A pointer to the memory address containing the copy of the frame buffer.

**7.82.4.13 copyFBRegionToMemory()** [2/2]

```
uint16_t * copyFBRegionToMemory (
    Rect meAbs,
    uint16_t * dst,
    uint32_t stride ) [virtual]
```

Copies a region of the currently displayed frame buffer to a buffer. Used for e.g. [SlideTransition](#) and for displaying pre-rendered drawables e.g. in animations where redrawing the drawable is not necessary. The buffer can e.g. be a dynamic bitmap.

**Note**

Requires animation storage to be present.

**Parameters**

<i>meAbs</i>	The frame buffer region to copy.
<i>dst</i>	Address of the buffer to store the copy in.
<i>stride</i>	The width of the target buffer (row length).

**Returns**

A pointer to the memory address containing the copy of the frame buffer.

**7.82.4.14 disableInterrupts()**

```
void disableInterrupts ( ) [pure virtual]
```

Disables the DMA and [LCD](#) interrupts.

Implemented in [HALSDL2](#).

**7.82.4.15 enableInterrupts()**

```
void enableInterrupts ( ) [pure virtual]
```

Enables the DMA and [LCD](#) interrupts.

Implemented in [HALSDL2](#).

**7.82.4.16 enableLCDControllerInterrupt()**

```
void enableLCDControllerInterrupt ( ) [pure virtual]
```

Configure the [LCD](#) controller to fire interrupts at VSYNC. Called automatically once TouchGFX initialization has completed.

Implemented in [HALSDL2](#).

**7.82.4.17 enableMCULoadCalculation()**

```
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**

<i>enabled</i>	If true, set flag to update MCU load.
----------------	---------------------------------------

**7.82.4.18 endFrame()**

```
void endFrame ( ) [protected], [virtual]
```

Called when a rendering pass is completed.

**7.82.4.19 flushDMA()**

```
void flushDMA ( ) [virtual]
```

This function blocks until the DMA queue (containing BlitOps) is empty.

**7.82.4.20 flushFrameBuffer()** [1/2]

```
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 [HALSDL2](#).

**7.82.4.21 flushFrameBuffer()** [2/2]

```
void flushFrameBuffer (
    const Rect & rect ) [virtual]
```

This function is called whenever the framework has performed a partial draw.

**Parameters**

<i>rect</i>	The area of the screen that has been drawn, expressed in absolute coordinates.
-------------	--------------------------------------------------------------------------------

See also

[flushFrameBuffer\(\)](#).

Reimplemented in [HALSDL2](#).

#### 7.82.4.22 frontPorchEntered()

```
void frontPorchEntered ( ) [inline]
```

Has to be called from within the [LCD](#) IRQ routine when the Front Porch Entry is reached.

#### 7.82.4.23 getAnimationStorage()

```
uint16_t * getAnimationStorage ( ) const [inline]
```

Gets the optional frame buffer used for animation storage.

**Returns**

The address or 0 if unused.

#### 7.82.4.24 getBlitCaps()

```
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.

#### 7.82.4.25 getButtonController()

```
ButtonController * getButtonController ( ) const [inline]
```

Gets the associated [ButtonController](#).

**Returns**

A pointer to the [ButtonController](#), or zero if no [ButtonController](#) has been set.

#### 7.82.4.26 getClientFrameBuffer()

```
uint16_t * getClientFrameBuffer ( ) [inline], [protected]
```

Gets client frame buffer.

**Returns**

The address of the framebuffer currently used by the framework to draw in.



**7.82.4.27 getCPUCycles()**

```
uint32_t getCPUCycles ( )
```

Gets the current cycle counter.

**Returns**

the cycle counter.

**7.82.4.28 getDisplayHeight()**

```
uint16_t getDisplayHeight ( ) const [inline]
```

Gets display height.

**Returns**

The display height.

**7.82.4.29 getDisplayOrientation()**

```
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.

**7.82.4.30 getDisplayWidth()**

```
uint16_t getDisplayWidth ( ) const [inline]
```

Gets display width.

**Returns**

The display width.

**7.82.4.31 getFingerSize()**

```
uint8_t getFingerSize ( ) const [inline]
```

Gets the finger size in pixels.

**Returns**

The size of the finger in pixels, 1 is the default value.

#### 7.82.4.32 getFrameRefreshStrategy()

```
FrameRefreshStrategy getFrameRefreshStrategy ( ) const [inline]
```

##### Returns

Current frame refresh strategy.

##### See also

bool [setFrameRefreshStrategy](#)(FrameRefreshStrategy s)

#### 7.82.4.33 getInstance()

```
static HAL * getInstance ( ) [inline], [static]
```

Gets the [HAL](#) instance.

##### Returns

The [HAL](#) instance.

#### 7.82.4.34 getLCDRefreshCount()

```
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.

#### 7.82.4.35 getMCULoadPct()

```
uint8_t getMCULoadPct ( ) const [inline]
```

Gets the current MCU load.

##### Returns

mcuLoadPct the MCU Load in %.

#### 7.82.4.36 getTFTCurrentLine()

```
uint16_t getTFTCurrentLine ( ) [inline], [virtual]
```

This function is used to obtain the progress of the TFT controller. More specifically, the line (or Y-value) currently being transferred.

Note: The value must be adjusted to account for vertical back porch before returning, such that the value is always within the range of  $0 \leq \text{value} < \text{actual display height in pixels}$

It is used for the REFRESH\_STRATEGY\_OPTIM\_SINGLE\_BUFFER\_TFT\_CTRL frame refresh strategy in order to synchronize frame buffer drawing with TFT controller progress. If this strategy is used, the concrete [HAL](#) subclass must provide an override of this function that returns correct line value. If this strategy is not used, then the `getTFTCurrentLine` function is never called and can be disregarded.

#### Returns

In this default implementation, 0xFFFF is returned to signify "not implemented".

#### 7.82.4.37 `getTFTFrameBuffer()`

```
uint16_t * getTFTFrameBuffer ( ) const [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 [HALSDL2](#).

#### 7.82.4.38 `getTouchSampleRate()`

```
int8_t getTouchSampleRate ( ) const [inline]
```

Gets the number of ticks between each touch screen sample.

#### Returns

Number of ticks between each touch screen sample.

#### 7.82.4.39 `initialize()`

```
void initialize ( )
```

This function is responsible for initializing the entire framework.

#### 7.82.4.40 `lcd()`

```
static LCD & lcd ( ) [inline], [static]
```

Gets a reference to the [LCD](#).

#### Returns

A reference to the [LCD](#).

**7.82.4.41 lockDMAToFrontPorch()**

```
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**

<i>enableLock</i>	True to lock DMA transfers to the front porch signal. Conservative, default setting. False to disable, which will normally yield substantial performance improvement.
-------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

**7.82.4.42 lockFrameBuffer()**

```
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.

**7.82.4.43 noTouch()**

```
void noTouch ( ) [protected], [virtual]
```

Called by the touch driver to indicate that no touch is currently detected.

**7.82.4.44 performDisplayOrientationChange()**

```
void performDisplayOrientationChange ( ) [inline], [protected], [virtual]
```

Perform the actual display orientation change.

Reimplemented in [HALSDL2](#).

## 7.82.4.45 registerEventListener()

```
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	<i>listener</i>	The listener to register.
----	-----------------	---------------------------

## 7.82.4.46 registerTaskDelayFunction()

```
void registerTaskDelayFunction (
    void(*) (uint16_t) delayF ) [inline]
```

In order to make use of the [HAL::taskDelay](#) function, a delay function must be registered by calling this function. Usually the delay function would be [OSWrappers::taskDelay](#).

## Parameters

in, out	<i>delayF</i>	A pointer to a function returning void with an uint16_t parameter specifying number of milliseconds to delay.
---------	---------------	---------------------------------------------------------------------------------------------------------------

## Note

The task delay capability is only used when the frame refresh strategy REFRESH\_STRATEGY\_OPTIM\_SINGLE\_BUFFER\_TFT\_CTRL is selected. Otherwise it is not necessary to register a delay function.

## 7.82.4.47 registerTextCache()

```
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	<i>str</i>	Pointer to buffer location.
	<i>length</i>	Buffer length (in UnicodeChar's)

## See also

[cacheTextString](#)

#### 7.82.4.48 sampleKey()

```
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 [HALSDL2](#).

#### 7.82.4.49 setButtonController()

```
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.
----	---------	-------------------------------

#### 7.82.4.50 setDisplayOrientation()

```
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.
-------------	----------------------------------

#### 7.82.4.51 setDragThreshold()

```
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

<i>value</i>	New threshold value.
--------------	----------------------

#### 7.82.4.52 setFingerSize()

```
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  $2 \times (\text{fingersize} - 1) + 1$ . 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	<i>size</i>	the size of the finger.
----	-------------	-------------------------

#### 7.82.4.53 setFrameBufferStartAddress()

```
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	<i>adr</i>	Starting address to use for frame buffers.
	<i>depth</i>	Depth of each pixel in bits, default is 16.
	<i>useDoubleBuffering</i>	If true, reserve memory for an extra frame buffer.
	<i>useAnimationStorage</i>	If true, reserve memory for animation storage.

#### 7.82.4.54 setFrameBufferStartAddresses()

```
void setFrameBufferStartAddresses (
    void * frameBuffer,
    void * doubleBuffer,
    void * animationStorage ) [inline], [virtual]
```

Sets individual frame buffer start addresses.

##### Parameters

in	<i>frameBuffer</i>	Buffer for frame buffer data, must be non-null.
in	<i>doubleBuffer</i>	If non-null, buffer for double buffer data. If null double buffering is disabled.
in	<i>animationStorage</i>	If non-null, the animation storage. If null animation storage is disabled.

#### 7.82.4.55 setFrameRateCompensation()

```
void setFrameRateCompensation (
    bool enabled ) [inline]
```

Enables or disables compensation for lost frames. See knowledge base article.

##### Parameters

<i>enabled</i>	true to enable, false to disable.
----------------	-----------------------------------

#### 7.82.4.56 setFrameRefreshStrategy()

```
bool setFrameRefreshStrategy (
    FrameRefreshStrategy s ) [inline]
```

By setting a different frame refresh strategy, the internals of how TouchGFX interacts with the frame buffer can be modified.

Currently there are two strategies available. This will increase over time.

- **REFRESH\_STRATEGY\_OPTIM\_SINGLE\_BUFFER\_TFT\_CTRL**: this strategy is available on targets that use single buffering on a TFT controller based system. It requires an implementation of the [getTFTCurrentLine\(\)](#) function as well as a task delay function being registered. The implementation of this strategy is that TouchGFX will carefully track the progress of the TFT controller, and draw parts of the frame buffer whenever possible. The effect is that the risk of tearing is much reduced compared to the default single buffer strategy of only drawing in porch areas. It does have a drawback of slightly increased MCU load. But in many cases employing this strategy will make it possible to avoid external RAM, by using just a single frame buffer in internal RAM and still avoid tearing.
- **REFRESH\_STRATEGY\_DEFAULT**: This is a general strategy that works for all target configurations.

Recommendation: Try using **REFRESH\_STRATEGY\_OPTIM\_SINGLE\_BUFFER\_TFT\_CTRL** if you're on a TFT controller based system (ie. non-8080) and you have a desire to avoid external RAM. Otherwise stick to **REFRESH\_STRATEGY\_DEFAULT**.

##### Parameters

<i>s</i>	The desired strategy to use.
----------	------------------------------



**Returns**

true if the desired strategy will be used, false otherwise.

**7.82.4.57 setMCUActive()**

```
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**

<i>active</i>	If true, MCU is registered as being active, inactive otherwise.
---------------	-----------------------------------------------------------------

**7.82.4.58 setMCUInstrumentation()**

```
void setMCUInstrumentation (
    MCUInstrumentation * mcuInstr ) [inline]
```

Stores a pointer to an instance of an MCU specific instrumentation class.

**Parameters**

in	<i>mcuInstr</i>	pointer to MCU instrumentation.
----	-----------------	---------------------------------

**7.82.4.59 setTFTFrameBuffer()**

```
void setTFTFrameBuffer (
    uint16_t * address ) [protected], [pure virtual]
```

Sets the frame buffer address used by the TFT controller.

**Parameters**

in	<i>address</i>	New frame buffer address.
----	----------------	---------------------------

Implemented in [HALSDL2](#).

**7.82.4.60 setTouchSampleRate()**

```
void setTouchSampleRate (
    int8_t sampleRateInTicks ) [inline]
```

Sets the number of ticks between each touch screen sample.

**Parameters**

<i>sampleRateInTicks</i>	Sample rate. Default is 1 (every tick).
--------------------------	-----------------------------------------

**7.82.4.61 signalDMAInterrupt()**

```
void signalDMAInterrupt ( ) [inline]
```

Notify the framework that a DMA interrupt has occurred.

**7.82.4.62 swapFrameBuffers()**

```
void swapFrameBuffers ( )
```

Swaps the two frame buffers.

**7.82.4.63 taskDelay()**

```
void taskDelay (
    uint16_t ms ) [inline], [virtual]
```

This function requires the presence of a task delay function. If a task delay function has not been registered, it returns immediately. Otherwise it returns when number of milliseconds has passed.

**Parameters**

<i>ms</i>	Number of milliseconds to wait.
-----------	---------------------------------

**See also**

void [registerTaskDelayFunction](#)(void (\*delayF)(uint16\_t))

**7.82.4.64 taskEntry()**

```
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 [HALSDL2](#).

**7.82.4.65 tick()**

```
void tick ( ) [protected], [virtual]
```

This function is called at each timer tick, depending on platform implementation.

## 7.82.4.66 touch()

```
void touch (
    int32_t x,
    int32_t y ) [protected], [virtual]
```

Called by the touch driver to indicate a touch.

## Parameters

<i>x</i>	The x coordinate of the touch.
<i>y</i>	The y coordinate of the touch.

## 7.82.4.67 unlockFrameBuffer()

```
void unlockFrameBuffer ( ) [virtual]
```

Unlocks the frame buffer (MUST be called exactly once for each call to [lockFrameBuffer\(\)](#)).

## 7.82.4.68 vSync()

```
void vSync ( ) [inline]
```

Called by the VSync interrupt for counting of [LCD](#) refreshes.

## 7.83 HALSDL2 Class Reference

[HAL](#) implementation for the TouchGFX simulator.

```
#include <platform/hal/simulator/sdl/HALSDL2.hpp>
```

## Public Member Functions

- [HALSDL2](#) ([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.*
- void [setVsyncInterval](#) (float ms)  
*Sets vsync interval.*
- bool [doSampleTouch](#) (int32\_t &x, int32\_t &y) const  
*Samples the position of the mouse cursor.*

- virtual bool [sdl\\_init](#) (int argcount, char \*\*args)  
*Initializes SDL.*
- void [setWindowTitle](#) (const char \*title)  
*Sets window title.*
- const char \* [getWindowTitle](#) () const  
*Gets window title.*
- void [loadSkin](#) ([touchgfx::DisplayOrientation](#) orientation, int x, int y)  
*Loads a skin for a given display orientation.*
- void [saveScreenshot](#) ()  
*Saves a screenshot.*
- virtual void [saveNextScreenshots](#) (int n)  
*Copy the next N screenshots to disk.*
- virtual void [saveScreenshot](#) (char \*folder, char \*filename)  
*Saves a screenshot.*
- virtual void [copyScreenshotToClipboard](#) ()  
*Copies the screenshot to clipboard.*

### Static Public Member Functions

- static char \*\* [getArgv](#) (int \*argc)  
*Gets the argc and argv.*

### 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, uint8\_t \*frameBuffer)  
*Update frame buffer using an SDL Surface.*
- 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.*

### Additional Inherited Members

#### 7.83.1 Detailed Description

[HAL](#) implementation for the TouchGFX simulator.

See also

[HAL](#)

## 7.83.2 Constructor & Destructor Documentation

### 7.83.2.1 HALSDL2()

```
HALSDL2 (
    DMA_Interface & dma,
    LCD & lcd,
    TouchController & touchCtrl,
    uint16_t width,
    uint16_t height ) [inline]
```

Constructor. Initializes members.

#### Parameters

in	<i>dma</i>	Reference to DMA interface.
in	<i>lcd</i>	Reference to the <a href="#">LCD</a> .
in	<i>touchCtrl</i>	Reference to Touch Controller driver.
	<i>width</i>	Width of the display.
	<i>height</i>	Height of the display.

## 7.83.3 Member Function Documentation

### 7.83.3.1 blitSetTransparencyKey()

```
void blitSetTransparencyKey (
    uint16_t key ) [virtual]
```

If Blit-operations are supported, transparency-keying support is implicitly assumed.

#### Parameters

<i>key</i>	The "transparent" color value.
------------	--------------------------------

Reimplemented from [HAL](#).

### 7.83.3.2 blockCopy()

```
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	<i>dest</i>	Pointer to destination memory.
	<i>src</i>	Pointer to source memory.

**Parameters**

	<i>numBytes</i>	Number of bytes to copy.
--	-----------------	--------------------------

**Returns**

true if the copy succeeded, false if copy was not performed.

Reimplemented from [HAL](#).

**7.83.3.3 configureInterrupts()**

```
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](#).

**7.83.3.4 configureLCDInterrupt()**

```
void configureLCDInterrupt ( ) [inline], [protected], [virtual]
```

Configures [LCD](#) interrupt.

**7.83.3.5 copyScreenshotToClipboard()**

```
void copyScreenshotToClipboard ( ) [virtual]
```

Copies the screenshot to clipboard.

**7.83.3.6 disableInterrupts()**

```
void disableInterrupts ( ) [inline], [protected], [virtual]
```

Disables the DMA and [LCD](#) interrupts.

Implements [HAL](#).

**7.83.3.7 doSampleTouch()**

```
bool doSampleTouch (
    int32_t & x,
    int32_t & y ) const
```

Samples the position of the mouse cursor.

**Parameters**

out	<i>x</i>	The x coordinate.
out	<i>y</i>	The y coordinate.

**Returns**

True if touch detected, false otherwise.

**7.83.3.8 enableInterrupts()**

```
void enableInterrupts ( ) [inline], [protected], [virtual]
```

Enables the DMA and [LCD](#) interrupts.

Implements [HAL](#).

**7.83.3.9 enableLCDControllerInterrupt()**

```
void enableLCDControllerInterrupt ( ) [inline], [protected], [virtual]
```

Enables the [LCD](#) interrupt.

Implements [HAL](#).

**7.83.3.10 flushFrameBuffer()** [1/2]

```
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](#).

**7.83.3.11 flushFrameBuffer()** [2/2]

```
void flushFrameBuffer (
    const Rect & rect ) [virtual]
```

This function is called whenever the framework has performed a partial draw.

**Parameters**

<i>rect</i>	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](#).

**7.83.3.12 getArgv()**

```
static char ** getArgv (
    int * argc ) [static]
```

Gets the argc and argv for a Windows program.

#### Parameters

<i>argc</i>	Pointer to where to store number of arguments
-------------	-----------------------------------------------

#### Returns

The argv list of arguments.

#### 7.83.3.13 getTFTFrameBuffer()

```
uint16_t * getTFTFrameBuffer ( ) const [protected], [virtual]
```

Gets TFT frame buffer.

#### Returns

null if it fails, else the TFT frame buffer.

Implements [HAL](#).

#### 7.83.3.14 getWindowTitle()

```
const char * getWindowTitle ( ) const
```

Gets window title.

#### Returns

null "TouchGFX simulator" unless set to something else using [setWindowTitle\(\)](#).

#### See also

[setWindowTitle](#)

#### 7.83.3.15 loadSkin()

```
void loadSkin (
    touchgfx::DisplayOrientation orientation,
    int x,
    int y )
```

Loads a skin for a given display orientation that will be rendered in the simulator window with the the TouchGFX framebuffer placed inside the bitmap at the given coordinates. Different bitmaps can be loaded in landscape and portrait mode. If the provided bitmap cannot be loaded, the TouchGFX framebuffer will be displayed as normal. If the png files contain areas with alpha < 255, this will be used to create a shaped window.

#### Parameters

<i>orientation</i>	The orientation.
<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.



**Note**

The skins must be named "portrait.png" and "landscape.png" and placed inside the "simulator/" folder. The build process of the simulator will automatically copy the skins to the folder where the executable simulator is generated.

When a skin is set, the entire framebuffer is rendered through SDL whenever there is a change. Without a skin, only the areas with changes are rendered through SDL.

**7.83.3.16 performDisplayOrientationChange()**

```
void performDisplayOrientationChange ( ) [protected], [virtual]
```

Perform the actual display orientation change.

Reimplemented from [HAL](#).

**7.83.3.17 renderLCD\_FrameBufferToMemory()**

```
void renderLCD_FrameBufferToMemory (
    const Rect & _rectToUpdate,
    uint8_t * frameBuffer ) [protected], [virtual]
```

Update frame buffer using an SDL Surface.

**Parameters**

	<i>_rectToUpdate</i>	Area to update.
in	<i>frameBuffer</i>	Target frame buffer.

**7.83.3.18 sampleKey()**

```
bool sampleKey (
    uint8_t & key ) [virtual]
```

Sample key event from keyboard.

**Parameters**

out	<i>key</i>	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](#).

**7.83.3.19 saveNextScreenshots()**

```
void saveNextScreenshots (
    int n ) [virtual]
```

Copy the next N screenshots to disk. On each screen update, the new screen is saved to disk.

#### Parameters

<i>n</i>	Number of screenshots to save. These are added to any ongoing amount of screenshots in queue.
----------	-----------------------------------------------------------------------------------------------

#### 7.83.3.20 saveScreenshot() [1/2]

```
void saveScreenshot ( )
```

Saves a screenshot to the default folder and default filename.

#### 7.83.3.21 saveScreenshot() [2/2]

```
void saveScreenshot (
    char * folder,
    char * filename ) [virtual]
```

Saves a screenshot.

#### Parameters

in	<i>folder</i>	Folder name to place the screenshot in.
in	<i>filename</i>	Filename to save the screenshot to.

#### 7.83.3.22 sdl\_init()

```
bool sdl_init (
    int argcount,
    char ** args ) [virtual]
```

Initializes SDL.

#### Parameters

	<i>argcount</i>	Number of arguments.
in	<i>args</i>	Arguments.

#### Returns

True if init went well, false otherwise.

#### 7.83.3.23 setTFTFrameBuffer()

```
void setTFTFrameBuffer (
    uint16_t * addr ) [protected], [virtual]
```

Sets TFT frame buffer.

## Parameters

<i>in</i>	<i>addr</i>	The address of the TFT frame buffer.
-----------	-------------	--------------------------------------

Implements [HAL](#).

## 7.83.3.24 setVsyncInterval()

```
void setVsyncInterval (
    float ms )
```

Sets vsync interval for simulating same tick speed as the real hardware. Due to limitations in the granularity of SDL, the generated ticks in the simulator might not occur at the exact time, but accumulated over several ticks, the precision is very good.

## Parameters

<i>ms</i>	The milliseconds between ticks.
-----------	---------------------------------

## Note

That you can also use [HAL::setFrameRateCompensation\(\)](#) in the simulator. The effect of this can easily be demonstrated by dragging the console output window of the simulator (when running from Visual Studio) as this will pause the SDL and generate a lot of ticks when the console window is released. Beware that since the missed vsyncs are accumulated in an 8 bit counter, only up to 255 ticks may be missed, so at VsyncInterval = 16.6667, dragging the windows for more than  $255 * 16.6667\text{ms} = 4250\text{ms} = 4.25\text{s}$  will not generate all the ticks that were actually missed. This situation is, however, not very realistic, as normally just a couple of vsyncs are skipped.

## 7.83.3.25 setWindowTitle()

```
void setWindowTitle (
    const char * title )
```

Sets window title of the TouchGFX simulator.

## Parameters

<i>title</i>	The title, if null the original "TouchGFX simulator" will be used.
--------------	--------------------------------------------------------------------

## See also

[getTitle](#)

## 7.83.3.26 taskEntry()

```
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](#).

## 7.84 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.*

#### 7.84.1 Detailed Description

Platform independent interface for [I2C](#) drivers.

#### 7.84.2 Constructor & Destructor Documentation

##### 7.84.2.1 I2C()

```
I2C (
    uint8_t ch ) [inline]
```

Stores the channel of the [I2C](#) bus to be configured.

#### Parameters

<i>ch</i>	<a href="#">I2C</a> channel.
-----------	------------------------------

### 7.84.2.2 ~I2C()

```
~I2C ( ) [inline], [virtual]
```

Destructor.

## 7.84.3 Member Function Documentation

### 7.84.3.1 init()

```
void init ( ) [pure virtual]
```

Initializes the I2C driver.

### 7.84.3.2 readRegister()

```
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

	<i>addr</i>	The I2C device address.
	<i>reg</i>	The register.
out	<i>data</i>	Pointer to buffer in which to place the result.
	<i>cnt</i>	Size of buffer in bytes.

#### Returns

true on success, false otherwise.

### 7.84.3.3 writeRegister()

```
bool writeRegister (
    uint8_t addr,
    uint8_t reg,
    uint8_t val ) [pure virtual]
```

Writes the specified value in a register.

#### Parameters

<i>addr</i>	The I2C device address.
<i>reg</i>	The register.
<i>val</i>	The new value.

**Returns**

true on success, false otherwise.

## 7.85 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.*

### 7.85.1 Detailed Description

Specific I2C-enabled type of Touch Controller.

See also

[TouchController](#)

### 7.85.2 Constructor & Destructor Documentation

#### 7.85.2.1 I2CTouchController()

```
I2CTouchController (  
    I2C & i2c ) [inline]
```

Constructor. Initializes [I2C](#) driver.

**Parameters**

<a href="#">in</a> , <a href="#">out</a>	<a href="#">i2c</a>	<a href="#">I2C</a> driver.
------------------------------------------	---------------------	-----------------------------

## 7.85.2.2 ~I2CTouchController()

```
~I2CTouchController ( ) [inline], [virtual]
```

Destructor.

## 7.85.3 Member Function Documentation

## 7.85.3.1 init()

```
void init ( ) [pure virtual]
```

Initializes touch controller.

Implements [TouchController](#).

## 7.85.3.2 sampleTouch()

```
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	<i>x</i>	The x position of the touch
out	<i>y</i>	The y position of the touch

## Returns

True if a touch has been detected, otherwise false.

Implements [TouchController](#).

## 7.86 IconButtonStyle&lt; T &gt; Class Template Reference

An icon button style.

```
#include <touchgfx/containers/buttons/IconButtonStyle.hpp>
```

## Public Member Functions

- [IconButtonStyle](#) ()  
*Default constructor.*
- virtual [~IconButtonStyle](#) ()  
*Destructor.*
- virtual void [setIconBitmaps](#) (const [Bitmap](#) &newIconReleased, const [Bitmap](#) &newIconPressed)  
*Sets icon bitmaps.*
- void [setIconX](#) (int16\_t x)

- Sets icon x coordinate.*
- void [setIconY](#) (int16\_t y)  
*Sets icon y coordinate.*
- void [setIconXY](#) (int16\_t x, int16\_t y)  
*Sets icon xy.*
- [Bitmap getCurrentlyDisplayedIcon](#) () const  
*Gets currently displayed icon.*
- int16\_t [getIconX](#) () const  
*Gets icon x coordinate.*
- int16\_t [getIconY](#) () const  
*Gets icon y coordinate.*

### Protected Member Functions

- virtual void [handlePressedUpdated](#) ()  
*Handles the pressed updated.*
- virtual void [handleAlphaUpdated](#) ()  
*Handles the alpha updated.*

### Protected Attributes

- [Bitmap iconReleased](#)  
*Icon to display when button is not pressed.*
- [Bitmap iconPressed](#)  
*Icon to display when button is pressed.*
- [Image iconImage](#)  
*The icon image.*

### 7.86.1 Detailed Description

```
template<class T>
class touchgfx::IconButtonStyle< T >
```

An icon button style. This class is supposed to be used with one of the ButtonTrigger classes to create a functional button. This class will show one of two icons depending on the state of the button (pressed or released).

To get a background behind the icon, use [IconButtonStyle](#) together with e.g. [ImageButtonStyle](#): `IconButtonStyle<ImageButtonStyle<ClickButtonTrigger> > myButton;`

The [IconButtonStyle](#) will center the icon on the enclosing container (normally [AbstractButtonContainer](#)). Set the size of the button before setting the icons.

The position of the icon can be adjusted with [setIconXY](#).

#### Template Parameters

<i>T</i>	Generic type parameter. Typically a <a href="#">AbstractButtonContainer</a> subclass.
----------	---------------------------------------------------------------------------------------



See also

[AbstractButtonContainer](#)

## 7.86.2 Member Function Documentation

### 7.86.2.1 getCurrentlyDisplayedIcon()

```
Bitmap getCurrentlyDisplayedIcon ( ) const [inline]
```

#### Returns

The currently displayed icon.

### 7.86.2.2 getIconX()

```
int16_t getIconX ( ) const [inline]
```

#### Returns

The icon x coordinate.

### 7.86.2.3 getIconY()

```
int16_t getIconY ( ) const [inline]
```

#### Returns

The icon y coordinate.

### 7.86.2.4 setIconBitmaps()

```
void setIconBitmaps (
    const Bitmap & newIconReleased,
    const Bitmap & newIconPressed ) [inline], [virtual]
```

#### Parameters

<i>newIconReleased</i>	The new icon released.
<i>newIconPressed</i>	The new icon pressed.

### 7.86.2.5 setIconX()

```
void setIconX (
```

```
int16_t x ) [inline]
```

#### Parameters

<i>x</i>	The x coordinate.
----------	-------------------

#### 7.86.2.6 setIconXY()

```
void setIconXY (
    int16_t x,
    int16_t y ) [inline]
```

#### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

#### 7.86.2.7 setIconY()

```
void setIconY (
    int16_t y ) [inline]
```

#### Parameters

<i>y</i>	The y coordinate.
----------	-------------------

## 7.87 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.*
- virtual 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

### 7.87.1 Detailed Description

Simple widget capable of showing a bitmap. The bitmap can be alpha-blended with the background and have areas of transparency.

See also

[Widget](#)

### 7.87.2 Constructor & Destructor Documentation

#### 7.87.2.1 Image()

```
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

<i>bmp</i>	The bitmap to display.
------------	------------------------

### 7.87.3 Member Function Documentation

#### 7.87.3.1 draw()

```
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**

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

Implements [Drawable](#).

Reimplemented in [TiledImage](#).

**7.87.3.2 getAlpha()**

```
uint8_t getAlpha ( ) const [inline]
```

Gets the current alpha value.

**Returns**

The current alpha value.

**7.87.3.3 getBitmap()**

```
BitmapId getBitmap ( ) const [inline]
```

Gets the BitmapId currently contained by the widget.

**Returns**

The current BitmapId of the widget.

**7.87.3.4 getSolidRect()**

```
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](#).

Reimplemented in [TiledImage](#).

**7.87.3.5 getType()**

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

## Returns

TYPE\_IMAGE.

Reimplemented from [Widget](#).

Reimplemented in [AnimatedImage](#), and [TiledImage](#).

## 7.87.3.6 setAlpha()

```
void setAlpha (
    uint8_t alpha ) [inline]
```

Sets the alpha channel for the image.

## Parameters

<i>alpha</i>	The alpha value. 255 = completely solid.
--------------	------------------------------------------

## 7.87.3.7 setBitmap()

```
void setBitmap (
    const Bitmap & bmp ) [virtual]
```

Sets the bitmap ID for this [Image](#). Updates the width and height of this widget to match that of the bitmap.

## Parameters

<i>bmp</i>	The bitmap instance.
------------	----------------------

## See also

[Bitmap](#)

Reimplemented in [AnimatedImage](#), and [TiledImage](#).

## 7.88 ImageButtonStyle&lt; T &gt; Class Template Reference

An image button style.

```
#include <touchgfx/containers/buttons/ImageButtonStyle.hpp>
```

## Public Member Functions

- [ImageButtonStyle](#) ()  
*Default constructor.*
- virtual [~ImageButtonStyle](#) ()  
*Destructor.*
- virtual void [setBitmaps](#) (const [Bitmap](#) &bmpReleased, const [Bitmap](#) &bmpPressed)  
*Sets the bitmaps.*
- void [setBitmapXY](#) (uint16\_t x, uint16\_t y)

*Sets bitmap xy.*

- [Bitmap getCurrentlyDisplayedBitmap \(\)](#) const

*Gets currently displayed bitmap.*

## Protected Member Functions

- virtual void [handlePressedUpdated \(\)](#)

*Handles the pressed updated.*

- virtual void [handleAlphaUpdated \(\)](#)

*Handles the alpha updated.*

## Protected Attributes

- [Image buttonImage](#)

*The button image.*

- [Bitmap up](#)

*The image to display when button is released.*

- [Bitmap down](#)

*The image to display when button is pressed.*

## 7.88.1 Detailed Description

```
template<class T>
```

```
class touchgfx::ImageButtonStyle< T >
```

An image button style. This class is supposed to be used with one of the [ButtonTrigger](#) classes to create a functional button. This class will show one of two images depending on the state of the button (pressed or released).

The [ImageButtonStyle](#) will set the size of the enclosing container (normally [AbstractButtonContainer](#)) to the size of the pressed [Bitmap](#). This can be overridden by calling `setWidth/setHeight` after setting the bitmaps.

The position of the bitmap can be adjusted with `setBitmapXY` (default is upper left corner).

### Template Parameters

<a href="#">T</a>	Generic type parameter. Typically a <a href="#">AbstractButtonContainer</a> subclass.
-------------------	---------------------------------------------------------------------------------------

### See also

[AbstractButtonContainer](#)

## 7.88.2 Member Function Documentation

### 7.88.2.1 getCurrentlyDisplayedBitmap()

```
Bitmap getCurrentlyDisplayedBitmap ( ) const [inline]
```

### Returns

The currently displayed bitmap.

## 7.88.2.2 setBitmaps()

```
void setBitmaps (
    const Bitmap & bmpReleased,
    const Bitmap & bmpPressed ) [inline], [virtual]
```

## Parameters

<i>bmpReleased</i>	The bitmap released.
<i>bmpPressed</i>	The bitmap pressed.

## 7.88.2.3 setBitmapXY()

```
void setBitmapXY (
    uint16_t x,
    uint16_t y ) [inline]
```

## Parameters

<i>x</i>	An uint16_t to process.
<i>y</i>	An uint16_t to process.

## 7.89 ImageProgress Class Reference

An image progress.

```
#include <ImageProgress.hpp>
```

## Public Member Functions

- [ImageProgress](#) ()  
*Default constructor.*
- virtual [~ImageProgress](#) ()  
*Destructor.*
- virtual void [setProgressIndicatorPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets the position and dimension of the image progress indicator.*
- virtual void [setAnchorAtZero](#) (bool anchorAtZero)  
*Sets anchor at zero.*
- virtual bool [getAnchorAtZero](#) () const  
*Gets anchor at zero.*
- virtual void [setBitmap](#) ([touchgfx::BitmapId](#) bitmapId)  
*Sets the bitmap id.*
- virtual [touchgfx::BitmapId](#) [getBitmap](#) () const  
*Gets the image.*
- virtual void [setAlpha](#) (uint8\_t alpha)  
*Sets the alpha.*
- virtual uint8\_t [getAlpha](#) () const  
*Gets the alpha.*
- virtual void [setValue](#) (int value)  
*Sets a value.*

## Protected Attributes

- [Container container](#)  
*The container.*
- [TiledImage image](#)  
*The image.*
- `bool` [fixedPosition](#)  
*true if the image should not move during progress*

## Additional Inherited Members

### 7.89.1 Detailed Description

touchgfx/containers/progress\_indicators/ImageProgress.hpp

An image progress will show parts of an image as a progress indicator. The image can progress from the left, the right, the bottom or the top of the given area, and can visually be fixed with a larger and larger portion of the image showing, or it can be moved into view.

### 7.89.2 Constructor & Destructor Documentation

#### 7.89.2.1 ImageProgress()

```
ImageProgress ( )
```

Default constructor.

#### 7.89.2.2 ~ImageProgress()

```
~ImageProgress ( ) [virtual]
```

Destructor.

### 7.89.3 Member Function Documentation

#### 7.89.3.1 getAlpha()

```
uint8_t getAlpha ( ) const [virtual]
```

#### Returns

The the alpha of the image.

#### See also

[setAlpha](#)  
[Image::getAlpha](#)



### 7.89.3.2 getAnchorAtZero()

```
bool getAnchorAtZero ( ) const [virtual]
```

Gets anchor at zero.

#### Returns

true if the image is anchored at zero, false if it is anchored at current progress.

#### See also

[setAnchorAtZero](#)

### 7.89.3.3 getBitmap()

```
touchgfx::BitmapId getBitmap ( ) const [virtual]
```

Gets the image.

#### Returns

The image.

#### See also

[setBitmap](#)

### 7.89.3.4 setAlpha()

```
void setAlpha (
    uint8_t alpha ) [virtual]
```

Sets the alpha of the image.

#### Parameters

<i>alpha</i>	The alpha.
--------------	------------

#### See also

[getAlpha](#)  
[Image::setAlpha](#)

### 7.89.3.5 setAnchorAtZero()

```
void setAnchorAtZero (
    bool anchorAtZero ) [virtual]
```

Sets anchor at zero will control whether the image will be placed so that it is not moving during progress, only more and more of the image will become visible, or if the image is anchored at the current progress and will appear to

slide into view.

#### Parameters

<i>anchorAtZero</i>	true to anchor at zero, false to anchor at current progress.
---------------------	--------------------------------------------------------------

#### See also

[getAnchoredAtZero](#)

#### 7.89.3.6 setBitmap()

```
void setBitmap (
    touchgfx::BitmapId bitmapId ) [virtual]
```

Sets the bitmap id to use for progress. Please note that the bitmap is tiled which will allow smaller bitmaps to repeat and save memory.

#### Parameters

<i>bitmapId</i>	The bitmap id.
-----------------	----------------

#### See also

[getBitmap](#)  
[TiledImage](#)

#### 7.89.3.7 setProgressIndicatorPosition()

```
void setProgressIndicatorPosition (
    int16_t x,
    int16_t y,
    int16_t width,
    int16_t height ) [virtual]
```

Sets the position and dimension of the image progress indicator relative to the background image.

#### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>width</i>	The width of the image progress indicator.
<i>height</i>	The height of the image progress indicator.

Reimplemented from [AbstractProgressIndicator](#).

## 7.89.3.8 setValue()

```
virtual void setValue (
    int value ) [virtual]
```

Sets the current value in the range (min..max) set by [setRange\(\)](#). Values lower than min are mapped to min, values higher than max are mapped to max.

## Parameters

<i>value</i>	The value.
--------------	------------

Reimplemented from [AbstractProgressIndicator](#).

## 7.90 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, const [Unicode::UnicodeChar](#) fallbackChar, const [Unicode::UnicodeChar](#) ellipsisChar)  
*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

## 7.90.1 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](#)

## 7.90.2 Constructor &amp; Destructor Documentation

## 7.90.2.1 InternalFlashFont()

```
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,
const Unicode::UnicodeChar fallbackChar,
const Unicode::UnicodeChar ellipsisChar )

```

Construct the [InternalFlashFont](#).

#### Parameters

<i>list</i>	The array of glyphs known to this font.
<i>size</i>	The number of glyphs in list.
<i>height</i>	The height in pixels of the highest character in this font.
<i>pixBelowBase</i>	The maximum number of pixels that can be drawn below the baseline in this font.
<i>bitsPerPixel</i>	The number of bits per pixel in this font.
<i>maxLeft</i>	The maximum a character extends to the left.
<i>maxRight</i>	The maximum a character extends to the right.
<i>glyphDataInternalFlash</i>	Pointer to the glyph data for the font, placed in internal flash.
<i>kerningList</i>	pointer to the kerning data for the font, placed in internal flash.
<i>fallbackChar</i>	The fallback character for the typography in case no glyph is available.
<i>ellipsisChar</i>	The ellipsis character used for truncating long texts.

## 7.90.3 Member Function Documentation

### 7.90.3.1 getKerning()

```

int8_t getKerning (
    Unicode::UnicodeChar prevChar,
    const GlyphNode * glyph ) const [virtual]

```

Gets the kerning distance between two characters.

#### Parameters

<i>prevChar</i>	The unicode value of the previous character.
<i>glyph</i>	the glyph object for the current character.

#### Returns

The kerning distance between prevChar and glyph char.

Implements [ConstFont](#).

### 7.90.3.2 getPixelData()

```

const uint8_t * getPixelData (
    const GlyphNode * glyph ) const [virtual]

```

Obtains a RAM-based pointer to the pixel data for the specified glyph.

#### Parameters

<code>glyph</code>	The glyph to get the pixels data of.
--------------------	--------------------------------------

#### Returns

The pixel data of the glyph.

Implements [ConstFont](#).

## 7.91 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](#).*

### 7.91.1 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.

### 7.91.2 Constructor & Destructor Documentation

#### 7.91.2.1 iterator()

```
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

<code>scanline</code>	The scanline to iterate.
-----------------------	--------------------------

### 7.91.3 Member Function Documentation

#### 7.91.3.1 getCovers()

```
const uint8_t * getCovers ( ) const [inline]
```

Gets the covers in the current run on the [Scanline](#).

##### Returns

array of covers of each individual pixel.

#### 7.91.3.2 getNumPix()

```
int getNumPix ( ) const [inline]
```

Gets number of consecutive pixels in the current run on the [Scanline](#).

##### Returns

The number of consecutive pixels.

#### 7.91.3.3 next()

```
int next ( ) [inline]
```

Gets the next element on the [Scanline](#).

##### Returns

An the next index in the array of [Scanline](#) elements.

## 7.92 JSMOCHelper Class Reference

Helper class providing caching of certain information while the JSMOCH 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.*

### 7.92.1 Detailed Description

Helper class providing caching of certain information while the JSMOCHelper algorithm runs during draw operations. Not intended for application-level use.

### 7.92.2 Constructor & Destructor Documentation

#### 7.92.2.1 JSMOCHelper()

```
JSMOCHelper ( ) [inline]
```

Default constructor.

### 7.92.3 Member Function Documentation

#### 7.92.3.1 draw()

```
void draw (
    const Rect & invalidatedArea ) [inline]
```

Draws the widget of this helper.

##### Parameters

<i>invalidatedArea</i>	The area of the widget to draw.
------------------------	---------------------------------

#### 7.92.3.2 getCachedAbsX()

```
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.

#### 7.92.3.3 `getCachedAbsY()`

```
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.

#### 7.92.3.4 `getCachedVisibleRect()`

```
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.

#### 7.92.3.5 `getHeight()`

```
int16_t getHeight ( ) [inline]
```

Gets the height of the widget of this helper.

##### Returns

The height of the widget of this helper.

#### 7.92.3.6 `getWidget()`

```
Drawable * getWidget ( ) [inline]
```

Gets the widget.

##### Returns

The widget this helper operates on.

#### 7.92.3.7 `getWidth()`

```
int16_t getWidth ( ) [inline]
```

Gets the width of the widget of this helper.

##### Returns

The width of the widget of this helper.



## 7.92.3.8 setWidget()

```
void setWidget (
    Drawable * newWidget ) [inline]
```

Sets a widget.

## Parameters

in	<i>newWidget</i>	The widget to operate on.
----	------------------	---------------------------

## 7.93 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.*

## 7.93.1 Detailed Description

Structure providing information about a kerning for a given char pair. Used by [LCD](#) when rendering.

## 7.94 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".*

## 7.95 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.*
- virtual void [handleDragEvent](#) (const [DragEvent](#) &evt)  
*Overrides the handleDragEvent 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) const  
*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) const  
*Gets the callback area defined by the layout for the specified coordinates.*
- virtual void [setupDrawChain](#) (const [Rect](#) &invalidatedArea, [Drawable](#) \*\*nextPreviousElement)  
*Add to draw chain.*

## 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.*
- bool [cancelsEmitted](#)  
*Tells if a cancel is emitted to check when a key is released.*

## Additional Inherited Members

### 7.95.1 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](#)

## 7.95.2 Constructor & Destructor Documentation

### 7.95.2.1 Keyboard()

`Keyboard ( )`

Creates a new [Keyboard](#).

### 7.95.2.2 ~Keyboard()

`~Keyboard ( ) [inline], [virtual]`

Destructor.

## 7.95.3 Member Function Documentation

### 7.95.3.1 draw()

```
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

<i>invalidatedArea</i>	The area to draw.
------------------------	-------------------

Reimplemented from [Container](#).

### 7.95.3.2 getBuffer()

`Unicode::UnicodeChar * getBuffer ( ) const [inline]`

Gets the buffer.

#### Returns

The buffer containing entered text currently being displayed.

### 7.95.3.3 getBufferPosition()

`uint16_t getBufferPosition ( ) [inline]`

Gets buffer position.

**Returns**

the buffer position, i.e. the current index where new characters will be placed.

**7.95.3.4 getCallbackAreaForCoordinates()**

```
CallbackArea getCallbackAreaForCoordinates (
    int16_t x,
    int16_t y ) const [protected]
```

Gets the callback area defined by the layout for the specified coordinates.

**Parameters**

<i>x</i>	The x coordinate to perform key look up with.
<i>y</i>	The y coordinate to perform key look up with.

**Returns**

The [CallbackArea](#), which is empty if not found.

**7.95.3.5 getCharForKey()**

```
Unicode::UnicodeChar getCharForKey (
    uint8_t keyId ) const [protected]
```

Maps a keyId to the UnicodeChar being displayed by that key.

**Parameters**

<i>keyId</i>	The id of the key to perform lookup with.
--------------	-------------------------------------------

**Returns**

the UnicodeChar used for the specified key.

**7.95.3.6 getKeyForCoordinates()**

```
Key getKeyForCoordinates (
    int16_t x,
    int16_t y ) const [protected]
```

Gets key for coordinates.

**Parameters**

<i>x</i>	The x coordinate to perform key look up with.
<i>y</i>	The y coordinate to perform key look up with.

**Returns**

The key for the given coordinates.

**7.95.3.7 getKeyMappingList()**

```
const KeyMappingList * getKeyMappingList ( ) const [inline]
```

Gets key mapping list.

**Returns**

The [KeyMappingList](#) used by the [Keyboard](#).

**7.95.3.8 getLayout()**

```
const Layout * getLayout ( ) const [inline]
```

Gets the layout.

**Returns**

The layout used by the [Keyboard](#).

**7.95.3.9 getType()**

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

**Returns**

TYPE\_KEYBOARD.

Reimplemented from [Container](#).

**7.95.3.10 handleClickEvent()**

```
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**

<i>evt</i>	The <a href="#">ClickEvent</a> .
------------	----------------------------------

Reimplemented from [Drawable](#).

7.95.3.11 `handleDragEvent()`

```
void handleDragEvent (
    const DragEvent & evt ) [virtual]
```

Overrides the `handleDragEvent` on the container. The keyboard handles drag events to enable the container to, emit a CANCEL, if the user drags outside the currently pressed key.

## Parameters

<i>evt</i>	The <a href="#">DragEvent</a> .
------------	---------------------------------

Reimplemented from [Drawable](#).

7.95.3.12 `setBuffer()`

```
void setBuffer (
    Unicode::UnicodeChar * newBuffer,
    uint16_t newBufferSize )
```

Sets the buffer to be used by the keyboard.

## Parameters

in	<i>newBuffer</i>	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.
	<i>newBufferSize</i>	Length of the buffer, i.e. number of <a href="#">UnicodeChar</a> 's.

7.95.3.13 `setBufferPosition()`

```
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

<i>newPos</i>	The buffer position.
---------------	----------------------

7.95.3.14 `setKeyListener()`

```
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 <a href="#">Callback</a> to invoke.
----	----------	-----------------------------------------

#### 7.95.3.15 setKeymappingList()

```
void setKeymappingList (
    const KeyMappingList * newKeyMappingList )
```

Set/change the [KeyMappingList](#) to use. The [Keyboard](#) will invalidate the space it occupies to request a redraw.

#### Parameters

<i>newKeyMappingList</i>	The new <a href="#">KeyMappingList</a> .
--------------------------	------------------------------------------

#### 7.95.3.16 setLayout()

```
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

<i>newLayout</i>	The new layout.
------------------	-----------------

#### 7.95.3.17 setTextIndentation()

```
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.

#### 7.95.3.18 setupDrawChain()

```
void setupDrawChain (
    const Rect & invalidatedArea,
    Drawable ** nextPreviousElement ) [protected], [virtual]
```



**Note**

For TouchGFX internal use only.

**Parameters**

	<i>invalidatedArea</i>	Include drawables that intersect with this area only.
<i>in, out</i>	<i>nextPreviousElement</i>	Modifiable element in linked list.

Reimplemented from [Container](#).

## 7.96 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 keyVal`  
*Unicode equivalent of the key id.*

## 7.97 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.*

## 7.98 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.*

## 7.99 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 frame buffer while converting the format.*
- virtual uint16\_t \* `copyFrameBufferRegionToMemory` (const `Rect` &region, const `BitmapId` bitmap=BITMAP↔\_ANIMATION\_STORAGE)=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 with 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 offsetY, 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` (const `Rect` &widgetArea, const `Rect` &invalidatedArea, const `StringVisuals` &visuals, const `Unicode::UnicodeChar` \*format, va\_list pArg)

*Draws the specified unicode string.*

- void `drawStringRTL` (const `Rect` &widgetArea, const `Rect` &invalidatedArea, const `StringVisuals` &visuals, 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` (const `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` (const `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 uint16\_t [getNumLines](#) ([TextProvider](#) &textProvider, [WideTextAction](#) wideTextAction, [TextDirection](#) textDirection, const [Font](#) \*font, int16\_t width)

*Gets number lines.*

## Static Protected Attributes

- static const uint16\_t [newLine](#) = 10

*NewLine value.*

### 7.99.1 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!

### 7.99.2 Constructor & Destructor Documentation

#### 7.99.2.1 ~LCD()

```
~LCD ( ) [inline], [virtual]
```

Destructor.

### 7.99.3 Member Function Documentation

#### 7.99.3.1 bitDepth()

```
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](#), [LCD2bpp](#), [LCD4bpp](#), [LCD16bpp](#), and [LCD1bpp](#).

## 7.99.3.2 blitCopy() [1/2]

```
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 performing alpha-blending (and transparency keying) as specified. Performs a software blend if [HAL](#) does not support BLIT\_COPY\_WITH\_ALPHA and alpha != 255.

## Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implemented in [LCD24bpp](#), [LCD2bpp](#), [LCD4bpp](#), [LCD16bpp](#), and [LCD1bpp](#).

## 7.99.3.3 blitCopy() [2/2]

```
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 frame buffer performing alpha-blending (and transparency keying) as specified. Performs a software blend if [HAL](#) does not support BLIT\_COPY\_WITH\_ALPHA and alpha != 255.

## Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
<i>sourceFormat</i>	The bitmap format used in the source data.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implemented in [LCD24bpp](#), [LCD2bpp](#), [LCD4bpp](#), [LCD16bpp](#), and [LCD1bpp](#).

#### 7.99.3.4 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
    const Rect & region,
    const BitmapId bitmap = BITMAP_ANIMATION_STORAGE ) [pure virtual]
```

Copies a part of the frame buffer to a bitmap. The bitmap must be a dynamic bitmap. Only the part specified with by parameter region is copied.

##### Parameters

<i>region</i>	The part to copy.
<i>bitmap</i>	The bitmap to store the data in. Default parameter is Animation Storage.

##### Returns

A pointer to the copy.

##### Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

##### See also

[blitCopy](#)

Implemented in [LCD24bpp](#), [LCD2bpp](#), [LCD4bpp](#), [LCD16bpp](#), and [LCD1bpp](#).

#### 7.99.3.5 drawBorder()

```
void drawBorder (
    const Rect & rect,
    uint16_t lineWidth,
    color_t color,
    uint8_t alpha = 255 )
```

Draws a rectangle with the specified line width and color.

##### Parameters

<i>rect</i>	The rectangle x, y, width, height in absolute coordinates.
<i>lineWidth</i>	The width of the line.
<i>color</i>	The color to use.
<i>alpha</i>	The alpha value to use (default=solid)

#### 7.99.3.6 drawGlyph() [1/2]

```
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

<i>x</i>	The x coordinate of the leftmost pixel.
<i>y</i>	The y coordinate of the topmost pixel.
<i>offsetX</i>	The offset x coordinate.
<i>offsetY</i>	The offset y coordinate.
<i>maxX</i>	The maximum x coordinate.
<i>maxY</i>	The maximum y coordinate.
<i>glyph</i>	The glyph.
<i>glyphData</i>	Information describing the glyph.
<i>color</i>	The color.
<i>bitsPerPixel</i>	The number of bits per pixel.
<i>alpha</i>	Default value is 255 (solid)

#### 7.99.3.7 drawGlyph() [2/2]

```

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	<i>wbuf</i>	The destination (frame) buffer to draw to.
	<i>widgetArea</i>	The canvas to draw the glyph inside.
	<i>x</i>	Horizontal offset to start drawing the glyph.

## Parameters

	<i>y</i>	Vertical offset to start drawing the glyph.
	<i>offsetX</i>	Horizontal offset in the glyph to start drawing from.
	<i>offsetY</i>	Vertical offset in the glyph to start drawing from.
	<i>invalidatedArea</i>	The area to draw inside.
	<i>glyph</i>	Specifications of the glyph to draw.
	<i>glyphData</i>	Data containing the actual glyph (dense format)
	<i>color</i>	The color of the glyph.
	<i>bitsPerPixel</i>	Bit depth of the glyph.
	<i>alpha</i>	The transparency of the glyph.
	<i>rotation</i>	Rotation to do before drawing the glyph.

Implemented in [LCD24bpp](#), [LCD2bpp](#), [LCD4bpp](#), [LCD16bpp](#), and [LCD1bpp](#).

## 7.99.3.8 drawHorizontalLine()

```
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

<i>x</i>	The x coordinate of the starting point, in absolute coordinates.
<i>y</i>	The y coordinate of the starting point, in absolute coordinates.
<i>width</i>	The length of the line.
<i>lineWidth</i>	The width of the line.
<i>color</i>	The color to use.
<i>alpha</i>	The alpha value to use (default=solid)

## 7.99.3.9 drawPartialBitmap()

```
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

<i>bitmap</i>	The bitmap to draw.
---------------	---------------------



## Parameters

<i>x</i>	The absolute x coordinate to place pixel (0,0) on the screen.
<i>y</i>	The absolute y coordinate to place pixel (0,0) on the screen.
<i>rect</i>	A rectangle describing what region of the bitmap is to be drawn.
<i>alpha</i>	Optional alpha value. Default is 255 (solid).
<i>useOptimized</i>	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implemented in [LCD24bpp](#), [LCD2bpp](#), [LCD4bpp](#), [LCD16bpp](#), and [LCD1bpp](#).

## 7.99.3.10 drawRect()

```
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

<i>rect</i>	The rectangle x, y, width, height in absolute coordinates.
<i>color</i>	The color to use.
<i>alpha</i>	The alpha value to use (default=solid)

## 7.99.3.11 drawString()

```
void drawString (
    Rect widgetArea,
    const Rect & invalidatedArea,
    StringVisuals & stringVisuals,
    const Unicode::UnicodeChar * format,
    ... )
```

Draws the specified unicode string. Breaks line on newline.

## Parameters

	<i>widgetArea</i>	The area covered by the drawing widget in absolute coordinates.
	<i>invalidatedArea</i>	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 <a href="#">LCD</a> is (x, y, width, height) = (15, 15, 6, 6).
in	<i>stringVisuals</i>	The string visuals (font, alignment, line space, color) with which to draw this string.
	<i>format</i>	A pointer to a zero terminated text string with optional additional wildcard arguments.
	...	Variable arguments providing additional information.

## 7.99.3.12 drawStringLTR()

```
void drawStringLTR (
    const Rect & widgetArea,
    const Rect & invalidatedArea,
    const StringVisuals & visuals,
    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

	<i>widgetArea</i>	The area covered by the drawing widget in absolute coordinates.
	<i>invalidatedArea</i>	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	<i>visuals</i>	The string visuals (font, alignment, line space, color) with which to draw this string.
	<i>format</i>	A pointer to a zero terminated text string with optional additional wildcard arguments.
	<i>pArg</i>	Variable arguments providing additional information.

## See also

[drawString](#)

## 7.99.3.13 drawStringRTL()

```
void drawStringRTL (
    const Rect & widgetArea,
    const Rect & invalidatedArea,
    const StringVisuals & visuals,
    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

	<i>widgetArea</i>	The area covered by the drawing widget in absolute coordinates.
	<i>invalidatedArea</i>	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	<i>visuals</i>	The string visuals (font, alignment, line space, color) with which to draw this string.
	<i>format</i>	A pointer to a zero terminated text string with optional additional wildcard arguments.
	<i>pArg</i>	Variable arguments providing additional information.

See also

[drawString](#)

#### 7.99.3.14 drawTextureMapScanLine()

```
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

<i>dest</i>	The description of where the texture is drawn - can be used to issue a draw off screen.
<i>gradients</i>	The gradients using in interpolation across the scan line.
<i>leftEdge</i>	The left edge of the scan line.
<i>rightEdge</i>	The right edge of the scan line.
<i>texture</i>	The texture.
<i>absoluteRect</i>	The containing rectangle in absolute coordinates.
<i>dirtyAreaAbsolute</i>	The dirty area in absolute coordinates.
<i>renderVariant</i>	The render variant - includes the algorithm and the pixel format.
<i>alpha</i>	The alpha.
<i>subDivisionLength</i>	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](#), [LCD2bpp](#), [LCD4bpp](#), [LCD16bpp](#), and [LCD1bpp](#).

#### 7.99.3.15 drawTextureMapTriangle()

```
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

<i>dest</i>	The description of where the texture is drawn - can be used to issue a draw off screen.
<i>vertices</i>	The vertices of the triangle.
<i>texture</i>	The texture.
<i>absoluteRect</i>	The containing rectangle in absolute coordinates.
<i>dirtyAreaAbsolute</i>	The dirty area in absolute coordinates.
<i>renderVariant</i>	The render variant - includes the algorithm and the pixel format.
<i>alpha</i>	the alpha. Default = 255.
<i>subDivisionSize</i>	the size of the subdivisions of the scan line. Default = 12.

## 7.99.3.16 drawVerticalLine()

```
void drawVerticalLine (
    int16_t x,
    int16_t y,
    uint16_t height,
    uint16_t lineWidth,
    color_t color,
    uint8_t alpha = 255 )
```

Draws a vertical line with the specified color.

## Parameters

<i>x</i>	The x coordinate of the starting point, in absolute coordinates.
<i>y</i>	The y coordinate of the starting point, in absolute coordinates.
<i>height</i>	The length of the line.
<i>lineWidth</i>	The width of the line.
<i>color</i>	The color to use.
<i>alpha</i>	The alpha value to use (default=solid)

## 7.99.3.17 fillRect()

```
void fillRect (
    const Rect & rect,
    color_t color,
    uint8_t alpha = 255 ) [pure virtual]
```

Draws a filled rectangle in the specified color.

## Parameters

<i>rect</i>	The rectangle to draw in absolute coordinates.
<i>color</i>	The rectangle color.
<i>alpha</i>	The rectangle opacity (255=solid)

Implemented in [LCD24bpp](#), [LCD2bpp](#), [LCD4bpp](#), [LCD16bpp](#), and [LCD1bpp](#).

## 7.99.3.18 getNumLines()

```
static uint16_t getNumLines (
    TextProvider & textProvider,
    WideTextAction wideTextAction,
    TextDirection textDirection,
    const Font * font,
    int16_t width ) [static], [protected]
```

Gets number of lines for a given text taking word wrap into consideration. The font and width are required to find the number of lines in case word wrap is true.

## Parameters

in	<i>textProvider</i>	The text provider.
	<i>wideTextAction</i>	The wide text action in case lines are longer than the width of the text area.
	<i>textDirection</i>	The text direction (LTR or RTL).
	<i>font</i>	The font.
	<i>width</i>	The width.

## Returns

The number lines.

## 7.99.3.19 init()

```
void init ( ) [inline], [virtual]
```

Performs initialization.

Reimplemented in [LCD2bpp](#), [LCD4bpp](#), [LCD16bpp](#), and [LCD24bpp](#).

## 7.99.3.20 realX()

```
static int realX (
    const 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	<i>widgetArea</i>	The widget containing the point.
	<i>x</i>	The x coordinate.
	<i>y</i>	The y coordinate.
	<i>rotation</i>	Rotation to perform.

**Returns**

The absolute x coordinate after applying appropriate rotation.

**7.99.3.21 realY()**

```
static int realY (
    const 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	<i>widgetArea</i>	The widget containing the point.
	<i>x</i>	The x coordinate.
	<i>y</i>	The y coordinate.
	<i>rotation</i>	Rotation to perform.

**Returns**

The absolute y coordinate after applying appropriate rotation.

**7.99.3.22 rotateRect()**

```
static void rotateRect (
    Rect & rect,
    const Rect & canvas,
    const TextRotation rotation ) [static], [protected]
```

Rotate a rectangle inside another rectangle.

**Parameters**

in, out	<i>rect</i>	The rectangle to rotate.
	<i>canvas</i>	The rectangle containing the rect to rotate.
	<i>rotation</i>	Rotation to perform on rect.

**7.99.3.23 stringWidth()**

```
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

<code>in, out</code>	<code>textProvider</code>	The text provider.
	<code>font</code>	The font.
	<code>numChars</code>	Number of characters.
	<code>textDirection</code>	The text direction.

## Returns

An `int16_t`.

## 7.100 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, const `BitmapId` bitmap=BITMAP↵\_ANIMATION\_STORAGE)  
*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`↵offsetY, 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.*

### 7.100.1 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](#)

### 7.100.2 Member Function Documentation

#### 7.100.2.1 bitDepth()

```
uint8_t bitDepth ( ) const [inline], [virtual]
```

Number of bits per pixel used by the display.

#### Returns

The number of bits per pixel.

Implements [LCD](#).



## 7.100.2.2 blitCopy() [1/2]

```
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 performing alpha-blending (and transparency keying) as specified. Performs a software blend if [HAL](#) does not support BLIT\_COPY\_WITH\_ALPHA and alpha != 255.

## Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements [LCD](#).

## 7.100.2.3 blitCopy() [2/2]

```
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 performing alpha-blending (and transparency 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

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
<i>sourceFormat</i>	The bitmap format used in the source data.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements [LCD](#).

#### 7.100.2.4 blitCopyAlphaPerPixel()

```
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 performing alpha-blending per pixel as specified. Performs always a software blend.

##### Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16- bits RGB565 values.
<i>alphaData</i>	The alpha channel array pointer (points to the beginning of the data)
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

#### 7.100.2.5 blitCopyARGB8888()

```
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 performing alpha-blending per pixel as specified if ARGB8888 is not supported by the DMA a software blend is performed.

##### Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

#### 7.100.2.6 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
    const Rect & region,
    const BitmapId bitmap = BITMAP_ANIMATION_STORAGE ) [virtual]
```

Copies a part of the frame buffer to a bitmap.

##### Parameters

<i>region</i>	The part to copy.
<i>bitmap</i>	The bitmap to store the data in. Default parameter is Animation Storage.

**Returns**

A pointer to the copy.

Implements [LCD](#).

**7.100.2.7 drawGlyph()**

```
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	<i>wbuf</i>	The destination (frame) buffer to draw to.
	<i>widgetArea</i>	The canvas to draw the glyph inside.
	<i>x</i>	Horizontal offset to start drawing the glyph.
	<i>y</i>	Vertical offset to start drawing the glyph.
	<i>offsetX</i>	Horizontal offset in the glyph to start drawing from.
	<i>offsetY</i>	Vertical offset in the glyph to start drawing from.
	<i>invalidatedArea</i>	The area to draw within.
	<i>glyph</i>	Specifications of the glyph to draw.
	<i>glyphData</i>	Data containing the actual glyph (dense format)
	<i>color</i>	The color of the glyph.
	<i>bitsPerPixel</i>	Bit depth of the glyph.
	<i>alpha</i>	The transparency of the glyph.
	<i>rotation</i>	Rotation to do before drawing the glyph.

Implements [LCD](#).

**7.100.2.8 drawPartialBitmap()**

```
void drawPartialBitmap (
    const Bitmap & bitmap,
    int16_t x,
    int16_t y,
    const Rect & rect,
```

```
uint8_t alpha = 255,
bool useOptimized = true ) [virtual]
```

#### Parameters

<i>bitmap</i>	The bitmap to draw.
<i>x</i>	The absolute x coordinate to place pixel (0, 0) on the screen.
<i>y</i>	The absolute y coordinate to place pixel (0, 0) on the screen.
<i>rect</i>	A rectangle describing what region of the bitmap is to be drawn.
<i>alpha</i>	Optional alpha value. Default is 255 (solid).
<i>useOptimized</i>	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements [LCD](#).

#### 7.100.2.9 drawTextureMapScanLine()

```
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

<i>dest</i>	The description of where the texture is drawn - can be used to issue a draw off screen.
<i>gradients</i>	The gradients using in interpolation across the scan line.
<i>leftEdge</i>	The left edge of the scan line.
<i>rightEdge</i>	The right edge of the scan line.
<i>texture</i>	The texture.
<i>absoluteRect</i>	The containing rectangle in absolute coordinates.
<i>dirtyAreaAbsolute</i>	The dirty area in absolute coordinates.
<i>renderVariant</i>	The render variant - includes the algorithm and the pixel format.
<i>alpha</i>	The alpha.
<i>subDivisionSize</i>	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](#).

**7.100.2.10 fillRect()**

```
void fillRect (
    const Rect & rect,
    color_t color,
    uint8_t alpha = 255 ) [virtual]
```

Draws a filled rectangle in the specified color.

**Parameters**

<i>rect</i>	The rectangle to draw in absolute coordinates.
<i>color</i>	The rectangle color.
<i>alpha</i>	The rectangle opacity (255=solid)

Implements [LCD](#).

**7.100.2.11 init()**

```
void init ( ) [virtual]
```

Performs initialization.

Reimplemented from [LCD](#).

**7.100.2.12 nextLine()**

```
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**

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

**Returns**

How much to advance to get to the next line.

**7.100.2.13 nextPixel()**

```
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

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

## Returns

How much to advance to get to the next pixel.

## 7.101 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, const [BitmapId](#) bitmap=BITMAP\_↔\_ANIMATION\_STORAGE)  
*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 ↔offsetY, 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, [color\\_t](#) color, uint16\_t bytesToFill)  
*Fill memory efficiently.*

## Additional Inherited Members

### 7.101.1 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](#)

### 7.101.2 Member Function Documentation

#### 7.101.2.1 bitDepth()

```
uint8_t bitDepth ( ) const [inline], [virtual]
```

Number of bits per pixel used by the display.

#### Returns

1.

Implements [LCD](#).

#### 7.101.2.2 blitCopy() [1/2]

```
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

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (0 = invisible, otherwise solid).
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements [LCD](#).

## 7.101.2.3 blitCopy() [2/2]

```
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 performing alpha-blending (and transparency keying) as specified. Performs a software blend if [HAL](#) does not support BLIT\_COPY\_WITH\_ALPHA and alpha != 255.

## Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
<i>sourceFormat</i>	The bitmap format used in the source data.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	Ignored

Implements [LCD](#).

## 7.101.2.4 blitCopyRLE()

```
void blitCopyRLE (
    const uint16_t * _sourceData,
    const Rect & source,
    const Rect & blitRect,
    uint8_t alpha ) [protected], [virtual]
```

Blits a run-length encoded 2D source-array to the frame buffer unless alpha is zero.

## Parameters

<i>_sourceData</i>	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.
--------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



## Parameters

<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (0 = invisible, otherwise solid).

## 7.101.2.5 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
    const Rect & region,
    const BitmapId bitmap = BITMAP_ANIMATION_STORAGE ) [virtual]
```

Copies a part of the frame buffer to a bitmap.

## Parameters

<i>region</i>	The part to copy.
<i>bitmap</i>	The bitmap to store the data in. Default value is Animation Storage.

## Returns

A pointer to the copy.

Implements [LCD](#).

## 7.101.2.6 drawGlyph()

```
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	<i>wbuf</i>	The destination (frame) buffer to draw to.
	<i>widgetArea</i>	The canvas to draw the glyph inside.
	<i>x</i>	Horizontal offset to start drawing the glyph.
	<i>y</i>	Vertical offset to start drawing the glyph.
	<i>offsetX</i>	Horizontal offset in the glyph to start drawing from.

## Parameters

	<i>offsetY</i>	Vertical offset in the glyph to start drawing from.
	<i>invalidatedArea</i>	The area to draw within.
	<i>glyph</i>	Specifications of the glyph to draw.
	<i>glyphData</i>	Data containing the actual glyph (dense format)
	<i>color</i>	The color of the glyph.
	<i>bitsPerPixel</i>	Bit depth of the glyph.
	<i>alpha</i>	The transparency of the glyph.
	<i>rotation</i>	Rotation to do before drawing the glyph.

Implements [LCD](#).

## 7.101.2.7 drawPartialBitmap()

```
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

<i>bitmap</i>	The bitmap to draw.
<i>x</i>	The absolute x coordinate to place pixel (0, 0) on the screen.
<i>y</i>	The absolute y coordinate to place pixel (0, 0) on the screen.
<i>rect</i>	A rectangle describing what region of the bitmap is to be drawn.
<i>alpha</i>	Optional alpha value (0 = invisible, otherwise solid). Default is 255 (solid).
<i>useOptimized</i>	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements [LCD](#).

## 7.101.2.8 drawTextureMapScanLine()

```
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

<i>dest</i>	The description of where the texture is drawn - can be used to issue a draw off screen.
<i>gradients</i>	The gradients using in interpolation across the scan line.
<i>leftEdge</i>	The left edge of the scan line.
<i>rightEdge</i>	The right edge of the scan line.
<i>texture</i>	The texture.
<i>absoluteRect</i>	The containing rectangle in absolute coordinates.
<i>dirtyAreaAbsolute</i>	The dirty area in absolute coordinates.
<i>renderVariant</i>	The render variant - includes the algorithm and the pixel format.
<i>alpha</i>	The alpha.
<i>subDivisionSize</i>	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](#).

## 7.101.2.9 fillMemory()

```
static void fillMemory (
    void *RESTRICT dst,
    color_t 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	<i>dst</i>	Pointer to memory to fill.
	<i>color</i>	<a href="#">Color</a> to write to memory, either 0 => 0x00000000 or 1 => 0xFFFFFFFF.
	<i>bytesToFill</i>	Number of bytes to fill.

## 7.101.2.10 fillRect()

```
void fillRect (
    const Rect & rect,
    color_t color,
    uint8_t alpha = 255 ) [virtual]
```

Draws a filled rectangle in the specified color.

## Parameters

<i>rect</i>	The rectangle to draw in absolute coordinates.
<i>color</i>	The rectangle color (values other than 0 or 1 are treated as being 1).
<i>alpha</i>	The rectangle opacity (0 = invisible, otherwise solid). Default is 255 (solid).

Implements [LCD](#).

**7.101.2.11 nextLine()**

```
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**

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

**Returns**

How much to advance to get to the next line.

**7.101.2.12 nextPixel()**

```
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**

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

**Returns**

How much to advance to get to the next pixel.

**7.102 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, const [BitmapId](#) bitmap=[BITMAP\\_↵\\_ANIMATION\\_STORAGE](#))

*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 offsetY, 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.*

### 7.102.1 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](#)

## 7.102.2 Member Function Documentation

### 7.102.2.1 bitDepth()

```
uint8_t bitDepth ( ) const [inline], [virtual]
```

Number of bits per pixel used by the display.

#### Returns

The number of bits per pixel.

Implements [LCD](#).

### 7.102.2.2 blitCopy() [1/2]

```
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 performing alpha-blending (and transparency keying) as specified. Performs a software blend if [HAL](#) does not support BLIT\_COPY\_WITH\_ALPHA and alpha != 255.

#### Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements [LCD](#).

### 7.102.2.3 blitCopy() [2/2]

```
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 performing alpha-blending (and transparency 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

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
<i>sourceFormat</i>	The bitmap format used in the source data.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements [LCD](#).

## 7.102.2.4 blitCopyARGB8888()

```
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 performing alpha-blending per pixel as specified if ARGB8888 is not supported by the DMA a software blend is performed.

## Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

## 7.102.2.5 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
    const Rect & region,
    const BitmapId bitmap = BITMAP_ANIMATION_STORAGE ) [virtual]
```

Copies a part of the frame buffer to a bitmap.

## Parameters

<i>region</i>	The part to copy.
<i>bitmap</i>	The bitmap to store the data in. Default parameter is Animation Storage.

## Returns

A pointer to the copy.

Implements [LCD](#).

## 7.102.2.6 drawGlyph()

```
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	<i>wbuf</i>	The destination (frame) buffer to draw to.
	<i>widgetArea</i>	The canvas to draw the glyph inside.
	<i>x</i>	Horizontal offset to start drawing the glyph.
	<i>y</i>	Vertical offset to start drawing the glyph.
	<i>offsetX</i>	Horizontal offset in the glyph to start drawing from.
	<i>offsetY</i>	Vertical offset in the glyph to start drawing from.
	<i>invalidatedArea</i>	The area to draw within.
	<i>glyph</i>	Specifications of the glyph to draw.
	<i>glyphData</i>	Data containing the actual glyph (dense format)
	<i>color</i>	The color of the glyph.
	<i>bitsPerPixel</i>	Bit depth of the glyph.
	<i>alpha</i>	The transparency of the glyph.
	<i>rotation</i>	Rotation to do before drawing the glyph.

Implements [LCD](#).

## 7.102.2.7 drawPartialBitmap()

```
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

<i>bitmap</i>	The bitmap to draw.
<i>x</i>	The absolute x coordinate to place pixel (0, 0) on the screen.



## Parameters

<i>y</i>	The absolute y coordinate to place pixel (0, 0) on the screen.
<i>rect</i>	A rectangle describing what region of the bitmap is to be drawn.
<i>alpha</i>	Optional alpha value. Default is 255 (solid).
<i>useOptimized</i>	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements [LCD](#).

## 7.102.2.8 drawTextureMapScanLine()

```
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 subDivisionSize will determine the size of the piecewise affine texture mapped lines.

## Parameters

<i>dest</i>	The description of where the texture is drawn - can be used to issue a draw off screen.
<i>gradients</i>	The gradients using in interpolation across the scan line.
<i>leftEdge</i>	The left edge of the scan line.
<i>rightEdge</i>	The right edge of the scan line.
<i>texture</i>	The texture.
<i>absoluteRect</i>	The containing rectangle in absolute coordinates.
<i>dirtyAreaAbsolute</i>	The dirty area in absolute coordinates.
<i>renderVariant</i>	The render variant - includes the algorithm and the pixel format.
<i>alpha</i>	The alpha.
<i>subDivisionSize</i>	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](#).

## 7.102.2.9 fillRect()

```
void fillRect (
    const Rect & rect,
    colortype color,
    uint8_t alpha = 255 ) [virtual]
```

Draws a filled rectangle in the specified color.

#### Parameters

<i>rect</i>	The rectangle to draw in absolute coordinates.
<i>color</i>	The rectangle color.
<i>alpha</i>	The rectangle opacity (255=solid)

Implements [LCD](#).

#### 7.102.2.10 init()

```
void init ( ) [virtual]
```

Performs initialization.

Reimplemented from [LCD](#).

#### 7.102.2.11 nextLine()

```
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

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

#### Returns

How much to advance to get to the next line.

#### 7.102.2.12 nextPixel()

```
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

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

## Returns

How much to advance to get to the next pixel.

## 7.103 LCD2bpp Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

```
#include <platform/driver/lcd/LCD2bpp.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, const [BitmapId](#) bitmap=BITMAP\_↵\_ANIMATION\_STORAGE)  
*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 ↵offsetY, 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 [blitCopyAlphaPerPixel](#) (const uint16\_t \*sourceData16, const uint8\_t \*sourceAlphaData, 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.*
- static const uint8\_t [alphaTable2bpp](#) [64]  
*The alpha lookup table to avoid arithmetics when alpha blending.*

### 7.103.1 Detailed Description

This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles on 2 bits per pixel grayscale displays.

#### Note

All coordinates are expected to be in absolute coordinates!

#### See also

[LCD](#)

### 7.103.2 Member Function Documentation

#### 7.103.2.1 bitDepth()

```
uint8_t bitDepth ( ) const [inline], [virtual]
```

Number of bits per pixel used by the display.

#### Returns

The number of bits per pixel.

Implements [LCD](#).

#### 7.103.2.2 blitCopy() [1/2]

```
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 performing alpha-blending (and transparency keying) as specified. Performs a software blend if [HAL](#) does not support BLIT\_COPY\_WITH\_ALPHA and alpha != 255.

## Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements [LCD](#).

## 7.103.2.3 blitCopy() [2/2]

```
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 performing alpha-blending (and transparency keying) as specified. Performs a software blend if [HAL](#) does not support BLIT\_COPY\_WITH\_ALPHA and alpha != 255. LCD2 supports source data formats: RGB565 and ARGB8888.

## Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
<i>sourceFormat</i>	The bitmap format used in the source data.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements [LCD](#).

## 7.103.2.4 blitCopyAlphaPerPixel()

```
static void blitCopyAlphaPerPixel (
    const uint16_t * sourceData16,
    const uint8_t * sourceAlphaData,
    const Rect & source,
    const Rect & blitRect,
    uint8_t alpha ) [static], [protected]
```

Blits a 2D source-array to the framebuffer performing alpha-blending per pixel as specified. Performs always a software blend.

## Parameters

<i>sourceData16</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 2bpp GREY2 values.
<i>sourceAlphaData</i>	The alpha channel array pointer (points to the beginning of the data)
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

## 7.103.2.5 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
    const Rect & region,
    const BitmapId bitmap = BITMAP_ANIMATION_STORAGE ) [virtual]
```

Copies a part of the frame buffer to a bitmap.

## Parameters

<i>region</i>	The part to copy.
<i>bitmap</i>	The bitmap to store the data in. Default parameter is Animation Storage.

## Returns

A pointer to the copy.

Implements [LCD](#).

## 7.103.2.6 drawGlyph()

```
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,
    color_t 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	<i>wbuf</i>	The destination (frame) buffer to draw to.
	<i>widgetArea</i>	The canvas to draw the glyph inside.

## Parameters

<i>x</i>	Horizontal offset to start drawing the glyph.
<i>y</i>	Vertical offset to start drawing the glyph.
<i>offsetX</i>	Horizontal offset in the glyph to start drawing from.
<i>offsetY</i>	Vertical offset in the glyph to start drawing from.
<i>invalidatedArea</i>	The area to draw within.
<i>glyph</i>	Specifications of the glyph to draw.
<i>glyphData</i>	Data containing the actual glyph (dense format)
<i>color</i>	The color of the glyph.
<i>bitsPerPixel</i>	Bit depth of the glyph.
<i>alpha</i>	The transparency of the glyph.
<i>rotation</i>	Rotation to do before drawing the glyph.

Implements [LCD](#).

## 7.103.2.7 drawPartialBitmap()

```
void drawPartialBitmap (
    const Bitmap & bitmap,
    int16_t x,
    int16_t y,
    const Rect & rect,
    uint8_t alpha = 255,
    bool useOptimized = true ) [virtual]
```

## Parameters

<i>bitmap</i>	The bitmap to draw.
<i>x</i>	The absolute x coordinate to place pixel (0, 0) on the screen.
<i>y</i>	The absolute y coordinate to place pixel (0, 0) on the screen.
<i>rect</i>	A rectangle describing what region of the bitmap is to be drawn.
<i>alpha</i>	Optional alpha value. Default is 255 (solid).
<i>useOptimized</i>	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements [LCD](#).

## 7.103.2.8 drawTextureMapScanLine()

```
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 subDivisionSize will determine the size of the piecewise affine texture mapped lines.

#### Parameters

<i>dest</i>	The description of where the texture is drawn - can be used to issue a draw off screen.
<i>gradients</i>	The gradients using in interpolation across the scan line.
<i>leftEdge</i>	The left edge of the scan line.
<i>rightEdge</i>	The right edge of the scan line.
<i>texture</i>	The texture.
<i>absoluteRect</i>	The containing rectangle in absolute coordinates.
<i>dirtyAreaAbsolute</i>	The dirty area in absolute coordinates.
<i>renderVariant</i>	The render variant - includes the algorithm and the pixel format.
<i>alpha</i>	The alpha.
<i>subDivisionSize</i>	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](#).

#### 7.103.2.9 fillRect()

```
void fillRect (
    const Rect & rect,
    colortype color,
    uint8_t alpha = 255 ) [virtual]
```

Draws a filled rectangle in the specified color.

#### Parameters

<i>rect</i>	The rectangle to draw in absolute coordinates.
<i>color</i>	The rectangle color.
<i>alpha</i>	The rectangle opacity (255=solid)

Implements [LCD](#).

#### 7.103.2.10 init()

```
void init ( ) [virtual]
```

Performs initialization.

Reimplemented from [LCD](#).



7.103.2.11 `nextLine()`

```
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

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

## Returns

How much to advance to get to the next line.

7.103.2.12 `nextPixel()`

```
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

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

## Returns

How much to advance to get to the next pixel.

## 7.104 LCD4bpp Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

```
#include <platform/driver/lcd/LCD4bpp.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, const [BitmapId](#) bitmap=[BITMAP\\_←\\_ANIMATION\\_STORAGE](#))

*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 offsetY, 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 [blitCopyAlphaPerPixel](#) (const uint16\_t \*sourceData16, const uint8\_t \*sourceAlphaData, 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.*

### 7.104.1 Detailed Description

This class contains the various low-level drawing routines for drawing bitmaps, texts and rectangles on 4 bits per pixel grayscale displays.

#### Note

All coordinates are expected to be in absolute coordinates!

#### See also

[LCD](#)

## 7.104.2 Member Function Documentation

### 7.104.2.1 bitDepth()

```
uint8_t bitDepth ( ) const [inline], [virtual]
```

Number of bits per pixel used by the display.

#### Returns

The number of bits per pixel.

Implements [LCD](#).

### 7.104.2.2 blitCopy() [1/2]

```
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 performing alpha-blending (and transparency keying) as specified. Performs a software blend if [HAL](#) does not support BLIT\_COPY\_WITH\_ALPHA and alpha != 255.

#### Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements [LCD](#).

### 7.104.2.3 blitCopy() [2/2]

```
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 performing alpha-blending (and transparency keying) as specified. Performs a software blend if [HAL](#) does not support BLIT\_COPY\_WITH\_ALPHA and alpha != 255. LCD4 supports source data formats: RGB565 and ARGB8888.

## Parameters

<i>sourceData</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
<i>sourceFormat</i>	The bitmap format used in the source data.
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending (255 = solid, no blending)
<i>hasTransparentPixels</i>	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements [LCD](#).

## 7.104.2.4 blitCopyAlphaPerPixel()

```
static void blitCopyAlphaPerPixel (
    const uint16_t * sourceData16,
    const uint8_t * sourceAlphaData,
    const Rect & source,
    const Rect & blitRect,
    uint8_t alpha ) [static], [protected]
```

Blits a 2D source-array to the framebuffer performing alpha-blending per pixel as specified. Performs always a software blend.

## Parameters

<i>sourceData16</i>	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 4bpp GREY4 values.
<i>sourceAlphaData</i>	The alpha channel array pointer (points to the beginning of the data)
<i>source</i>	The location and dimension of the source.
<i>blitRect</i>	A rectangle describing what region is to be drawn.
<i>alpha</i>	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

## 7.104.2.5 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
    const Rect & region,
    const BitmapId bitmap = BITMAP_ANIMATION_STORAGE ) [virtual]
```

Copies a part of the frame buffer to a bitmap.

## Parameters

<i>region</i>	The part to copy.
<i>bitmap</i>	The bitmap to store the data in. Default value is Animation Storage.

**Returns**

A pointer to the copy.

Implements [LCD](#).

**7.104.2.6 drawGlyph()**

```
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	<i>wbuf</i>	The destination (frame) buffer to draw to.
	<i>widgetArea</i>	The canvas to draw the glyph inside.
	<i>x</i>	Horizontal offset to start drawing the glyph.
	<i>y</i>	Vertical offset to start drawing the glyph.
	<i>offsetX</i>	Horizontal offset in the glyph to start drawing from.
	<i>offsetY</i>	Vertical offset in the glyph to start drawing from.
	<i>invalidatedArea</i>	The area to draw within.
	<i>glyph</i>	Specifications of the glyph to draw.
	<i>glyphData</i>	Data containing the actual glyph (dense format)
	<i>color</i>	The color of the glyph.
	<i>bitsPerPixel</i>	Bit depth of the glyph.
	<i>alpha</i>	The transparency of the glyph.
	<i>rotation</i>	Rotation to do before drawing the glyph.

Implements [LCD](#).

**7.104.2.7 drawPartialBitmap()**

```
void drawPartialBitmap (
    const Bitmap & bitmap,
    int16_t x,
    int16_t y,
    const Rect & rect,
```

```
uint8_t alpha = 255,
bool useOptimized = true ) [virtual]
```

#### Parameters

<i>bitmap</i>	The bitmap to draw.
<i>x</i>	The absolute x coordinate to place pixel (0, 0) on the screen.
<i>y</i>	The absolute y coordinate to place pixel (0, 0) on the screen.
<i>rect</i>	A rectangle describing what region of the bitmap is to be drawn.
<i>alpha</i>	Optional alpha value. Default is 255 (solid).
<i>useOptimized</i>	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements [LCD](#).

#### 7.104.2.8 drawTextureMapScanLine()

```
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 subDivisionSize will determine the size of the piecewise affine texture mapped lines.

#### Parameters

<i>dest</i>	The description of where the texture is drawn - can be used to issue a draw off screen.
<i>gradients</i>	The gradients using in interpolation across the scan line.
<i>leftEdge</i>	The left edge of the scan line.
<i>rightEdge</i>	The right edge of the scan line.
<i>texture</i>	The texture.
<i>absoluteRect</i>	The containing rectangle in absolute coordinates.
<i>dirtyAreaAbsolute</i>	The dirty area in absolute coordinates.
<i>renderVariant</i>	The render variant - includes the algorithm and the pixel format.
<i>alpha</i>	The alpha.
<i>subDivisionSize</i>	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](#).

### 7.104.2.9 fillRect()

```
void fillRect (
    const Rect & rect,
    color_t color,
    uint8_t alpha = 255 ) [virtual]
```

Draws a filled rectangle in the specified color.

#### Parameters

<i>rect</i>	The rectangle to draw in absolute coordinates.
<i>color</i>	The rectangle color.
<i>alpha</i>	The rectangle opacity (255=solid)

Implements [LCD](#).

### 7.104.2.10 init()

```
void init ( ) [virtual]
```

Performs initialization.

Reimplemented from [LCD](#).

### 7.104.2.11 nextLine()

```
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

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

#### Returns

How much to advance to get to the next line.

### 7.104.2.12 nextPixel()

```
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**

<i>portrait</i>	Is the display running in portrait mode?
<i>rotation</i>	Rotation to perform.

**Returns**

How much to advance to get to the next pixel.

## 7.105 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](#).*

### 7.105.1 Member Function Documentation

#### 7.105.1.1 [get\(\)](#)

```
static bool get (  
    uint8_t nr ) [static]
```

Get state of a [LED](#).

**Parameters**

<i>nr</i>	of the <a href="#">LED</a> to toggle.
-----------	---------------------------------------

**Returns**

the state of the [LED](#).

#### 7.105.1.2 [init\(\)](#)

```
static void init ( ) [static]
```



Perform configuration of IO pins.

#### 7.105.1.3 off()

```
static void off (
    uint8_t nr ) [static]
```

Turn off a [LED](#).

##### Parameters

<i>nr</i>	of the <a href="#">LED</a> to switch off.
-----------	-------------------------------------------

#### 7.105.1.4 on()

```
static void on (
    uint8_t nr ) [static]
```

Turn on a [LED](#).

##### Parameters

<i>nr</i>	of the <a href="#">LED</a> to switch on.
-----------	------------------------------------------

#### 7.105.1.5 toggle()

```
static void toggle (
    uint8_t nr ) [static]
```

Toggles a [LED](#).

##### Parameters

<i>nr</i>	of the <a href="#">LED</a> to toggle.
-----------	---------------------------------------

## 7.106 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](#), [ROUND\\_CAP\\_ENDING](#), [SQUARE\\_CAP\\_ENDING](#) }

*Values that represent line ending styles.*

## Public Member Functions

- [Line](#) ()  
Construct a new [Line](#).
- template<typename T >  
void [setLine](#) (T x1, T y1, T x2, T y2)  
Sets the endpoints of the line.
- template<typename T >  
void [setStart](#) (T x, T y)  
Sets the start point for this [Line](#).
- void [setStart](#) ([CWRUtil::Q5](#) xQ5, [CWRUtil::Q5](#) yQ5)  
Sets the start point for this [Line](#).
- template<typename T >  
void [updateStart](#) (T x, T y)  
Update the start point for this [Line](#).
- void [updateStart](#) ([CWRUtil::Q5](#) xQ5, [CWRUtil::Q5](#) yQ5)  
Update the start point for this [Line](#).
- template<typename T >  
void [getStart](#) (T &x, T &y) const  
Gets the start coordinates for the line.
- template<typename T >  
void [setEnd](#) (T x, T y)  
Sets the end point for this [Line](#).
- void [setEnd](#) ([CWRUtil::Q5](#) xQ5, [CWRUtil::Q5](#) yQ5)  
Sets the end point for this [Line](#).
- template<typename T >  
void [updateEnd](#) (T x, T y)  
Update the end point for this [Line](#).
- void [updateEnd](#) ([CWRUtil::Q5](#) xQ5, [CWRUtil::Q5](#) yQ5)  
Update the end point for this [Line](#).
- template<typename T >  
void [getEnd](#) (T &x, T &y) const  
Gets the end coordinates for the line.
- template<typename T >  
void [setLineWidth](#) (T width)  
Sets the width for this [Line](#).
- void [setLineWidth](#) ([CWRUtil::Q5](#) widthQ5)  
Sets the width for this [Line](#).
- template<typename T >  
void [updateLineWidth](#) (T width)  
Update the width for this [Line](#).
- void [updateLineWidth](#) ([CWRUtil::Q5](#) widthQ5)  
Update the width for this [Line](#).
- template<typename T >  
void [getLineWidth](#) (T &width) const  
Gets line width.
- template<typename T >  
T [getLineWidth](#) () const  
Gets line width.
- void [setLineEndingStyle](#) ([LINE\\_ENDING\\_STYLE](#) lineEnding)  
Sets line ending style.
- [LINE\\_ENDING\\_STYLE](#) [getLineEndingStyle](#) () const

*Gets line ending style.*

- void [setCapPrecision](#) (int precision)  
*Sets a precision of the ends of the [Line](#).*
- 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

### 7.106.1 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](#)

### 7.106.2 Member Enumeration Documentation

#### 7.106.2.1 LINE\_ENDING\_STYLE

enum [LINE\\_ENDING\\_STYLE](#)

Enumerator

BUTT_CAP_ENDING	The line ending is cut 90 degrees at the end of the line.
ROUND_CAP_ENDING	The line ending is rounded as a circle with center at the end of the line.
SQUARE_CAP_ENDING	The line ending is cut 90 degrees, but extends half the width of the line.

### 7.106.3 Constructor & Destructor Documentation

#### 7.106.3.1 Line()

[Line](#) ( )

Construct a new line.

## 7.106.4 Member Function Documentation

### 7.106.4.1 drawCanvasWidget()

```
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

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

#### Returns

true if it succeeds, false if it fails.

Implements [CanvasWidget](#).

### 7.106.4.2 getEnd()

```
template< typename T > void getEnd (
    T & x,
    T & y ) const [inline]
```

Gets the end coordinates for the line.

#### Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

#### Parameters

out	<i>x</i>	The x coordinate.
out	<i>y</i>	The y coordinate.

#### See also

[setEnd](#)  
[setLine](#)

### 7.106.4.3 getLineEndingStyle()

```
LINE\_ENDING\_STYLE getLineEndingStyle ( ) const
```

Gets line ending style. See [LINE\\_ENDING\\_STYLE](#) for possible styles.

**Returns**

The line ending style.

**See also**

[LINE\\_ENDING\\_STYLE](#)  
[setLineEndingStyle](#)

**7.106.4.4 getLineWidth()** [1/2]

```
template< typename T > void getLineWidth (
    T & width ) const [inline]
```

Gets line width.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

out	<i>width</i>	The line width.
-----	--------------	-----------------

**See also**

[setLineWidth](#)

**7.106.4.5 getLineWidth()** [2/2]

```
template< typename T > T getLineWidth ( ) const [inline]
```

Gets line width.

**Template Parameters**

<i>T</i>	Generic type parameter, either int (default) or float.
----------	--------------------------------------------------------

**Returns**

The line width.

**See also**

[setLineWidth](#)

param [out] width The line width.

7.106.4.6 `getMinimalRect()`

```
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](#).

7.106.4.7 `getStart()`

```
template< typename T > void getStart (
    T & x,
    T & y ) const [inline]
```

Gets the start coordinates for the line.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

out	<i>x</i>	The x coordinate.
out	<i>y</i>	The y coordinate.

**See also**

[setStart](#)  
[setLine](#)

7.106.4.8 `setCapPrecision()`

```
void setCapPrecision (
    int precision )
```

Sets a precision of the ends of the [Line](#) arc. The precision is given in degrees where 18 is the default which results in a nice half circle with 10 line segments. 90 will draw "an arrow head".

**Note**

The line is not invalidated.  
 This is only used if line ending is set to `ROUND_CAP_ENDING`.

**Parameters**

<i>precision</i>	The new cap precision.
------------------	------------------------

**7.106.4.9 setEnd()** [1/2]

```
template< typename 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**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>x</i>	The x coordinate of the end point.
<i>y</i>	The y coordinate of the end point.

**See also**

[updateEnd](#)  
[getEnd](#)

**7.106.4.10 setEnd()** [2/2]

```
void setEnd (  
    CWRUtil::Q5 xQ5,  
    CWRUtil::Q5 yQ5 )
```

Sets the end point for this [Line](#).

**Parameters**

<i>xQ5</i>	The x coordinate of the end point in Q5 format.
<i>yQ5</i>	The y coordinate of the end point in Q5 format.

**Note**

The area containing the [Line](#) is not invalidated.

**See also**

[updateEnd](#)  
[getEnd](#)

#### 7.106.4.11 setLine()

```
template< typename 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

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

##### Parameters

<i>x1</i>	The x coordinate of the start point.
<i>y1</i>	The y coordinate of the start point.
<i>x2</i>	The x coordinate of the end point.
<i>y2</i>	The y coordinate of the end point.

##### See also

[setStart](#)  
[setEnd](#)

#### 7.106.4.12 setLineEndingStyle()

```
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

<i>lineEnding</i>	The line ending style.
-------------------	------------------------

##### See also

[LINE\\_ENDING\\_STYLE](#)  
[getLineEndingStyle](#)



**7.106.4.13 setLineWidth()** [1/2]

```
template< typename T > void setLineWidth (
    T width ) [inline]
```

Sets the width for this [Line](#).

**Note**

The area containing the [Line](#) is not invalidated.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>width</i>	The width of the line measured in pixels.
--------------	-------------------------------------------

**See also**

[updateLineWidth](#)

**7.106.4.14 setLineWidth()** [2/2]

```
void setLineWidth (
    CWRUtil::Q5 widthQ5 ) [inline]
```

Sets the width for this [Line](#).

**Parameters**

<i>widthQ5</i>	The width of the line measured in pixels in Q5 format.
----------------	--------------------------------------------------------

**Note**

The area containing the [Line](#) is not invalidated.

**See also**

[updateLineWidth](#)

**7.106.4.15 setStart()** [1/2]

```
template< typename 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**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>x</i>	The x coordinate of the start point.
<i>y</i>	The y coordinate of the start point.

**See also**

[updateStart](#)  
[getStart](#)  
[setLine](#)  
[setEnd](#)

**7.106.4.16 setStart()** [2/2]

```
void setStart (
    CWRUtil::Q5 xQ5,
    CWRUtil::Q5 yQ5 )
```

Sets the start point for this [Line](#).

**Parameters**

<i>xQ5</i>	The x coordinate of the start point in Q5 format.
<i>yQ5</i>	The y coordinate of the start point in Q5 format.

**Note**

The area containing the [Line](#) is not invalidated.

**See also**

[updateStart](#)  
[getStart](#)  
[setLine](#)  
[setEnd](#)

**7.106.4.17 updateEnd()** [1/2]

```
template< typename 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.

**Note**

The area containing the [Line](#) is invalidated before and after the change.

**Template Parameters**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>x</i>	The x coordinate of the end point.
<i>y</i>	The y coordinate of the end point.

**7.106.4.18 updateEnd()** [2/2]

```
void updateEnd (
    CWRUtil::Q5 xQ5,
    CWRUtil::Q5 yQ5 )
```

Update the end point for this [Line](#). The rectangle that surrounds the line before and after will be invalidated.

**Parameters**

<i>xQ5</i>	The x coordinate of the end point in Q5 format.
<i>yQ5</i>	The y coordinate of the end point in Q5 format.

**Note**

The area containing the [Line](#) is invalidated before and after the change.

**7.106.4.19 updateLineWidth()** [1/2]

```
template< typename 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**

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

**Parameters**

<i>width</i>	The width of the line measured in pixels.
--------------	-------------------------------------------

See also

[setLineWidth](#)

#### 7.106.4.20 updateLineWidth() [2/2]

```
void updateLineWidth (
    CWRUtil::Q5 widthQ5 ) [inline]
```

Update the width for this [Line](#) and invalidates the minimal rectangle surrounding the line on screen.

##### Parameters

<i>widthQ5</i>	The width of the line measured in pixels in Q5 format.
----------------	--------------------------------------------------------

##### Note

The area containing the [Line](#) is invalidated before and after the change.

See also

[setLineWidth](#)

#### 7.106.4.21 updateStart() [1/2]

```
template< typename 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.

##### Note

The area containing the [Line](#) is invalidated before and after the change.

##### Template Parameters

<i>T</i>	Generic type parameter, either int or float.
----------	----------------------------------------------

##### Parameters

<i>x</i>	The x coordinate of the start point.
<i>y</i>	The y coordinate of the start point.

See also

[setStart](#)

[updateEnd](#)

## 7.106.4.22 updateStart() [2/2]

```
void updateStart (
    CWRUtil::Q5 xQ5,
    CWRUtil::Q5 yQ5 )
```

Update the start point for this [Line](#). The rectangle that surrounds the line before and after will be invalidated.

## Parameters

xQ5	The x coordinate of the start point in Q5 format.
yQ5	The y coordinate of the start point in Q5 format.

## Note

The area containing the [Line](#) is invalidated before and after the change.

## See also

[setStart](#)  
[updateEnd](#)

## 7.107 LineProgress Class Reference

A line progress.

```
#include <touchgfx/containers/progress_indicators/LineProgress.hpp>
```

## Public Member Functions

- [LineProgress](#) ()  
*Default constructor.*
- virtual [~LineProgress](#) ()  
*Destructor.*
- virtual void [setProgressIndicatorPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets the position and dimension of the line progress indicator.*
- virtual void [setPainter](#) ([AbstractPainter](#) &painter)  
*Sets a painter.*
- virtual void [setStart](#) (int x, int y)  
*Sets a starting point for the line.*
- virtual void [getStart](#) (int &x, int &y) const  
*Gets the coordinates of the starting point of the line.*
- virtual void [setEnd](#) (int x, int y)  
*Sets an end point for the line.*
- virtual void [getEnd](#) (int &x, int &y) const  
*Gets the coordinates of the end point of the line.*
- virtual void [setLineWidth](#) (int width)  
*Sets line width.*
- virtual int [getLineWidth](#) () const  
*Gets line width.*
- virtual void [setLineEndingStyle](#) ([Line::LINE\\_ENDING\\_STYLE](#) lineEndingStyle)  
*Sets line ending style.*

- virtual [Line::LINE\\_ENDING\\_STYLE](#) [getLineEndingStyle](#) () const  
*Gets line ending style.*
- virtual void [setAlpha](#) (uint8\_t alpha)  
*Sets the alpha.*
- virtual uint8\_t [getAlpha](#) () const  
*Gets the alpha.*
- virtual void [setValue](#) (int value)  
*Sets a value.*

## Protected Attributes

- [Line line](#)  
*The line.*
- [CWRUtil::Q5 startX](#)  
*The start x coordinate.*
- [CWRUtil::Q5 startY](#)  
*The start y coordinate.*
- [CWRUtil::Q5 endX](#)  
*The end x coordinate.*
- [CWRUtil::Q5 endY](#)  
*The end y coordinate.*

## Additional Inherited Members

### 7.107.1 Detailed Description

Using [Line](#) from canvas widgets, progress will be rendered as a line. The line does not need to be horizontal or vertical, but can start at any coordinate and finish at any coordinate.

#### Note

As [LineProgress](#) uses [CanvasWidgetRenderer](#), it is important that a buffer is set up by calling `CanvasWidgetRenderer::setBuffer()`.

### 7.107.2 Constructor & Destructor Documentation

#### 7.107.2.1 [LineProgress\(\)](#)

[LineProgress](#) ( )

Default constructor.

#### 7.107.2.2 [~LineProgress\(\)](#)

[~LineProgress](#) ( ) [virtual]

Destructor.

### 7.107.3 Member Function Documentation

### 7.107.3.1 getAlpha()

```
uint8_t getAlpha ( ) const [virtual]
```

Gets the alpha.

#### Returns

The alpha.

#### See also

[Line::getAlpha](#)

### 7.107.3.2 getEnd()

```
void getEnd (
    int & x,
    int & y ) const [virtual]
```

Gets the coordinates of the end point of the line. Beware that this is not the coordinates of the current progress of the line, but the coordinates when the line is at 100%.

#### Parameters

out	<i>x</i>	The x coordinate.
out	<i>y</i>	The y coordinate.

### 7.107.3.3 getLineEndingStyle()

```
Line::LINE_ENDING_STYLE getLineEndingStyle ( ) const [virtual]
```

Gets line ending style.

#### Returns

The line ending style.

### 7.107.3.4 getLineWidth()

```
int getLineWidth ( ) const [virtual]
```

Gets line width.

#### Returns

The line width.

#### 7.107.3.5 `getStart()`

```
void getStart (
    int & x,
    int & y ) const [virtual]
```

Gets the coordinates of the starting point of the line.

##### Parameters

out	<i>x</i>	The x coordinate.
out	<i>y</i>	The y coordinate.

#### 7.107.3.6 `setAlpha()`

```
void setAlpha (
    uint8_t alpha ) [virtual]
```

Sets the alpha for the line.

##### Parameters

<i>alpha</i>	The alpha.
--------------	------------

##### See also

[Line::setAlpha](#)

#### 7.107.3.7 `setEnd()`

```
void setEnd (
    int x,
    int y ) [virtual]
```

Sets an end point for the line. When progress is at 100%, the line will go from the coordinates set by [setStart\(\)](#) to these coordinates set by [setEnd\(\)](#)

##### Parameters

<i>x</i>	The x coordinate of the end point.
<i>y</i>	The y coordinate of the end point.

##### See also

[setStart](#)

#### 7.107.3.8 `setLineEndingStyle()`

```
void setLineEndingStyle (
```



```
Line::LINE_ENDING_STYLE lineEndingStyle ) [virtual]
```

Sets line ending style.

#### Parameters

<i>lineEndingStyle</i>	The line ending style.
------------------------	------------------------

#### See also

[Line::setLineEndingStyle](#)

#### 7.107.3.9 setLineWidth()

```
void setLineWidth (
    int width ) [virtual]
```

Sets line width.

#### Parameters

<i>width</i>	The width.
--------------	------------

#### See also

[Line::setLineWidth](#)

#### 7.107.3.10 setPainter()

```
void setPainter (
    AbstractPainter & painter ) [virtual]
```

Sets a painter to be used for drawing the line. This can be any Painter, a simple single color painter, a bitmap painter or a custom painter.

#### Parameters

in	<i>painter</i>	The painter.
----	----------------	--------------

#### 7.107.3.11 setProgressIndicatorPosition()

```
void setProgressIndicatorPosition (
    int16_t x,
    int16_t y,
    int16_t width,
    int16_t height ) [virtual]
```

Sets the position and dimension of the line progress indicator relative to the background image.

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>width</i>	The width of the line progress indicator.
<i>height</i>	The height of the line progress indicator.

Reimplemented from [AbstractProgressIndicator](#).

**7.107.3.12 setStart()**

```
void setStart (
    int x,
    int y ) [virtual]
```

Sets a starting point for the line.

**Parameters**

<i>x</i>	The x coordinate of the start point.
<i>y</i>	The y coordinate of the start point.

**See also**

[setEnd](#)

**7.107.3.13 setValue()**

```
virtual void setValue (
    int value ) [virtual]
```

Sets the current value in the range (min..max) set by [setRange\(\)](#). Values lower than min are mapped to min, values higher than max are mapped to max.

**Parameters**

<i>value</i>	The value.
--------------	------------

Reimplemented from [AbstractProgressIndicator](#).

**7.108 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 `setDirection` (const `Direction` d)

*Sets the direction of the ListLayout.*

- virtual `Direction` `getDirection` () const

*Gets the direction of the ListLayout.*

- 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

### 7.108.1 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](#)

### 7.108.2 Constructor & Destructor Documentation

#### 7.108.2.1 ListLayout()

```
ListLayout (
    const Direction d = SOUTH ) [inline]
```

Constructor. Constructs a `ListLayout` instance that arranges the added elements in the specified `Direction`.

Parameters

<code>d</code>	The direction to grow in when adding children.
----------------	------------------------------------------------

#### 7.108.2.2 ~ListLayout()

```
~ListLayout ( ) [inline], [virtual]
```

Destructor.

### 7.108.3 Member Function Documentation

#### 7.108.3.1 add()

```
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	<i>d</i>	The <a href="#">Drawable</a> to add.
----	----------	--------------------------------------

Reimplemented from [Container](#).

#### 7.108.3.2 getDirection()

```
Direction getDirection ( ) const [inline], [virtual]
```

Gets the direction of the [ListLayout](#).

##### Returns

The current direction to grow in when added children (either SOUTH or EAST).

##### See also

[setDirection\(\)](#)

#### 7.108.3.3 getType()

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

##### Returns

TYPE\_LISTLAYOUT.

Reimplemented from [Container](#).

#### 7.108.3.4 insert()

```
void insert (
    Drawable * previousElement,
    Drawable & d ) [virtual]
```

Inserts a [Drawable](#).

## Parameters

in	<i>previousElement</i>	The element to insert the new element after.
in	<i>d</i>	The element to insert.

Reimplemented from [Container](#).

## 7.108.3.5 remove()

```
void remove (
    Drawable & d ) [virtual]
```

Removes a [Drawable](#). Safe to call even if drawable has not been added.

## Parameters

in	<i>d</i>	The drawable to remove.
----	----------	-------------------------

Reimplemented from [Container](#).

## 7.108.3.6 removeAll()

```
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](#).

## 7.108.3.7 setDirection()

```
void setDirection (
    const Direction d ) [virtual]
```

Sets the direction of the [ListLayout](#). If elements have already been added to the [ListLayout](#), these elements will be repositioned to adhere to the new direction.

## Parameters

<i>d</i>	The new Direction to grow in when added children (either SOUTH or EAST).
----------	--------------------------------------------------------------------------

## See also

[getDirection\(\)](#)

## 7.109 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.*

### 7.109.1 Detailed Description

This implements a simple lock-free FIFO queue (single producer, single consumer)

See also

[DMA\\_Queue](#)

### 7.109.2 Constructor & Destructor Documentation

#### 7.109.2.1 LockFreeDMA\_Queue()

```
LockFreeDMA_Queue (
    BlitOp * mem,
    atomic_t n )
```

Constructs a lockfree queue.

#### Parameters

out	<i>mem</i>	Pointer to the memory used by the queue to store elements.
	<i>n</i>	Number of elements the memory provided can contain.

### 7.109.3 Member Function Documentation

#### 7.109.3.1 first()

```
const BlitOp * first ( ) [protected], [virtual]
```

##### Returns

the first blitop.

Implements [DMA\\_Queue](#).

#### 7.109.3.2 isEmpty()

```
bool isEmpty ( ) [virtual]
```

Query if this object is empty.

##### Returns

true if empty, false if not.

Implements [DMA\\_Queue](#).

#### 7.109.3.3 isFull()

```
bool isFull ( ) [virtual]
```

Query if this object is full.

##### Returns

true if full, false if not.

Implements [DMA\\_Queue](#).

#### 7.109.3.4 pop()

```
void pop ( ) [protected], [virtual]
```

Removes the top-of-stack object.

Implements [DMA\\_Queue](#).

#### 7.109.3.5 pushCopyOf()

```
void pushCopyOf (
    const BlitOp & op ) [virtual]
```

Push copy of the given operation.

## Parameters

<i>op</i>	The operation.
-----------	----------------

Implements [DMA\\_Queue](#).

## 7.110 Matrix4x4 Class Reference

This class represents row major 4x4 homogeneous matrices.

```
#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].*



### 7.110.1 Detailed Description

This class represents row major 4x4 homogeneous matrices.

### 7.110.2 Constructor & Destructor Documentation

#### 7.110.2.1 Matrix4x4()

```
Matrix4x4 ( )
```

Default constructor.

### 7.110.3 Member Function Documentation

#### 7.110.3.1 concatenateXRotation()

```
Matrix4x4 & concatenateXRotation (
    float radians )
```

Concatenate x coordinate rotation.

##### Parameters

<i>radians</i>	The radians.
----------------	--------------

##### Returns

A matrix\_4x4&

#### 7.110.3.2 concatenateXScale()

```
Matrix4x4 & concatenateXScale (
    float distance )
```

Concatenate x coordinate scale.

##### Parameters

<i>distance</i>	The distance.
-----------------	---------------

##### Returns

A matrix\_4x4&

### 7.110.3.3 concatenateXTranslation()

```
Matrix4x4 & concatenateXTranslation (
    float distance )
```

Concatenate x coordinate translation.

#### Parameters

<i>distance</i>	The distance.
-----------------	---------------

#### Returns

A matrix\_4x4&

### 7.110.3.4 concatenateYRotation()

```
Matrix4x4 & concatenateYRotation (
    float radians )
```

Concatenate y coordinate rotation.

#### Parameters

<i>radians</i>	The radians.
----------------	--------------

#### Returns

A matrix\_4x4&

### 7.110.3.5 concatenateYScale()

```
Matrix4x4 & concatenateYScale (
    float distance )
```

Concatenate y coordinate scale.

#### Parameters

<i>distance</i>	The distance.
-----------------	---------------

#### Returns

A matrix\_4x4&

### 7.110.3.6 concatenateYTranslation()

```
Matrix4x4 & concatenateYTranslation (
    float distance )
```

Concatenate y coordinate translation.

#### Parameters

<i>distance</i>	The distance.
-----------------	---------------

#### Returns

A matrix\_4x4&

#### 7.110.3.7 concatenateZRotation()

```
Matrix4x4 & concatenateZRotation (
    float radians )
```

Concatenate z coordinate rotation.

#### Parameters

<i>radians</i>	The radians.
----------------	--------------

#### Returns

A matrix\_4x4&

#### 7.110.3.8 concatenateZScale()

```
Matrix4x4 & concatenateZScale (
    float distance )
```

Concatenate z coordinate scale.

#### Parameters

<i>distance</i>	The distance.
-----------------	---------------

#### Returns

A matrix\_4x4&

#### 7.110.3.9 concatenateZTranslation()

```
Matrix4x4 & concatenateZTranslation (
    float distance )
```

Concatenate z coordinate translation.

#### Parameters

<i>distance</i>	The distance.
-----------------	---------------

**Returns**

A matrix\_4x4&

**7.110.3.10 getElement()**

```
float getElement (
    int row,
    int column ) const [inline]
```

Gets an element.

**Parameters**

<i>row</i>	The row.
<i>column</i>	The column.

**Returns**

The element.

**7.110.3.11 identity()**

```
static Matrix4x4 identity ( ) [inline], [static]
```

Gets the identity. Instead of using "Matrix4x4::identity()" consider using "Matrix4x4()" instead.

**Returns**

A matrix\_4x4.

**7.110.3.12 setElement()**

```
Matrix4x4 setElement (
    int row,
    int column,
    float value ) [inline]
```

Sets an element.

**Parameters**

<i>row</i>	The row.
<i>column</i>	The column.
<i>value</i>	The value.

**Returns**

A matrix\_4x4&

7.110.3.13 `setViewDistance()`

```
void setViewDistance (
    float distance )
```

Sets view distance.

## Parameters

<i>distance</i>	The distance.
-----------------	---------------

## 7.111 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.*

#### 7.111.1 Detailed Description

Interface for instrumenting processors to measure MCU load via measured CPU cycles.

#### 7.111.2 Constructor & Destructor Documentation

### 7.111.2.1 MCUInstrumentation()

```
MCUInstrumentation ( ) [inline]
```

Constructor. Initializes members.

### 7.111.2.2 ~MCUInstrumentation()

```
~MCUInstrumentation ( ) [inline], [virtual]
```

Virtual destructor.

## 7.111.3 Member Function Documentation

### 7.111.3.1 getCCConsumed()

```
uint32_t getCCConsumed ( ) [inline], [virtual]
```

Gets number of consumed clock cycles.

#### Returns

clock cycles.

### 7.111.3.2 getCPUCycles()

```
unsigned int getCPUCycles (
    void ) [pure virtual]
```

Gets CPU cycles from register.

#### Returns

CPU cycles.

### 7.111.3.3 getElapsedUS()

```
unsigned int getElapsedUS (
    unsigned int start,
    unsigned int now,
    unsigned int clockfrequency ) [pure virtual]
```

Gets elapsed microseconds based on clock frequency.

#### Parameters

<i>start</i>	Start time.
<i>now</i>	Current time.
<i>clockfrequency</i>	Clock frequency of the system expressed in MHz.

**Returns**

Elapsed microseconds start and now.

**7.111.3.4 init()**

```
void init ( ) [pure virtual]
```

Initialize.

**7.111.3.5 setCCConsumed()**

```
void setCCConsumed (
    uint32_t val ) [inline], [virtual]
```

Sets number of consumed clock cycles.

**Parameters**

<i>val</i>	number of clock cycles.
------------	-------------------------

**7.111.3.6 setMCUActive()**

```
void setMCUActive (
    bool active ) [inline], [virtual]
```

Sets MCU activity high.

**Parameters**

<i>active</i>	if True, inactive otherwise.
---------------	------------------------------

## 7.112 ModalWindow Class Reference

[Container](#) for displaying a modal window and hijacking touch event to underlaying view and widgets.

```
#include <include/gui/common/ModalWindow.hpp>
```

**Public Member Functions**

- [ModalWindow](#) ()  
*Default constructor.*
- virtual [~ModalWindow](#) ()  
*Destructor.*
- virtual void [setBackground](#) (const [BitmapId](#) &bmpId)  
*Sets the background of the actual window.*
- virtual void [setBackground](#) (const [BitmapId](#) &bmpId, int16\_t backgroundX, int16\_t backgroundY)  
*Sets the background of the actual window.*
- virtual uint16\_t [getBackgroundWidth](#) () const

- Gets the width of the actual window (the background images).*
- virtual uint16\_t [getHeight](#) () const  
*Gets the height of the actual window (the background images).*
- virtual void [add](#) ([Drawable](#) &d)  
*Adds a drawable to the [ModalWindow](#).*
- virtual void [remove](#) ([Drawable](#) &d)  
*Removes the drawable from the [ModalWindow](#).*
- virtual void [setShadeAlpha](#) (uint8\_t alpha)  
*Sets the alpha value of the background shade.*
- virtual uint8\_t [getShadeAlpha](#) () const  
*Gets the alpha value of the background shade.*
- virtual void [setShadeColor](#) (colortype color)  
*Sets the color of the background shade.*
- virtual colortype [getShadeColor](#) () const  
*Gets the color of the background shade.*
- virtual void [show](#) ()  
*Make the [ModalWindow](#) visible.*
- virtual void [hide](#) ()  
*Make the [ModalWindow](#) invisible.*
- virtual bool [isShowing](#) () const  
*Query if this [ModalWindow](#) is showing.*

## Protected Attributes

- [Box](#) [backgroundShade](#)  
*The background shade.*
- [Container](#) [windowContainer](#)  
*The window container that defines the active container area where both the [windowBackground](#) and added drawables are placed.*
- [Image](#) [windowBackground](#)  
*The window background.*

## Additional Inherited Members

### 7.112.1 Detailed Description

[Container](#) for displaying a modal window and hijacking touch event to underlying view and widgets. The container has a background image and a surround box that acts as a shade on top of the rest of the screen. The background image must be set (using the [setBackground](#) method) and the shade can be adjusted (using the [setShadeAlpha](#) and [setShadeColor](#) methods).

The [ModalWindow](#) can either be used directly by adding widgets/containers to the [ModalWindow](#) from your view or by sub-classing it if you need a specific [ModalWindow](#) with predefined behavior across your application.

The [ModalWindow](#) should be instantiated in the view class and added as the last element (to always be on top). The [ModalWindow](#) will fill up the entire screen so it should always be placed at x=0, y=0.

To control the visibility of the [ModalWindow](#) use the [show](#) and [hide](#) methods.

### 7.112.2 Constructor & Destructor Documentation



### 7.112.2.1 ModalWindow()

`ModalWindow ( )`

Default constructor.

### 7.112.2.2 ~ModalWindow()

`~ModalWindow ( ) [virtual]`

Destructor.

## 7.112.3 Member Function Documentation

### 7.112.3.1 add()

```
void add (
    Drawable & d ) [virtual]
```

Adds a drawable to the [ModalWindow](#). The drawable will be placed relative to the background image.

#### Parameters

in	<i>d</i>	The drawable to add.
----	----------	----------------------

Reimplemented from [Container](#).

### 7.112.3.2 getBackgroundHeight()

```
uint16_t getBackgroundHeight ( ) const [virtual]
```

Gets the height of the actual window (the background images). Whereas the [getHeight\(\)](#) method will return the height including the shade.

#### Returns

The height of the actual window.

### 7.112.3.3 getBackgroundWidth()

```
uint16_t getBackgroundWidth ( ) const [virtual]
```

Gets the width of the actual window (the background images). Whereas the [getWidth\(\)](#) method will return the width including the shade.

#### Returns

The width of the actual window.

#### 7.112.3.4 getShadeAlpha()

```
uint8_t getShadeAlpha ( ) const [virtual]
```

Gets the alpha value of the background shade.

##### Returns

The background shades alpha.

#### 7.112.3.5 getShadeColor()

```
colortype getShadeColor ( ) const [virtual]
```

Gets the color of the background shade.

##### Returns

The color of the background shade.

#### 7.112.3.6 hide()

```
void hide ( ) [virtual]
```

Make the [ModalWindow](#) invisible.

#### 7.112.3.7 isShowing()

```
bool isShowing ( ) const [virtual]
```

Query if this [ModalWindow](#) is showing.

##### Returns

true if showing, false if not.

#### 7.112.3.8 remove()

```
void remove (
    Drawable & d ) [virtual]
```

Removes the drawable from the [ModalWindow](#).

##### Parameters

in	<i>d</i>	The drawable to remove.
----	----------	-------------------------

Reimplemented from [Container](#).

**7.112.3.9 setBackground()** [1/2]

```
void setBackground (
    const BitmapId & bmpId ) [virtual]
```

Sets the background of the actual window. The remaining area of the screen will be covered by the shade. The background image is centered on the screen.

**Parameters**

<i>bmpId</i>	Identifier for the background bitmap.
--------------	---------------------------------------

**7.112.3.10 setBackground()** [2/2]

```
void setBackground (
    const BitmapId & bmpId,
    int16_t backgroundX,
    int16_t backgroundY ) [virtual]
```

Sets the background of the actual window. The remaining area of the screen will be covered by the shade. The background image will be placed at the backgroundX and backgroundY coordinate.

**Parameters**

<i>bmpId</i>	Identifier for the bitmap.
<i>backgroundX</i>	The background x coordinate.
<i>backgroundY</i>	The background y coordinate.

**7.112.3.11 setShadeAlpha()**

```
void setShadeAlpha (
    uint8_t alpha ) [virtual]
```

Sets the alpha value of the background shade. Default = 96.

**Parameters**

<i>alpha</i>	The new alpha.
--------------	----------------

**7.112.3.12 setShadeColor()**

```
void setShadeColor (
    colortype color ) [virtual]
```

Sets the color of the background shade. Default = Black.

**Parameters**

<i>color</i>	The new color.
--------------	----------------

## 7.112.3.13 show()

```
void show ( ) [virtual]
```

Make the [ModalWindow](#) visible.

7.113 [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.*
- void [clearMoveAnimationEndedAction](#) ()  
*Clears the move animation ended action previously set by setMoveAnimationEndedAction.*
- 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.*
- virtual bool [isMoveAnimationRunning](#) () const  
*Gets whether or not the move animation is running.*
- void [startMoveAnimation](#) (int16\_t endX, int16\_t endY, uint16\_t duration, [EasingEquation](#) xProgression↵  
Equation=&[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.*
- void [nextMoveAnimationStep](#) ()  
*Execute next step in move animation.*

## 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](#).*

### 7.113.1 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

<a href="#">T</a>	Specifies the type should have the move animation capability.
-------------------	---------------------------------------------------------------

### 7.113.2 Constructor & Destructor Documentation

#### 7.113.2.1 MoveAnimator()

```
MoveAnimator ( ) [inline]
```

Default constructor. Creates and initialize the [MoveAnimator](#).

### 7.113.2.2 ~MoveAnimator()

```
~MoveAnimator ( ) [inline], [virtual]
```

Destructor. Destroys the [MoveAnimator](#).

## 7.113.3 Member Function Documentation

### 7.113.3.1 cancelMoveAnimation()

```
void cancelMoveAnimation ( ) [inline]
```

Cancel move animation.

### 7.113.3.2 clearMoveAnimationEndedAction()

```
void clearMoveAnimationEndedAction ( ) [inline]
```

Clears the move animation ended action previously set by [setMoveAnimationEndedAction](#).

See also

[setMoveAnimationEndedAction](#)

### 7.113.3.3 getMoveAnimationDelay()

```
uint16_t getMoveAnimationDelay ( ) const [inline], [virtual]
```

Gets the current animation delay.

Returns

The current animation delay.

### 7.113.3.4 handleTickEvent()

```
void handleTickEvent ( ) [inline], [protected], [virtual]
```

The tick handler that handles the actual animation steps.

### 7.113.3.5 isMoveAnimationRunning()

```
bool isMoveAnimationRunning ( ) const [inline], [virtual]
```

Gets whether or not the move animation is running.

Returns

true if the move animation is running.

## 7.113.3.6 isRunning()

```
bool isRunning ( ) const [inline], [virtual]
```

Gets whether or not the move animation is running.

## Returns

true if the move animation is running.

## 7.113.3.7 nextMoveAnimationStep()

```
void nextMoveAnimationStep ( ) [inline], [protected]
```

Execute next step in move animation and stop the timer if necessary.

## 7.113.3.8 setMoveAnimationDelay()

```
void setMoveAnimationDelay (
    uint16_t delay ) [inline], [virtual]
```

Sets a delay on animations done by the [MoveAnimator](#).

## Parameters

<i>delay</i>	The delay in ticks.
--------------	---------------------

## 7.113.3.9 setMoveAnimationEndedAction()

```
void setMoveAnimationEndedAction (
    GenericCallback< const MoveAnimator< T > & > & callback ) [inline]
```

Associates an action to be performed when the animation ends.

## Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">MoveAnimator</a> .
-----------------	-----------------------------------------------------------------------------------------------------------

## See also

[GenericCallback](#)

## 7.113.3.10 startMoveAnimation()

```
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

<i>endX</i>	The X position of T at animation end. Relative to the container or view that holds T.
<i>endY</i>	The Y position of T at animation end. Relative to the container or view that holds T.
<i>duration</i>	The duration of the animation measured in ticks.
<i>xProgressionEquation</i>	The equation that describes the development of the X position during the animation. Default = <a href="#">EasingEquations::linearEaseNone</a> .
<i>yProgressionEquation</i>	The equation that describes the development of the Y position during the animation. Default = <a href="#">EasingEquations::linearEaseNone</a> .

## 7.114 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

#### 7.114.1 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](#)

## 7.114.2 Constructor & Destructor Documentation

### 7.114.2.1 MVPApplication()

```
MVPApplication ( ) [inline]
```

Default constructor.

### 7.114.2.2 ~MVPApplication()

```
~MVPApplication ( ) [inline], [virtual]
```

Destructor.

## 7.114.3 Member Function Documentation

### 7.114.3.1 evaluatePendingScreenTransition()

```
void evaluatePendingScreenTransition ( ) [inline], [protected]
```

Evaluates the pending [Callback](#) instances. If a callback is valid, it is executed and a [Screen](#) transition is executed.

### 7.114.3.2 handlePendingScreenTransition()

```
void handlePendingScreenTransition ( ) [inline], [virtual]
```

Delegates the work to [evaluatePendingScreenTransition\(\)](#)

Reimplemented from [Application](#).

## 7.115 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.

### 7.115.1 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`.

### 7.115.2 Constructor & Destructor Documentation

#### 7.115.2.1 MVPHeap()

```
MVPHeap (
    AbstractPartition & pres,
    AbstractPartition & scr,
    AbstractPartition & tra,
    MVPApplication & app ) [inline]
```

Constructs an [MVPHeap](#).

#### Parameters

in	<i>pres</i>	A memory partition containing enough memory to hold the largest presenter.
in	<i>scr</i>	A memory partition containing enough memory to hold the largest view.
in	<i>tra</i>	A memory partition containing enough memory to hold the largest transition.
in	<i>app</i>	A reference to the <a href="#">MVPApplication</a> instance.

#### 7.115.2.2 ~MVPHeap()

```
~MVPHeap ( ) [inline], [virtual]
```

Destructor.

## 7.116 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

### 7.116.1 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](#)

### 7.116.2 Constructor & Destructor Documentation

#### 7.116.2.1 NoDMA()

```
NoDMA ( ) [inline]
```

Constructs a [NoDMA](#) object, with a queue of 1 element.

### 7.116.3 Member Function Documentation

#### 7.116.3.1 flush()

```
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](#).

### 7.116.3.2 getBlitCaps()

```
BlitOperations getBlitCaps ( ) [inline], [virtual]
```

No blit operations supported by this DMA implementation.

#### Returns

Zero (no blit ops supported).

Implements [DMA\\_Interface](#).

### 7.116.3.3 setupDataCopy()

```
void setupDataCopy (
    const BlitOp & blitOp ) [inline], [virtual]
```

Asserts if used.

#### Parameters

<i>blitOp</i>	The blit operation to be performed by this DMA instance.
---------------	----------------------------------------------------------

Implements [DMA\\_Interface](#).

### 7.116.3.4 setupDataFill()

```
void setupDataFill (
    const BlitOp & blitOp ) [inline], [virtual]
```

Asserts if used.

#### Parameters

<i>blitOp</i>	The blit operation to be performed by this DMA instance.
---------------	----------------------------------------------------------

Implements [DMA\\_Interface](#).

### 7.116.3.5 signalDMAInterrupt()

```
void signalDMAInterrupt ( ) [inline], [virtual]
```

Does nothing.

Implements [DMA\\_Interface](#).

## 7.117 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.*

### 7.117.1 Detailed Description

Empty [TouchController](#) implementation which does nothing. Use this if your display does not have touch input capabilities.

### 7.117.2 Constructor & Destructor Documentation

#### 7.117.2.1 [~NoTouchController](#)()

```
~NoTouchController ( ) [inline], [virtual]
```

Destructor.

### 7.117.3 Member Function Documentation

#### 7.117.3.1 [init](#)()

```
void init ( ) [inline], [virtual]
```

Empty initialization.

Implements [TouchController](#).

#### 7.117.3.2 [sampleTouch](#)()

```
bool sampleTouch (  
    int32_t & x,  
    int32_t & y ) [inline], [virtual]
```

##### Parameters

out	<i>x</i>	unused.
out	<i>y</i>	unused.

##### Returns

false.

Implements [TouchController](#).

## 7.118 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

#### 7.118.1 Detailed Description

The most simple [Transition](#) without any visual effects.

See also

[Transition](#)

#### 7.118.2 Constructor & Destructor Documentation

##### 7.118.2.1 NoTransition()

```
NoTransition ( ) [inline]
```

Default constructor.

##### 7.118.2.2 ~NoTransition()

```
~NoTransition ( ) [inline], [virtual]
```

Destructor.

#### 7.118.3 Member Function Documentation

##### 7.118.3.1 handleTickEvent()

```
void handleTickEvent ( ) [inline], [virtual]
```

Indicates that the transition is done after the first tick.

Reimplemented from [Transition](#).

## 7.119 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.*
- static void `taskDelay(uint16_t ms)`  
*A function that causes executing task to sleep for a number of milliseconds.*

### 7.119.1 Detailed Description

This class specifies OS wrappers for dealing with the frame buffer semaphore and the VSYNC signal.

### 7.119.2 Member Function Documentation

#### 7.119.2.1 `giveFrameBufferSemaphore()`

```
static void giveFrameBufferSemaphore ( ) [static]
```

Release the frame buffer semaphore.

#### 7.119.2.2 `giveFrameBufferSemaphoreFromISR()`

```
static void giveFrameBufferSemaphoreFromISR ( ) [static]
```

Release the frame buffer semaphore in a way that is safe in interrupt context. Called from ISR.

#### 7.119.2.3 `initialize()`

```
static void initialize ( ) [static]
```

Initialize frame buffer semaphore and queue/mutex for VSYNC signal.

#### 7.119.2.4 signalVSync()

```
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.

#### 7.119.2.5 takeFrameBufferSemaphore()

```
static void takeFrameBufferSemaphore ( ) [static]
```

Take the frame buffer semaphore. Blocks until semaphore is available.

#### 7.119.2.6 taskDelay()

```
static void taskDelay (
    uint16_t ms ) [static]
```

A function that causes executing task to sleep for a number of milliseconds. This function is OPTIONAL. It is only used by the TouchGFX in the case of a specific frame refresh strategy (REFRESH\_STRATEGY\_OPTIM\_SINGLE\_BUFFER\_TFT\_CTRL). Due to backwards compatibility, in order for this function to be useable by the [HAL](#) the function must be explicitly registered: `hal.registerTaskDelayFunction(&OSWrappers::taskDelay)`

##### Parameters

<i>ms</i>	The number of milliseconds to sleep
-----------	-------------------------------------

##### See also

[HAL::setFrameRefreshStrategy\(FrameRefreshStrategy s\)](#)  
[HAL::registerTaskDelayFunction\(void \(\\*delayF\)\(uint16\\_t\)\)](#)

#### 7.119.2.7 tryTakeFrameBufferSemaphore()

```
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 taken the semaphore, it only serves to make sure that the semaphore is taken by someone.

#### 7.119.2.8 waitForVSync()

```
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.

## 7.120 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.*
- virtual [~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.*

#### 7.120.1 Detailed Description

An internal class that implements the main rasterization algorithm. Used in the [Rasterizer](#). Should not be used directly.

## 7.120.2 Member Typedef Documentation

### 7.120.2.1 OutlineFlags\_t

```
unsigned int OutlineFlags_t
```

Defines an alias representing the outline flags.

## 7.120.3 Constructor & Destructor Documentation

### 7.120.3.1 Outline()

```
Outline ( )
```

Default constructor.

### 7.120.3.2 ~Outline()

```
~Outline ( ) [virtual]
```

Destructor.

## 7.120.4 Member Function Documentation

### 7.120.4.1 getCells()

```
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](#).

### 7.120.4.2 getNumCells()

```
unsigned getNumCells ( ) const [inline]
```

Gets number cells registered in the current drawn path for the [Outline](#).

#### Returns

The number of cells.

#### 7.120.4.3 lineTo()

```
void lineTo (
    int x,
    int y )
```

Create a line from the current virtual pen coordinate to the given coordinate creating an [Outline](#).

##### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

#### 7.120.4.4 moveTo()

```
void moveTo (
    int x,
    int y )
```

Move a virtual pen to the specified coordinate.

##### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

#### 7.120.4.5 reset()

```
void reset ( )
```

Resets this object. This implies removing the current [Cell](#) objects and preparing for a new [Outline](#).

#### 7.120.4.6 setMaxRenderY()

```
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 or higher than the given y.

##### Parameters

<i>y</i>	The max y coordinate to render for the <a href="#">Outline</a> .
----------	------------------------------------------------------------------

#### 7.120.4.7 wasOutlineTooComplex()

```
bool wasOutlineTooComplex ( ) [inline]
```

Determines if there was enough memory to register the entire outline, or if the outline was too complex.

### Returns

false if the buffer for [Outline Cell](#) objects was too small.

## 7.121 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](#) ([colortype](#) color)  
*Sets color to use when drawing the [CanvasWidget](#).*
- [colortype](#) [getColor](#) () const  
*Gets the current color.*
- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

### 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.*

#### 7.121.1 Detailed Description

[PainterBW](#) is used for drawing one 1bpp displays. The color is either on or off No transparency is supported.

#### See also

[AbstractPainter](#)

#### 7.121.2 Member Function Documentation

## 7.121.2.1 bw()

```
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

<i>red</i>	The red color.
<i>green</i>	The green color.
<i>blue</i>	The blue color.

## Returns

1 (white) if the brightness of the RGB color is above 50% and 0 (black) otherwise.

## 7.121.2.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

## Returns

The color.

## 7.121.2.3 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterBW](#).

#### 7.121.2.4 renderNext()

```
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](#).

#### 7.121.2.5 setColor()

```
void setColor (
    colortype color )
```

Sets color to use when drawing the [CanvasWidget](#).

##### Parameters

color	The color, 0=black, otherwise white.
-------	--------------------------------------

## 7.122 PainterBWBitmap Class Reference

A Painter that will paint using a bitmap.

```
#include <touchgfx/widgets/canvas/PainterBWBitmap.hpp>
```

### Public Member Functions

- [PainterBWBitmap](#) (const [Bitmap](#) &bmp=[Bitmap](#)([BITMAP\\_INVALID](#)))  
*Constructor.*
- void [setBitmap](#) (const [Bitmap](#) &bmp)  
*Sets a bitmap to be used when drawing the [CanvasWidget](#).*
- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

### Protected Member Functions

- virtual bool [renderInit](#) ()  
*Initialize rendering of a single scan line of pixels for the render.*

- virtual bool [renderNext](#) (uint8\_t &color)  
*Get the color of the next pixel in the scan line.*

## Protected Attributes

- const uint8\_t \* [bitmapBWPointer](#)  
*Pointer to the bitmap (BW)*
- LCD1bpp::bwRLEdata [bw\\_rle](#)  
*Pointer to class for walking through bw\_rle image.*
- [Bitmap](#) [bitmap](#)  
*The bitmap to be used when painting.*
- [Rect](#) [bitmapRectToFrameBuffer](#)  
*Bitmap rectangle translated to frame buffer coordinates.*

### 7.122.1 Detailed Description

[PainterBWBitmap](#) 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](#)

### 7.122.2 Constructor & Destructor Documentation

#### 7.122.2.1 PainterBWBitmap()

```
PainterBWBitmap (
    const Bitmap & bmp = Bitmap(BITMAP\_INVALID) )
```

Constructor.

Parameters

<i>bmp</i>	The bitmap.
------------	-------------

### 7.122.3 Member Function Documentation

#### 7.122.3.1 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterBW](#).

#### 7.122.3.2 renderInit()

```
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 [AbstractPainterBW](#).

#### 7.122.3.3 renderNext()

```
virtual bool renderNext (
    uint8_t & color ) [protected], [virtual]
```

Get the color of the next pixel in the scan line.

#### Parameters

out	<i>color</i>	<a href="#">Color</a> of the pixel, 0 or 1.
-----	--------------	---------------------------------------------

#### Returns

true if the pixel should be painted, false otherwise.

Implements [AbstractPainterBW](#).

#### 7.122.3.4 setBitmap()

```
void setBitmap (
    const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the [CanvasWidget](#).



## Parameters

<i>bmp</i>	The bitmap.
------------	-------------

## 7.123 PainterGRAY2 Class Reference

A Painter that will paint using a color and an alpha value.

```
#include <touchgfx/widgets/canvas/PainterGRAY2.hpp>
```

### Public Member Functions

- [PainterGRAY2](#) ([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](#).*
- [colortype](#) [getColor](#) () const  
*Gets the current color.*
- void [setAlpha](#) ([uint8\\_t](#) alpha)  
*Sets an alpha value for the painter.*
- [uint8\\_t](#) [getAlpha](#) () const  
*Gets the current alpha value.*
- virtual void [render](#) ([uint8\\_t](#) \*ptr, int x, int xAdjust, int y, unsigned count, const [uint8\\_t](#) \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

### Protected Member Functions

- virtual bool [renderNext](#) ([uint8\\_t](#) &gray, [uint8\\_t](#) &alpha)  
*Get the color of the next pixel in the scan line.*

### Protected Attributes

- [uint8\\_t](#) [painterGray](#)  
*The grey color.*
- [uint8\\_t](#) [painterAlpha](#)  
*The alpha value.*

#### 7.123.1 Detailed Description

The [PainterGRAY2](#) 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](#)

#### 7.123.2 Constructor & Destructor Documentation

### 7.123.2.1 PainterGRAY2()

```
PainterGRAY2 (
    colortype color = 0,
    uint8_t alpha = 255 )
```

Constructor.

#### Parameters

<i>color</i>	the color.
<i>alpha</i>	the alpha.

## 7.123.3 Member Function Documentation

### 7.123.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

#### Returns

The current alpha value.

#### See also

[setAlpha](#)

### 7.123.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

#### Returns

The color.

### 7.123.3.3 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterGRAY2](#).

## 7.123.3.4 renderNext()

```
virtual bool renderNext (
    uint8_t & gray,
    uint8_t & alpha ) [protected], [virtual]
```

Get the color of the next pixel in the scan line.

## Parameters

out	<i>gray</i>	The gray (0-15).
out	<i>alpha</i>	The alpha.

## Returns

true if the pixel should be painted, false otherwise.

Implements [AbstractPainterGRAY2](#).

## 7.123.3.5 setAlpha()

```
void setAlpha (
    uint8_t alpha )
```

Sets an alpha value for the painter.

## Parameters

<i>alpha</i>	The alpha value to use.
--------------	-------------------------

## 7.123.3.6 setColor()

```
void setColor (
    colortype color,
    uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the [CanvasWidget](#).

## Parameters

<i>color</i>	The color.
<i>alpha</i>	The alpha.

## 7.124 PainterGRAY2Bitmap Class Reference

A Painter that will paint using a bitmap.

```
#include <touchgfx/widgets/canvas/PainterGRAY2Bitmap.hpp>
```

### Public Member Functions

- [PainterGRAY2Bitmap](#) (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.*
- uint8\_t [getAlpha](#) () const  
*Gets the current alpha value.*
- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

### Protected Member Functions

- virtual bool [renderInit](#) ()  
*Initialize rendering of a single scan line of pixels for the render.*
- virtual bool [renderNext](#) (uint8\_t &gray, uint8\_t &alpha)  
*Get the color of the next pixel in the scan line.*

### Protected Attributes

- const uint8\_t \* [bitmapGRAY2Pointer](#)  
*Pointer to the bitmap (GRAY2)*
- const uint8\_t \* [bitmapAlphaPointer](#)  
*Pointer to the bitmap alpha data for GRAY2.*
- [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.*

### 7.124.1 Detailed Description

[PainterGRAY2Bitmap](#) 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](#)

### 7.124.2 Constructor & Destructor Documentation

#### 7.124.2.1 PainterGRAY2Bitmap()

```
PainterGRAY2Bitmap (
    const Bitmap & bmp = Bitmap(BITMAP\_INVALID),
    uint8_t alpha = 255 )
```

Constructor.

Parameters

<i>bmp</i>	The bitmap.
<i>alpha</i>	the alpha.

### 7.124.3 Member Function Documentation

#### 7.124.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

[setAlpha](#)

#### 7.124.3.2 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
```

```

    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterGRAY2](#).

#### 7.124.3.3 renderInit()

```

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 [AbstractPainterGRAY2](#).

#### 7.124.3.4 renderNext()

```

virtual bool renderNext (
    uint8_t & gray,
    uint8_t & alpha ) [protected], [virtual]

```

Get the color of the next pixel in the scan line.

#### Parameters

out	<i>gray</i>	The gray (0-15).
out	<i>alpha</i>	The alpha.

#### Returns

true if the pixel should be painted, false otherwise.

Implements [AbstractPainterGRAY2](#).

## 7.124.3.5 setAlpha()

```
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

<i>alpha</i>	The alpha value to use if there is no alpha channel in the bitmap.
--------------	--------------------------------------------------------------------

## 7.124.3.6 setBitmap()

```
void setBitmap (
    const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the [CanvasWidget](#).

## Parameters

<i>bmp</i>	The bitmap.
------------	-------------

## 7.125 PainterGRAY4 Class Reference

A Painter that will paint using a color and an alpha value.

```
#include <touchgfx/widgets/canvas/PainterGRAY4.hpp>
```

## Public Member Functions

- [PainterGRAY4](#) ([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](#).*
- [colortype](#) [getColor](#) () const  
*Gets the current color.*
- void [setAlpha](#) (uint8\_t alpha)  
*Sets an alpha value for the painter.*
- uint8\_t [getAlpha](#) () const  
*Gets the current alpha value.*
- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

## Protected Member Functions

- virtual bool [renderNext](#) (uint8\_t &gray, uint8\_t &alpha)  
*Get the color of the next pixel in the scan line.*

## Protected Attributes

- `uint8_t painterGray`  
*The grey color.*
- `uint8_t painterAlpha`  
*The alpha value.*

### 7.125.1 Detailed Description

The [PainterGRAY4](#) 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](#)

### 7.125.2 Constructor & Destructor Documentation

#### 7.125.2.1 PainterGRAY4()

```
PainterGRAY4 (
    colortype color = 0,
    uint8_t alpha = 255 )
```

Constructor.

Parameters

<i>color</i>	the color.
<i>alpha</i>	the alpha.

### 7.125.3 Member Function Documentation

#### 7.125.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

[setAlpha](#)



## 7.125.3.2 getColor()

```
color_t getColor ( ) const
```

Gets the current color.

## Returns

The color.

## 7.125.3.3 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterGRAY4](#).

## 7.125.3.4 renderNext()

```
virtual bool renderNext (
    uint8_t & gray,
    uint8_t & alpha ) [protected], [virtual]
```

Get the color of the next pixel in the scan line.

## Parameters

out	<i>gray</i>	The gray (0-15).
out	<i>alpha</i>	The alpha.

## Returns

true if the pixel should be painted, false otherwise.

Implements [AbstractPainterGRAY4](#).

### 7.125.3.5 setAlpha()

```
void setAlpha (
    uint8_t alpha )
```

Sets an alpha value for the painter.

#### Parameters

<i>alpha</i>	The alpha value to use.
--------------	-------------------------

### 7.125.3.6 setColor()

```
void setColor (
    colortype color,
    uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the [CanvasWidget](#).

#### Parameters

<i>color</i>	The color.
<i>alpha</i>	The alpha.

## 7.126 PainterGRAY4Bitmap Class Reference

A Painter that will paint using a bitmap.

```
#include <touchgfx/widgets/canvas/PainterGRAY4Bitmap.hpp>
```

### Public Member Functions

- [PainterGRAY4Bitmap](#) (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.*
- uint8\_t [getAlpha](#) () const  
*Gets the current alpha value.*
- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

### Protected Member Functions

- virtual bool [renderInit](#) ()  
*Initialize rendering of a single scan line of pixels for the render.*

- virtual bool [renderNext](#) (uint8\_t &gray, uint8\_t &alpha)

*Get the color of the next pixel in the scan line.*

## Protected Attributes

- const uint8\_t \* [bitmapGRAY4Pointer](#)  
*Pointer to the bitmap (GRAY4)*
- const uint8\_t \* [bitmapAlphaPointer](#)  
*Pointer to the bitmap alpha data for GRAY4.*
- [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.*

### 7.126.1 Detailed Description

[PainterGRAY4Bitmap](#) 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](#)

### 7.126.2 Constructor & Destructor Documentation

#### 7.126.2.1 PainterGRAY4Bitmap()

```
PainterGRAY4Bitmap (
    const Bitmap & bmp = Bitmap(BITMAP\_INVALID),
    uint8_t alpha = 255 )
```

Constructor.

#### Parameters

<i>bmp</i>	The bitmap.
<i>alpha</i>	the alpha.

### 7.126.3 Member Function Documentation

#### 7.126.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

#### Returns

The current alpha value.

#### See also

[setAlpha](#)

#### 7.126.3.2 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterGRAY4](#).

#### 7.126.3.3 renderInit()

```
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 [AbstractPainterGRAY4](#).

#### 7.126.3.4 renderNext()

```
virtual bool renderNext (
```

```
uint8_t & gray,
uint8_t & alpha ) [protected], [virtual]
```

Get the color of the next pixel in the scan line.

#### Parameters

out	<i>gray</i>	The gray (0-15).
out	<i>alpha</i>	The alpha.

#### Returns

true if the pixel should be painted, false otherwise.

Implements [AbstractPainterGRAY4](#).

#### 7.126.3.5 setAlpha()

```
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

<i>alpha</i>	The alpha value to use if there is no alpha channel in the bitmap.
--------------	--------------------------------------------------------------------

#### 7.126.3.6 setBitmap()

```
void setBitmap (
    const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the [CanvasWidget](#).

#### Parameters

<i>bmp</i>	The bitmap.
------------	-------------

## 7.127 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](#).*
- `colortype` `getColor` () const  
*Gets the current color.*
- void `setAlpha` (`uint8_t` alpha)  
*Sets an alpha value for the painter.*
- `uint8_t` `getAlpha` () const  
*Gets the current alpha value.*
- virtual void `render` (`uint8_t` \*ptr, int x, int xAdjust, int y, unsigned count, const `uint8_t` \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

### 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

- `uint16_t` `painterColor`  
*The color.*
- `uint16_t` `painterRed`  
*The red part of the color.*
- `uint16_t` `painterGreen`  
*The green part of the color.*
- `uint16_t` `painterBlue`  
*The blue part of the color.*
- `uint8_t` `painterAlpha`  
*The alpha value.*

### Additional Inherited Members

#### 7.127.1 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](#)

#### 7.127.2 Constructor & Destructor Documentation

##### 7.127.2.1 PainterRGB565()

```
PainterRGB565 (
    colortype color = 0,
    uint8_t alpha = 255 )
```

Constructor.

## Parameters

<i>color</i>	the color.
<i>alpha</i>	the alpha.

## 7.127.3 Member Function Documentation

## 7.127.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

## Returns

The current alpha value.

## See also

[setAlpha](#)

## 7.127.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

## Returns

The color.

## 7.127.3.3 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).

## Parameters

	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterRGB565](#).

7.127.3.4 `renderNext()`

```
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	<i>red</i>	The red.
out	<i>green</i>	The green.
out	<i>blue</i>	The blue.
out	<i>alpha</i>	The alpha.

## Returns

true if the pixel should be painted, false otherwise.

Implements [AbstractPainterRGB565](#).

7.127.3.5 `setAlpha()`

```
void setAlpha (
    uint8_t alpha )
```

Sets an alpha value for the painter.

## Parameters

<i>alpha</i>	The alpha value to use.
--------------	-------------------------

7.127.3.6 `setColor()`

```
void setColor (
    colortype color,
    uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the [CanvasWidget](#).



## Parameters

<i>color</i>	The color.
<i>alpha</i>	The alpha.

## 7.128 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.*
- uint8\_t [getAlpha](#) () const  
*Gets the current alpha value.*
- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

### 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.*

## Additional Inherited Members

### 7.128.1 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](#)

### 7.128.2 Constructor & Destructor Documentation

#### 7.128.2.1 PainterRGB565Bitmap()

```
PainterRGB565Bitmap (
    const Bitmap & bmp = Bitmap(BITMAP\_INVALID),
    uint8_t alpha = 255 )
```

Constructor.

Parameters

<i>bmp</i>	The bitmap.
<i>alpha</i>	the alpha.

### 7.128.3 Member Function Documentation

#### 7.128.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

[setAlpha](#)

#### 7.128.3.2 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
```

```

    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels futher).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterRGB565](#).

#### 7.128.3.3 renderInit()

```
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](#).

#### 7.128.3.4 renderNext()

```

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	<i>red</i>	The red.
out	<i>green</i>	The green.
out	<i>blue</i>	The blue.
out	<i>alpha</i>	The alpha.

**Returns**

true if the pixel should be painted, false otherwise.

Implements [AbstractPainterRGB565](#).

**7.128.3.5 setAlpha()**

```
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**

<i>alpha</i>	The alpha value to use if there is no alpha channel in the bitmap.
--------------	--------------------------------------------------------------------

**7.128.3.6 setBitmap()**

```
void setBitmap (
    const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the [CanvasWidget](#).

**Parameters**

<i>bmp</i>	The bitmap.
------------	-------------

**7.129 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](#).*
- [colortype](#) [getColor](#) () const  
*Gets the current color.*
- void [setAlpha](#) (uint8\_t alpha)  
*Sets an alpha value for the painter.*
- uint8\_t [getAlpha](#) () const  
*Gets the current alpha value.*
- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

## 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.*

### 7.129.1 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](#)

### 7.129.2 Constructor & Destructor Documentation

#### 7.129.2.1 PainterRGB888()

```
PainterRGB888 (
    color_t color = 0,
    uint8_t alpha = 255 )
```

Constructor.

Parameters

<i>color</i>	the color.
<i>alpha</i>	the alpha.

### 7.129.3 Member Function Documentation

#### 7.129.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

**Returns**

The current alpha value.

**See also**

[setAlpha](#)

**7.129.3.2 getColor()**

```
colortype getColor ( ) const
```

Gets the current color.

**Returns**

The color.

**7.129.3.3 render()**

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels further).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterRGB888](#).

**7.129.3.4 renderNext()**

```
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	<i>red</i>	The red.
out	<i>green</i>	The green.
out	<i>blue</i>	The blue.
out	<i>alpha</i>	The alpha.

#### Returns

true if the pixel should be painted, false otherwise.

Implements [AbstractPainterRGB888](#).

#### 7.129.3.5 setAlpha()

```
void setAlpha (
    uint8_t alpha )
```

Sets an alpha value for the painter.

#### Parameters

<i>alpha</i>	The alpha value to use.
--------------	-------------------------

#### 7.129.3.6 setColor()

```
void setColor (
    colortype color,
    uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the [CanvasWidget](#).

#### Parameters

<i>color</i>	The color.
<i>alpha</i>	The alpha.

## 7.130 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.*
- uint8\_t [getAlpha](#) () const  
*Gets the current alpha value.*
- virtual void [render](#) (uint8\_t \*ptr, int x, int xAdjust, int y, unsigned count, const uint8\_t \*covers)  
*Paint a designated part of the [RenderingBuffer](#).*

## 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 combined with alpha data from the bitmap.*

### 7.130.1 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](#)

### 7.130.2 Constructor & Destructor Documentation

#### 7.130.2.1 PainterRGB888Bitmap()

```
PainterRGB888Bitmap (
    const Bitmap & bmp = Bitmap(BITMAP\_INVALID),
    uint8_t alpha = 255 )
```

Constructor.



## Parameters

<i>bmp</i>	The bitmap.
<i>alpha</i>	the alpha.

## 7.130.3 Member Function Documentation

## 7.130.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

## Returns

The current alpha value.

## See also

[setAlpha](#)

## 7.130.3.2 render()

```
virtual void render (
    uint8_t * ptr,
    int x,
    int xAdjust,
    int y,
    unsigned count,
    const uint8_t * covers ) [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	<i>ptr</i>	Pointer to the row in the <a href="#">RenderingBuffer</a> .
	<i>x</i>	The x coordinate.
	<i>xAdjust</i>	The minor adjustment of x (used when a pixel is smaller than a byte to specify that the pointer should have been advanced "xAdjust" pixels futher).
	<i>y</i>	The y coordinate.
	<i>count</i>	Number of pixels to fill.
	<i>covers</i>	The coverage in of each pixel.

Reimplemented from [AbstractPainterRGB888](#).

### 7.130.3.3 renderInit()

```
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](#).

### 7.130.3.4 renderNext()

```
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	<i>red</i>	The red.
out	<i>green</i>	The green.
out	<i>blue</i>	The blue.
out	<i>alpha</i>	The alpha.

#### Returns

true if the pixel should be painted, false otherwise.

Implements [AbstractPainterRGB888](#).

### 7.130.3.5 setAlpha()

```
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

<i>alpha</i>	The alpha value to use if there is no alpha channel in the bitmap.
--------------	--------------------------------------------------------------------

### 7.130.3.6 setBitmap()

```
void setBitmap (
    const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the [CanvasWidget](#).

#### Parameters

<i>bmp</i>	The bitmap.
------------	-------------

## 7.131 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.*

### 7.131.1 Detailed Description

```
template<class T1, class T2>
struct touchgfx::Pair< T1, T2 >
```

A simple struct for holding pairs of data.

#### Template Parameters

<i>T1</i>	The type of the first element.
<i>T2</i>	The type of the second element.

### 7.131.2 Constructor & Destructor Documentation

#### 7.131.2.1 Pair() [1/3]

```
Pair ( ) [inline]
```

Constructor initializing the elements it holds using their default constructors.

#### 7.131.2.2 `Pair()` [2/3]

```
Pair (
    const T1 & x,
    const T2 & y ) [inline]
```

Constructor initializing the elements it holds using their copy constructor.

##### Parameters

<code>x</code>	Reference to the first element.
<code>y</code>	Reference to the second element.

#### 7.131.2.3 `Pair()` [3/3]

```
Pair (
    const Pair< U, V > & p ) [inline]
```

Copy constructor.

##### Template Parameters

<code>U</code>	Generic type parameter.
<code>V</code>	Generic type parameter.

##### Parameters

<code>p</code>	The pair to copy from.
----------------	------------------------

## 7.132 `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.*

### 7.132.1 Detailed Description

```
template<typename ListOfTypes, uint16_t NUMBER_OF_ELEMENTS>
class touchgfx::Partition< ListOfTypes, NUMBER_OF_ELEMENTS >
```

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

<i>ListOfTypes</i>	Type of the list of types.
<i>NUMBER_OF_ELEMENTS</i>	Type of the number of elements.

See also

[AbstractPartition](#)

### 7.132.2 Member Typedef Documentation

#### 7.132.2.1 SupportedTypesList

```
ListOfTypes SupportedTypesList
```

Provides a generic public type containing the list of supported types.

### 7.132.3 Member Enumeration Documentation

#### 7.132.3.1 anonymous enum

```
anonymous enum
```

Compile-time generated constants specifying the "element" or "slot" size used by this partition

## 7.132.4 Constructor & Destructor Documentation

### 7.132.4.1 Partition()

```
Partition ( ) [inline]
```

Constructs an empty [Partition](#).

## 7.132.5 Member Function Documentation

### 7.132.5.1 capacity()

```
uint16_t capacity ( ) const [inline], [virtual]
```

Specialization of [AbstractPartition::capacity\(\)](#).

#### Returns

An uint16\_t.

#### See also

[touchgfx::AbstractPartition::capacity\(\)](#)

Implements [AbstractPartition](#).

### 7.132.5.2 element() [1/2]

```
void * element (
    uint16_t index ) [inline], [protected], [virtual]
```

#### Parameters

<i>index</i>	Zero-based index of the.
--------------	--------------------------

#### Returns

null if it fails, else a void\*.

#### See also

[touchgfx::AbstractPartition::element\(\)](#)

Implements [AbstractPartition](#).

### 7.132.5.3 element() [2/2]

```
const void * element (
    uint16_t index ) const [inline], [protected], [virtual]
```

## Parameters

<i>index</i>	Zero-based index of the.
--------------	--------------------------

## Returns

null if it fails, else a void\*.

## See also

[touchgfx::AbstractPartition::element\(\)](#)

Implements [AbstractPartition](#).

## 7.132.5.4 element\_size()

```
uint32_t element_size ( ) [inline], [virtual]
```

Specialization of [AbstractPartition::element\\_size\(\)](#).

## Returns

An uint32\_t.

## See also

[touchgfx::AbstractPartition::element\\_size\(\)](#)

Implements [AbstractPartition](#).

## 7.133 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 a)  
*Sets the alpha channel for the image.*
- uint8\_t [getAlpha](#) () const  
*Gets the current alpha value.*

## 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

### 7.133.1 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](#)

### 7.133.2 Constructor & Destructor Documentation

#### 7.133.2.1 PixelDataWidget()

```
PixelDataWidget ( )
```

Default constructor.

### 7.133.3 Member Function Documentation

#### 7.133.3.1 draw()

```
void draw (
    const touchgfx::Rect & invalidatedArea ) const [virtual]
```

Draw the part of the RAM buffer that is inside the invalidated area.

Parameters

<code>invalidatedArea</code>	The region of this drawable that needs to be redrawn.
------------------------------	-------------------------------------------------------

See also

[touchgfx::Drawable](#)

Implements [Drawable](#).



### 7.133.3.2 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

#### Returns

The current alpha value.

#### See also

[setAlpha](#)

### 7.133.3.3 getSolidRect()

```
touchgfx::Rect getSolidRect ( ) const [virtual]
```

Report this widget as being completely solid.

#### Returns

The solid rect.

Implements [Drawable](#).

### 7.133.3.4 setAlpha()

```
void setAlpha (
    uint8_t a )
```

Sets the alpha channel for the image.

#### Parameters

<i>a</i>	The alpha value. 255 = completely solid.
----------	------------------------------------------

### 7.133.3.5 setBitmapFormat()

```
void setBitmapFormat (
    Bitmap::BitmapFormat format )
```

Set the format of the pixel data.

#### Parameters

<i>format</i>	Describes the format to use.
---------------	------------------------------

### 7.133.3.6 setPixelData()

```
void setPixelData (
    uint8_t *const data )
```

Set the pixel data to display.

#### Parameters

in, out	data	If non-null, the data.
---------	------	------------------------

## 7.134 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.*

## 7.134.1 Member Function Documentation

### 7.134.1.1 dist\_sqr()

```
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.

## 7.135 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.*

## 7.136 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

#### 7.136.1 Detailed Description

This class represents a homogeneous 3D point.

See also

quadruple

#### 7.136.2 Constructor & Destructor Documentation

##### 7.136.2.1 [Point4\(\)](#) [1/2]

```
Point4 ( ) [inline]
```

Default constructor.

### 7.136.2.2 Point4() [2/2]

```
Point4 (
    float x,
    float y,
    float z ) [inline]
```

Constructor.

#### Parameters

x	The x value.
y	The y value.
z	The z value.

## 7.137 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.*

### 7.137.1 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 <a href="#">Drawable</a> to add this functionality to.
---	--------------------------------------------------------------------

## 7.137.2 Constructor & Destructor Documentation

### 7.137.2.1 PreRenderable()

```
PreRenderable ( ) [inline]
```

Default constructor. Initializes the [PreRenderable](#).

## 7.137.3 Member Function Documentation

### 7.137.3.1 draw()

```
void draw (
    const Rect & invalidatedArea ) const [inline]
```

Overrides the draw function. If preRender has been called, perform a memcopy of the cached version. If not, just call the base class version of draw.

#### Parameters

<i>invalidatedArea</i>	The subregion of this <a href="#">Drawable</a> which needs to be redrawn.
------------------------	---------------------------------------------------------------------------

### 7.137.3.2 isPreRendered()

```
bool isPreRendered ( ) const [inline]
```

#### Returns

Is the widget rendered.

### 7.137.3.3 preRender()

```
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.

### 7.137.3.4 setupDrawChain()

```
void setupDrawChain (
    const Rect & invalidatedArea,
    Drawable ** nextPreviousElement ) [inline], [virtual]
```

#### Note

For TouchGFX internal use only.

## Parameters

	<i>invalidatedArea</i>	Include drawables that intersect with this area only.
<i>in, out</i>	<i>nextPreviousElement</i>	Modifiable element in linked list.

## 7.138 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.*

#### 7.138.1 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.

#### 7.138.2 Constructor & Destructor Documentation

##### 7.138.2.1 [~Presenter\(\)](#)

```
~Presenter ( ) [inline], [virtual]
```

Destructor.

##### 7.138.2.2 [Presenter\(\)](#)

```
Presenter ( ) [inline], [protected]
```

Default constructor.

#### 7.138.3 Member Function Documentation

## 7.138.3.1 activate()

```
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.

## 7.138.3.2 deactivate()

```
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.

## 7.139 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.*

## 7.139.1 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/Q_%28number_format%29)

[http://en.wikipedia.org/wiki/Fixed-point\\_arithmetic](http://en.wikipedia.org/wiki/Fixed-point_arithmetic)

## 7.139.2 Constructor &amp; Destructor Documentation

**7.139.2.1 Q10()** [1/2]

```
Q10 ( ) [inline]
```

Default constructor.

**7.139.2.2 Q10()** [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>i</i>	int pre-formatted in <b>Q10</b> format.
----------	-----------------------------------------

**7.139.3 Member Function Documentation****7.139.3.1 operator int()**

```
operator int ( ) const [inline]
```

Gets the **Q10** as an integer without conversion.

**Returns**

The unconverted **Q10** value.

**7.139.3.2 operator\*()**

```
Q15 operator* (
    const Q5 & q5 ) const [inline]
```

Multiplication operator. The result is a **Q15**, not a **Q10**, for increased precision.

**Parameters**

<i>q5</i>	The <b>Q5</b> to multiply this with.
-----------	--------------------------------------

**Returns**

The result of the operation.

**7.139.3.3 operator+()**

```
Q10 operator+ (
    const Q10 & q10 ) const [inline]
```



Addition operator.

#### Parameters

<code>q10</code>	The <a href="#">Q10</a> to add to this.
------------------	-----------------------------------------

#### Returns

The result of the operation.

#### 7.139.3.4 `operator-()`

```
Q10 operator- ( ) const [inline]
```

Negation operator.

#### Returns

The negative value of this.

#### 7.139.3.5 `operator/()`

```
Q5 operator/ (
    const Q5 & q5 ) const [inline]
```

Division operator.

#### Parameters

<code>q5</code>	The <a href="#">Q5</a> to divide this by.
-----------------	-------------------------------------------

#### Returns

The result of the operation.

## 7.140 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.*
- [Q15 operator-](#) () const  
*Negation operator.*

- **Q15 operator+** (const **Q15** &q15) const  
*Addition operator.*
- **Q10 operator/** (const **Q5** &q5) const  
*Q5 / Q5 which requires the result of a Q15 / Q5 to be calculated.*

### 7.140.1 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/Q_%28number_format%29)

[http://en.wikipedia.org/wiki/Fixed-point\\_arithmetic](http://en.wikipedia.org/wiki/Fixed-point_arithmetic)

### 7.140.2 Constructor & Destructor Documentation

#### 7.140.2.1 Q15()

```
Q15 (
    int i ) [inline], [explicit]
```

Constructor from integer. No conversion is done - the integer is assumed to already be in **Q15** format.

Parameters

<i>i</i>	int pre-formatted in <b>Q15</b> format.
----------	-----------------------------------------

### 7.140.3 Member Function Documentation

#### 7.140.3.1 operator int()

```
operator int ( ) const [inline]
```

Gets the **Q15** as an integer without conversion.

Returns

The unconverted **Q15** value.

#### 7.140.3.2 operator+()

```
Q15 operator+ (
    const Q15 & q15 ) const [inline]
```

Addition operator.

#### Parameters

<code>q15</code>	The <code>Q10</code> to add to this.
------------------	--------------------------------------

#### Returns

The result of the operation.

#### 7.140.3.3 operator-()

```
Q15 operator- ( ) const [inline]
```

Negation operator.

#### Returns

The negative value of this.

#### 7.140.3.4 operator/()

```
Q10 operator/ (
    const Q5 & q5 ) const [inline]
```

`Q5 / Q5` which requires the result of a `Q15 / Q5` to be calculated.

#### Parameters

<code>q5</code>	The <code>Q5</code> to divide this by.
-----------------	----------------------------------------

#### Returns

The result of the operation.

## 7.141 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<typename T >`  
`T to () const`  
*Converts the Q5 value to an int or a float.*

### 7.141.1 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\\_number\\_format](http://en.wikipedia.org/wiki/Q_number_format)

[http://en.wikipedia.org/wiki/Fixed-point\\_arithmetic](http://en.wikipedia.org/wiki/Fixed-point_arithmetic)

### 7.141.2 Constructor & Destructor Documentation

#### 7.141.2.1 Q5() [1/2]

```
Q5 (
    int i ) [inline], [explicit]
```

Constructor from integer. No conversion is done - the integer is assumed to already be in Q5 format.

Parameters

<i>i</i>	Integer pre-formatted in Q5 format.
----------	-------------------------------------

## 7.141.2.2 Q5() [2/2]

```
Q5 (
    const Q10 q10 ) [inline]
```

Constructor from Q10. The Q10 is shifted down to convert it to Q5.

## Parameters

<i>q10</i>	The Q10 value to convert to a Q5 value.
------------	-----------------------------------------

## See also

Q10

## 7.141.3 Member Function Documentation

## 7.141.3.1 operator int()

```
operator int ( ) const [inline]
```

Gets the Q5 as an integer without conversion.

## Returns

The unconverted Q5 value.

## 7.141.3.2 operator\*() [1/3]

```
Q10 operator* (
    const Q5 & q5 ) const [inline]
```

Multiplication operator. The result is a Q10, not a Q5, for increased precision.

## Parameters

<i>q5</i>	The Q5 to multiply this with.
-----------	-------------------------------

## Returns

The result of the operation.

## See also

Q10

## 7.141.3.3 operator\*() [2/3]

```
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

<i>q15</i>	The <a href="#">Q15</a> to multiply this with.
------------	------------------------------------------------

#### Returns

The result of the operation.

#### See also

[Q15](#)

#### 7.141.3.4 operator\*() [3/3]

```
Q5 operator* (
    const int i ) const [inline]
```

Multiplication operator.

#### Parameters

<i>i</i>	The integer to multiply this with.
----------	------------------------------------

#### Returns

The result of the operation.

#### 7.141.3.5 operator+()

```
Q5 operator+ (
    const Q5 & q5 ) const [inline]
```

Addition operator.

#### Parameters

<i>q5</i>	The <a href="#">Q5</a> to add to this.
-----------	----------------------------------------

#### Returns

The result of the operation.

#### 7.141.3.6 operator-() [1/2]

```
Q5 operator- ( ) const [inline]
```

Negation operator.

#### Returns

The negative value of this.

#### 7.141.3.7 operator-() [2/2]

```
Q5 operator- (
    const Q5 & q5 ) const [inline]
```

Subtraction operator.

#### Parameters

<i>q5</i>	The Q5 to subtract from this.
-----------	-------------------------------

#### Returns

The result of the operation.

#### 7.141.3.8 operator/() [1/2]

```
Q5 operator/ (
    const int i ) const [inline]
```

Division operator.

#### Parameters

<i>i</i>	The integer to divide this by.
----------	--------------------------------

#### Returns

The result of the operation.

#### 7.141.3.9 operator/() [2/2]

```
Q5 operator/ (
    const Q5 q5 ) const [inline]
```

Division operator. Internally this Q5 is converted to Q10 before the division to increased precision.

#### Parameters

<i>q5</i>	The Q5 to divide this by.
-----------	---------------------------

**Returns**

The result of the operation.

**See also**

[Q10](#)

**7.141.3.10 to()**

```
template< typename 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**

<i>T</i>	Either int or float.
----------	----------------------

**Returns**

[Q5](#) value as a type T.

**7.142 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].*

### 7.142.1 Detailed Description

Base class for homogeneous vectors and points.

### 7.142.2 Constructor & Destructor Documentation

#### 7.142.2.1 [Quadruple\(\)](#) [1/2]

[Quadruple](#) ( ) [inline], [protected]

Default constructor. < The elements[ 3]

#### 7.142.2.2 [Quadruple\(\)](#) [2/2]

```
Quadruple (
    float x,
    float y,
    float z,
    float w ) [inline], [protected]
```

Constructor.

#### Parameters

<i>x</i>	The x value.
<i>y</i>	The y value.
<i>z</i>	The z value.
<i>w</i>	The w value.

### 7.142.3 Member Function Documentation

#### 7.142.3.1 [getElement\(\)](#)

```
float getElement (
    int row ) const [inline]
```

Gets an element.

#### Parameters

<i>row</i>	The row.
------------	----------

#### Returns

The element.

#### 7.142.3.2 getW()

```
float getW ( ) const [inline]
```

Gets the w.

#### Returns

The w.

#### 7.142.3.3 getX()

```
float getX ( ) const [inline]
```

Get x coordinate.

#### Returns

The x coordinate.

#### 7.142.3.4 getY()

```
float getY ( ) const [inline]
```

Get y coordinate.

#### Returns

The y coordinate.

#### 7.142.3.5 getZ()

```
float getZ ( ) const [inline]
```

Get z coordinate.

#### Returns

The z coordinate.

### 7.142.3.6 setElement()

```
void setElement (
    int row,
    float value ) [inline]
```

Sets an element.

#### Parameters

<i>row</i>	The row.
<i>value</i>	The value.

### 7.142.3.7 setW()

```
void setW (
    float value ) [inline]
```

Sets a w.

#### Parameters

<i>value</i>	The value.
--------------	------------

### 7.142.3.8 setX()

```
void setX (
    float value ) [inline]
```

Sets an x coordinate.

#### Parameters

<i>value</i>	The value.
--------------	------------

### 7.142.3.9 setY()

```
void setY (
    float value ) [inline]
```

Sets a y coordinate.

#### Parameters

<i>value</i>	The value.
--------------	------------

### 7.142.3.10 setZ()

```
void setZ (
    float value ) [inline]
```

Sets a z coordinate.

#### Parameters

<i>value</i>	The value.
--------------	------------

## 7.143 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

### 7.143.1 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](#)

### 7.143.2 Constructor & Destructor Documentation

#### 7.143.2.1 RadioButton()

```
RadioButton ( ) [inline]
```

Default constructor.

#### 7.143.2.2 ~RadioButton()

```
~RadioButton ( ) [inline], [virtual]
```

Destructor.

### 7.143.3 Member Function Documentation

#### 7.143.3.1 draw()

```
void draw (
    const Rect & invalidatedArea ) const [virtual]
```

Draws the given invalidated area.

##### Parameters

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

##### See also

[Drawable::draw\(\)](#)

Implements [Drawable](#).

#### 7.143.3.2 getAlpha()

```
uint8_t getAlpha ( ) const [inline]
```

Gets the current alpha value.

##### Returns

The current alpha value.

#### 7.143.3.3 getCurrentlyDisplayedBitmap()

```
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.

#### 7.143.3.4 getDeselectionEnabled()

```
bool getDeselectionEnabled ( ) const [inline]
```

Gets the current deselectionEnabled state.

##### Returns

The current deselectionEnabled state.

#### 7.143.3.5 getSelected()

```
bool getSelected ( ) const [inline]
```

Gets the current selected state.

##### Returns

The current selected state.

#### 7.143.3.6 getSolidRect()

```
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](#).

#### 7.143.3.7 getType()

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

##### Returns

TYPE\_RADIOBUTTON.

Reimplemented from [AbstractButton](#).

#### 7.143.3.8 handleClickEvent()

```
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

<i>event</i>	Information about the click.
--------------	------------------------------

##### See also

[Drawable::handleClickEvent\(\)](#)

Reimplemented from [AbstractButton](#).

### 7.143.3.9 setAlpha()

```
void setAlpha (
    uint8_t alpha ) [inline]
```

Sets the alpha channel for the image.

#### Parameters

<i>alpha</i>	The alpha value. 255 = completely solid.
--------------	------------------------------------------

### 7.143.3.10 setBitmaps()

```
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

<i>bmpUnselected</i>	<a href="#">Bitmap</a> to use when button is unselected.
<i>bmpUnselectedPressed</i>	<a href="#">Bitmap</a> to use when button is unselected and pressed.
<i>bmpSelected</i>	<a href="#">Bitmap</a> to use when button is selected.
<i>bmpSelectedPressed</i>	<a href="#">Bitmap</a> to use when button is selected and pressed.

### 7.143.3.11 setDeselectedAction()

```
void setDeselectedAction (
    GenericCallback< const AbstractButton & > & callback ) [inline]
```

Associates an action to be performed when the [AbstractButton](#) is deselected.

#### Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">AbstractButton</a> .
-----------------	-------------------------------------------------------------------------------------------------------------

#### See also

[GenericCallback](#)

### 7.143.3.12 setDeselectionEnabled()

```
void setDeselectionEnabled (
    bool state ) [inline]
```



States whether or not it is possible to de-select the [RadioButton](#) by clicking it.

## Parameters

<code>state</code>	true if it should be possible to de-select by click.
--------------------	------------------------------------------------------

## 7.143.3.13 setSelected()

```
void setSelected (
    bool newSelected )
```

Sets the radio buttons selected state.

## Parameters

<code>newSelected</code>	the new selected state.
--------------------------	-------------------------

## 7.144 RadioButtonGroup&lt; CAPACITY &gt; 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.*

### 7.144.1 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

<i>CAPACITY</i>	Type of the capacity.
-----------------	-----------------------

See also

[RadioButton](#)

### 7.144.2 Constructor & Destructor Documentation

#### 7.144.2.1 RadioButtonGroup()

```
RadioButtonGroup ( ) [inline]
```

Default constructor.

### 7.144.2.2 ~RadioButtonGroup()

```
~RadioButtonGroup ( ) [inline], [virtual]
```

Destructor.

## 7.144.3 Member Function Documentation

### 7.144.3.1 add()

```
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	<i>radioButton</i>	the <a href="#">RadioButton</a> that is to be added.
----	--------------------	------------------------------------------------------

### 7.144.3.2 getDeselectionEnabled()

```
bool getDeselectionEnabled ( ) const [inline], [virtual]
```

Gets the current deselectionEnabled state.

#### Returns

The current deselectionEnabled state.

### 7.144.3.3 getRadioButton()

```
RadioButton * getRadioButton (
    uint16_t index ) const [inline], [virtual]
```

Gets the [RadioButton](#) at the specified index.

#### Parameters

<i>index</i>	the index of the <a href="#">RadioButton</a> to return.
--------------	---------------------------------------------------------

#### Returns

the [RadioButton](#) at the specified index. Returns 0 if illegal index.

### 7.144.3.4 getSelectedRadioButton()

```
RadioButton * getSelectedRadioButton ( ) const [inline], [virtual]
```

Gets the selected [RadioButton](#).

#### Returns

a pointer to the selected [RadioButton](#). Returns 0 if no [RadioButton](#) is selected.

#### 7.144.3.5 getSelectedRadioButtonIndex()

```
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.

#### 7.144.3.6 radioButtonClickedHandler()

```
void radioButtonClickedHandler (
    const AbstractButton & radioButton ) [inline], [protected], [virtual]
```

Handles the event that a [RadioButton](#) has been selected. deselects all other RadioButtons.

#### Parameters

<i>radioButton</i>	the <a href="#">RadioButton</a> that has been selected.
--------------------	---------------------------------------------------------

#### 7.144.3.7 radioButtonDeselectedHandler()

```
void radioButtonDeselectedHandler (
    const AbstractButton & radioButton ) [inline], [protected], [virtual]
```

Handles the event that a [RadioButton](#) has been deselected.

#### Parameters

<i>radioButton</i>	the <a href="#">RadioButton</a> that has been deselected.
--------------------	-----------------------------------------------------------

#### 7.144.3.8 setDeselectionEnabled()

```
void setDeselectionEnabled (
    bool deselectionEnabled ) [inline], [virtual]
```

Sets whether or not it is possible to deselect RadioButtons by clicking them when they are selected.

#### Parameters

<i>deselectionEnabled</i>	true if it should be possible to deselect by click.
---------------------------	-----------------------------------------------------

#### 7.144.3.9 `setRadioButtonDeselectedHandler()`

```
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

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">RadioButton</a> that was selected.
-----------------	---------------------------------------------------------------------------------------------------------------------------

##### See also

[GenericCallback](#)

#### 7.144.3.10 `setRadioButtonSelectedHandler()`

```
void setRadioButtonSelectedHandler (
    GenericCallback< const AbstractButton & > & callback ) [inline]
```

Associates an action to be performed when a radio button belonging to this group is selected.

##### Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">RadioButton</a> that was selected.
-----------------	---------------------------------------------------------------------------------------------------------------------------

##### See also

[GenericCallback](#)

#### 7.144.3.11 `setSelected()`

```
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	<i>radioButton</i>	the <a href="#">RadioButton</a> to be selected.
----	--------------------	-------------------------------------------------

## 7.145 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.*
- bool [wasOutlineTooComplex](#) ()
- Determines if we the outline was too complex to draw completely.*

#### 7.145.1 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\\_BASE\\_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".

## 7.145.2 Member Enumeration Documentation

### 7.145.2.1 anonymous enum

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.

### 7.145.2.2 anonymous enum

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.

### 7.145.2.3 FillingRule

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.



### 7.145.3 Constructor & Destructor Documentation

#### 7.145.3.1 Rasterizer()

```
Rasterizer ( ) [inline]
```

Default constructor.

### 7.145.4 Member Function Documentation

#### 7.145.4.1 calculateAlpha()

```
unsigned calculateAlpha (
    int area ) const [inline]
```

Calculates the alpha.

##### Parameters

<i>area</i>	The area.
-------------	-----------

##### Returns

The calculated alpha.

#### 7.145.4.2 lineTo()

```
void lineTo (
    int x,
    int y ) [inline]
```

Line to.

##### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

#### 7.145.4.3 moveTo()

```
void moveTo (
    int x,
    int y ) [inline]
```

Move to.

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

**7.145.4.4 render()**

```
template< class Renderer > bool render (
    Renderer & r ) [inline]
```

Renders this object.

**Template Parameters**

<a href="#">Renderer</a>	Type of the renderer.
--------------------------	-----------------------

**Parameters**

in	<i>r</i>	The <a href="#">Renderer</a> 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.

**7.145.4.5 reset()**

```
void reset ( ) [inline]
```

Resets this object. Basically this is done by resetting the the [Outline](#).

**7.145.4.6 setFillingRule()**

```
void setFillingRule (
    FillingRule fillingRule ) [inline]
```

Sets the filling rule to be used when rendering the outline.

**Parameters**

<i>fillingRule</i>	The filling rule.
--------------------	-------------------

**7.145.4.7 setMaxRenderY()**

```
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 or higher than the given y.

#### Parameters

y	The max y coordinate to render for the <a href="#">Outline</a> .
---	------------------------------------------------------------------

#### 7.145.4.8 wasOutlineTooComplex()

```
bool wasOutlineTooComplex ( ) [inline]
```

Determines if we the outline was too complex to draw completely.

#### Returns

True if it was too complex, false if not.

## 7.146 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.*

## 7.146.1 Detailed Description

Class representing a Rectangle with a few convenient methods. Size: 8 bytes.

## 7.146.2 Constructor & Destructor Documentation

### 7.146.2.1 `Rect()` [1/2]

`Rect ( )` [inline]

Default constructor. Resulting in an empty `Rect` with coordinates 0,0.

### 7.146.2.2 `Rect()` [2/2]

```
Rect (
    int16_t x,
    int16_t y,
    int16_t width,
    int16_t height ) [inline]
```

Constructor.

#### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>width</i>	The width.
<i>height</i>	The height.

## 7.146.3 Member Function Documentation

### 7.146.3.1 area()

```
uint32_t area ( ) const [inline]
```

Calculate the area of the rectangle.

#### Returns

area of the rectangle.

### 7.146.3.2 bottom()

```
int16_t bottom ( ) const [inline]
```

Gets the y coordinate of the bottom edge of the [Rect](#).

#### Returns

y coordinate of the bottom edge.

### 7.146.3.3 expandToFit()

```
void expandToFit (
    const Rect & other ) [inline]
```

Increases the area covered by this rectangle to encompass the area covered by supplied rectangle.

#### Parameters

<i>other</i>	The other rectangle.
--------------	----------------------

### 7.146.3.4 includes()

```
bool includes (
    const Rect & other ) const [inline]
```

Determines whether the specified rectangle is completely included in this rectangle.

#### Parameters

<i>other</i>	The other rectangle.
--------------	----------------------

#### Returns

true if the specified rectangle is completely included.

### 7.146.3.5 intersect() [1/2]

```
bool intersect (
```

```
int16_t otherX,  
int16_t otherY ) const [inline]
```

Determines whether specified point lies inside this rectangle.

#### Parameters

<i>otherX</i>	The x coordinate of the point.
<i>otherY</i>	The y coordinate of the point.

#### Returns

true if point lies inside rectangle.

#### 7.146.3.6 intersect() [2/2]

```
bool intersect (  
    const Rect & other ) const [inline]
```

Determines whether specified rectangle intersects with this rectangle.

#### Parameters

<i>other</i>	The other rectangle.
--------------	----------------------

#### Returns

true if the two rectangles intersect.

#### 7.146.3.7 isEmpty()

```
bool isEmpty ( ) const [inline]
```

Query if this object is empty.

#### Returns

true if any of the dimensions are 0.

#### 7.146.3.8 operator!=( )

```
bool operator!= (  
    const Rect & other ) const [inline]
```

Opposite of the == operator.

#### Parameters

<i>other</i>	The Rect to compare with.
--------------	---------------------------

**Returns**

true if the compared [Rect](#) differ in dimensions or coordinates.

**7.146.3.9 operator&()**

```
Rect operator & (
    const Rect & other ) const [inline]
```

Gets a rectangle describing the intersecting area between this rectangle and the supplied rectangle.

**Parameters**

<i>other</i>	The other rectangle.
--------------	----------------------

**Returns**

Intersecting rectangle or Rect(0, 0, 0, 0) in case of no intersection.

**7.146.3.10 operator&=()**

```
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**

<i>other</i>	The rect to intersect with.
--------------	-----------------------------

**7.146.3.11 operator==()**

```
bool operator== (
    const Rect & other ) const [inline]
```

Compares equality of two [Rect](#) by the dimensions and position of these.

**Parameters**

<i>other</i>	The <a href="#">Rect</a> to compare with.
--------------	-------------------------------------------

**Returns**

true if the compared [Rect](#) have the same dimensions and coordinates.

### 7.146.3.12 right()

```
int16_t right ( ) const [inline]
```

Gets the x coordinate of the right edge of the [Rect](#).

#### Returns

x coordinate of the right edge.

## 7.147 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.*

### 7.147.1 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.

### 7.147.2 Constructor & Destructor Documentation

#### 7.147.2.1 [Renderer](#)() [1/2]

```
Renderer ( ) [inline]
```

Default constructor. Function [setRenderingBuffer\(\)](#) should be called to specify where the polygon should be rendered.

#### 7.147.2.2 [Renderer](#)() [2/2]

```
Renderer (
    RenderingBuffer & renderingBuffer,
    AbstractPainter & painter ) [inline]
```

Constructor.



## Parameters

in	<i>renderingBuffer</i>	The screen buffer to render the polygon in.
in	<i>painter</i>	The painter to use for drawing individual pixels in a scanline.

## 7.147.3 Member Function Documentation

## 7.147.3.1 getRenderingBuffer()

```
RenderingBuffer & getRenderingBuffer ( ) [inline]
```

Gets the getRenderingBuffer.

## Returns

A [RenderingBuffer&](#)

## 7.147.3.2 render()

```
void render (
    const Scanline & scanline ) [inline]
```

Render the given [Scanline](#) in the given color.

## Parameters

<i>scanline</i>	The <a href="#">Scanline</a> .
-----------------	--------------------------------

## 7.147.3.3 setRenderingBuffer()

```
void setRenderingBuffer (
    RenderingBuffer & renderingBuffer ) [inline]
```

Sets rendering buffer.

## Parameters

in	<i>renderingBuffer</i>	The screen buffer to render the polygon in.
----	------------------------	---------------------------------------------

## 7.148 RenderingBuffer Class Reference

Rendering buffer wrapper.

```
#include <touchgfx/canvas_widget_renderer/RenderingBuffer.hpp>
```

## Public Member Functions

- [RenderingBuffer](#) ()  
*Default constructor.*
- [~RenderingBuffer](#) ()  
*Destructor.*
- [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](#).*

### 7.148.1 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 `height * 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.

## 7.148.2 Constructor & Destructor Documentation

### 7.148.2.1 RenderingBuffer() [1/2]

```
RenderingBuffer ( )
```

Default constructor.

### 7.148.2.2 ~RenderingBuffer()

```
~RenderingBuffer ( )
```

Destructor.

### 7.148.2.3 RenderingBuffer() [2/2]

```
RenderingBuffer (
    unsigned char * buf_,
    unsigned char xAdjust_,
    unsigned width_,
    unsigned height_,
    int stride_ )
```

Constructor.

#### Parameters

in	<i>buf_</i>	Pointer to the frame buffer where the image is rendered.
	<i>xAdjust_</i>	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.
	<i>width_</i>	The width of the frame buffer to write.
	<i>height_</i>	The height of the frame buffer to write.
	<i>stride_</i>	How much to add the a pointer inside the frame buffer to advance to the next line in the frame buffer.

## 7.148.3 Member Function Documentation

### 7.148.3.1 attach()

```
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	<i>buf_</i>	Pointer to the frame buffer where the image is rendered.
	<i>x↔ Adjust↔ _</i>	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.
	<i>width_</i>	The width of the frame buffer to write.
	<i>height↔ _</i>	The height of the frame buffer to write.
	<i>stride↔ _</i>	How much to add the a pointer inside the frame buffer to advance to the next line in the frame buffer.

## 7.148.3.2 getHeight()

```
unsigned getHeight ( ) const [inline]
```

Gets the height.

## Returns

The height.

## 7.148.3.3 getWidth()

```
unsigned getWidth ( ) const [inline]
```

Gets the width.

## Returns

The width.

## 7.148.3.4 getXAdjust()

```
unsigned char getXAdjust ( ) const [inline]
```

Gets x coordinate adjust.

## Returns

The x coordinate adjust.

## 7.148.3.5 inbox()

```
bool inbox (
    int x,
    int y ) const [inline]
```

Tests if a given coordinate is inside the [RenderingBuffer](#).

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

**Returns**

true if (x,y) is inside the [RenderingBuffer](#), false otherwise.

**7.148.3.6 row() [1/2]**

```
unsigned char * row (  
    unsigned y ) [inline]
```

Gets a pointer to the given row in the [RenderingBuffer](#).

**Parameters**

<i>y</i>	The line number, ie the row.
----------	------------------------------

**Returns**

The pointer to the start of the given line in the [RenderingBuffer](#).

**7.148.3.7 row() [2/2]**

```
const unsigned char * row (  
    unsigned y ) const [inline]
```

Gets a pointer to the given row in the [RenderingBuffer](#).

**Parameters**

<i>y</i>	The line number, ie the row.
----------	------------------------------

**Returns**

The pointer to the start of the given line in the [RenderingBuffer](#).

## 7.149 RepeatButton Class Reference

A button with two states.

```
#include <touchgfx/widgets/RepeatButton.hpp>
```

**Public Member Functions**

- [RepeatButton](#) ()

*Default constructor.*

- virtual void [setDelay](#) (int delay)  
*Sets the delay.*
- virtual int [getDelay](#) ()  
*Gets the delay.*
- virtual void [setInterval](#) (int interval)  
*Sets the interval.*
- virtual int [getInterval](#) ()  
*Gets the interval.*
- virtual void [handleClickEvent](#) (const [touchgfx::ClickEvent](#) &event)  
*Handles the click event.*
- virtual void [handleTickEvent](#) ()  
*Handles the tick event.*

## Additional Inherited Members

### 7.149.1 Detailed Description

A button consists of two images, one for its normal state and one when it is pressed down. The button activates its pressed action immediately, then after a given delay and then repeatedly after an interval.

See also

[Button](#)

### 7.149.2 Constructor & Destructor Documentation

#### 7.149.2.1 RepeatButton()

[RepeatButton](#) ( )

Default constructor. Sets delay to 10 ticks and interval to 5 ticks.

See also

[setDelay](#)  
[setInterval](#)

### 7.149.3 Member Function Documentation

#### 7.149.3.1 getDelay()

int [getDelay](#) ( ) [virtual]

Gets the delay in ticks.

Returns

The delay.

See also

[setDelay](#)

#### 7.149.3.2 `getInterval()`

```
int getInterval ( ) [virtual]
```

Gets the interval in ticks.

##### Returns

The interval.

#### 7.149.3.3 `handleClickEvent()`

```
void handleClickEvent (
    const touchgfx::ClickEvent & event ) [virtual]
```

Handles the click event by immediately activating the button and then setting up a timer to repeatedly activate the button.

##### Parameters

<i>event</i>	The event.
--------------	------------

Reimplemented from [AbstractButton](#).

#### 7.149.3.4 `handleTickEvent()`

```
void handleTickEvent ( ) [virtual]
```

Handles the tick event that takes care of counting down until the next time the buttons should be activated.

Reimplemented from [Drawable](#).

#### 7.149.3.5 `setDelay()`

```
void setDelay (
    int delay ) [virtual]
```

Sets the number of ticks from the first button activation until the next time it gets activated.

##### Parameters

<i>delay</i>	The delay.
--------------	------------

See also

[setInterval](#)  
[getDelay](#)

#### 7.149.3.6 setInterval()

```
void setInterval (
    int interval ) [virtual]
```

Sets the interval in number of ticks between each activation of the pressed button.

Parameters

<i>interval</i>	The interval.
-----------------	---------------

See also

[setDelay](#)  
[getInterval](#)

## 7.150 RepeatButtonTrigger Class Reference

A repeat button trigger.

```
#include <touchgfx/containers/buttons/RepeatButtonTrigger.hpp>
```

### Public Member Functions

- [RepeatButtonTrigger](#) ()  
*Default constructor.*
- virtual [~RepeatButtonTrigger](#) ()  
*Destructor.*
- void [setDelay](#) (int delay)  
*Sets a delay.*
- int [getDelay](#) ()  
*Gets the delay.*
- void [setInterval](#) (int interval)  
*Sets an interval.*
- int [getInterval](#) ()  
*Gets the interval.*
- void [handleClickEvent](#) (const [touchgfx::ClickEvent](#) &event)  
*Handles the click event described by event.*
- void [handleTickEvent](#) ()  
*Handles the tick event.*

### Additional Inherited Members



### 7.150.1 Detailed Description

A repeat button trigger. This trigger will create a button that reacts to a consistent touch. This means it will call the action repeatedly as long as it is touched.

The [RepeatButtonTrigger](#) can be combined with one or more of the [ButtonStyle](#) classes to create a functional button.

### 7.150.2 Member Function Documentation

#### 7.150.2.1 getDelay()

```
int getDelay ( ) [inline]
```

##### Returns

The delay.

#### 7.150.2.2 getInterval()

```
int getInterval ( ) [inline]
```

##### Returns

The interval.

#### 7.150.2.3 handleClickEvent()

```
void handleClickEvent (
    const touchgfx::ClickEvent & event ) [inline], [virtual]
```

##### Parameters

<i>event</i>	The event.
--------------	------------

Reimplemented from [Drawable](#).

#### 7.150.2.4 setDelay()

```
void setDelay (
    int delay ) [inline]
```

##### Parameters

<i>delay</i>	The delay.
--------------	------------

7.150.2.5 `setInterval()`

```
void setInterval (
    int interval ) [inline]
```

## Parameters

<code>interval</code>	The interval.
-----------------------	---------------

## 7.151 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

### 7.151.1 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](#)

### 7.151.2 Member Enumeration Documentation

#### 7.151.2.1 ScalingAlgorithm

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.

### 7.151.3 Constructor & Destructor Documentation

#### 7.151.3.1 ScalableImage()

[ScalableImage](#) ( )

Default constructor.

#### 7.151.3.2 ~ScalableImage()

[~ScalableImage](#) ( ) [virtual]

Destructor.

## 7.151.4 Member Function Documentation

### 7.151.4.1 draw()

```
void draw (
    const Rect & invalidatedArea ) const [virtual]
```

Draws the given invalidated area.

#### Parameters

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

#### See also

[Drawable::draw\(\)](#)

Implements [Drawable](#).

### 7.151.4.2 drawTriangle()

```
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

	<i>invalidatedArea</i>	The invalidated area.
<i>in, out</i>	<i>fb</i>	If non-null, the fb.
	<i>triangleXs</i>	The triangle xs.
	<i>triangleYs</i>	The triangle ys.
	<i>triangleZs</i>	The triangle zs.
	<i>triangleUs</i>	The triangle us.
	<i>triangleVs</i>	The triangle vs.

### 7.151.4.3 getAlpha()

```
uint8_t getAlpha ( ) const [inline], [virtual]
```

Gets the current alpha value.

**Returns**

The current alpha value.

**7.151.4.4 getBitmap()**

```
Bitmap getBitmap ( ) const [inline]
```

Gets the bitmap for the image.

**Returns**

the small bitmap.

**7.151.4.5 getScalingAlgorithm()**

```
ScalingAlgorithm getScalingAlgorithm ( ) [virtual]
```

Gets the algorithm used when rendering.

**Returns**

The algorithm used when rendering.

**7.151.4.6 getSolidRect()**

```
Rect getSolidRect ( ) const [virtual]
```

Gets solid rectangle.

**Returns**

largest possible solid rect.

**See also**

[Drawable::getSolidRect\(\)](#)

Implements [Drawable](#).

**7.151.4.7 getType()**

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

**Returns**

TYPE\_SCALABLEIMAGE.

Reimplemented from [Widget](#).

#### 7.151.4.8 lookupRenderVariant()

```
RenderingVariant lookupRenderVariant ( ) const [protected]
```

Looks up the appropriate render variant based on the bitmap format and scaling algorithm.

##### Returns

A RenderingVariant.

#### 7.151.4.9 setAlpha()

```
void setAlpha (
    uint8_t alpha ) [inline], [virtual]
```

Sets the alpha channel for the image.

##### Parameters

<i>alpha</i>	The alpha value. 255 = completely solid.
--------------	------------------------------------------

#### 7.151.4.10 setBitmap()

```
void setBitmap (
    const Bitmap & bmp ) [virtual]
```

Sets the bitmap for the image.

##### Parameters

<i>bmp</i>	The bitmap to be used by the widget.
------------	--------------------------------------

#### 7.151.4.11 setScalingAlgorithm()

```
void setScalingAlgorithm (
    ScalingAlgorithm algorithm ) [virtual]
```

Sets the algorithm to be used.

##### Parameters

<i>algorithm</i>	The algorithm to use when rendering.
------------------	--------------------------------------

## 7.152 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 Member Functions

- [Scanline](#) ()  
*Default constructor.*
- virtual [~Scanline](#) ()  
*Destructor.*
- 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](#).*

### 7.152.1 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 it 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.

## 7.152.2 Constructor & Destructor Documentation

### 7.152.2.1 Scanline()

```
Scanline ( )
```

Default constructor. Initiate a [Scanline](#) by setting up pointers to store covers, and counts.

### 7.152.2.2 ~Scanline()

```
~Scanline ( ) [inline], [virtual]
```

Destructor.

## 7.152.3 Member Function Documentation

### 7.152.3.1 addCell()

```
FORCE_INLINE_FUNCTION void addCell (
    int x,
    int y,
    unsigned cover )
```

Adds a single cell to the current [Scanline](#). Works just like invoking [addSpan\(\)](#) with a len=1.

#### Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>cover</i>	The cover.



### 7.152.3.2 addSpan()

```
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

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>len</i>	The length.
<i>cover</i>	The cover.

### 7.152.3.3 getNumSpans()

```
unsigned getNumSpans ( ) const [inline]
```

Gets number spans in the [Scanline](#).

#### Returns

The number spans.

### 7.152.3.4 getY()

```
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.

### 7.152.3.5 isReady()

```
FORCE_INLINE_FUNCTION int isReady (
    int y ) const
```

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

<code>y</code>	The y coordinate.
----------------	-------------------

## Returns

True if the given y coordinate differs from the y coordinate for the cells in the current [Scanline](#).

7.152.3.6 `reset()`

```
void reset ( )
```

Resets the [Scanline](#) object in preparation for the handling the next [Scanline](#).

7.152.3.7 `resetSpans()`

```
FORCE_INLINE_FUNCTION void resetSpans ( )
```

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\(\)](#).

## 7.153 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).*

### 7.153.1 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.

### 7.153.2 Constructor & Destructor Documentation

#### 7.153.2.1 `Screen()`

```
Screen ( )
```

Default constructor.

#### 7.153.2.2 `~Screen()`

```
~Screen ( ) [inline], [virtual]
```

Destructor.

### 7.153.3 Member Function Documentation

#### 7.153.3.1 add()

```
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	<i>d</i>	The <a href="#">Drawable</a> to add.
----	----------	--------------------------------------

#### 7.153.3.2 afterTransition()

```
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()`

#### 7.153.3.3 bindTransition()

```
void bindTransition (
    Transition & trans )
```

Enables the transition to access the containers.

##### Parameters

in	<i>trans</i>	The transition to bind.
----	--------------	-------------------------

#### 7.153.3.4 draw() [1/2]

```
void draw ( )
```

Tells the screen to draw its entire area.

**Note**

The more specific draw([Rect](#)&) version is preferred when possible.

**7.153.3.5 draw()** [2/2]

```
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	<i>rect</i>	The area in absolute coordinates.
----	-------------	-----------------------------------

**7.153.3.6 getRootContainer()**

```
Container & getRootContainer ( ) [inline]
```

Obtain a reference to the root container of this screen.

**Returns**

The root container.

**7.153.3.7 handleClickEvent()**

```
void handleClickEvent (
    const ClickEvent & evt ) [virtual]
```

Traverse the drawables in reverse z-order and notify them of a click event.

**Parameters**

<i>evt</i>	The event to handle.
------------	----------------------

**7.153.3.8 handleDragEvent()**

```
void handleDragEvent (
    const DragEvent & evt ) [virtual]
```

Traverse the drawables in reverse z-order and notify them of a drag event.

**Parameters**

<i>evt</i>	The event to handle.
------------	----------------------

**7.153.3.9 handleGestureEvent()**

```
void handleGestureEvent (
    const GestureEvent & evt ) [virtual]
```

Handle gestures. Traverses drawables in reverse-z and notifies them of the gesture.

**Parameters**

<i>evt</i>	The event to handle.
------------	----------------------

**7.153.3.10 handleKeyEvent()**

```
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**

<i>key</i>	The key to handle.
------------	--------------------

**7.153.3.11 handleTickEvent()**

```
void handleTickEvent ( ) [inline], [virtual]
```

Called by the [Application](#) on the current screen with a frequency of [Application::TICK\\_INTERVAL\\_MS](#).

**7.153.3.12 JSMOC()**

```
void JSMOC (
    const Rect & invalidatedArea,
    Drawable * widgetToDraw )
```

Recursive JSMOC function. This is the actual occlusion culling implementation.

**Parameters**

in	<i>invalidatedArea</i>	The area to redraw, expressed in absolute coordinates.
in	<i>widgetToDraw</i>	<a href="#">Widget</a> currently being drawn.

**7.153.3.13 remove()**

```
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	<i>d</i>	The <a href="#">Drawable</a> to remove.
----	----------	-----------------------------------------

**7.153.3.14 setupScreen()**

```
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\(\)](#)

**7.153.3.15 startSMOC()**

```
void startSMOC (
    Rect & invalidatedArea )
```

Starts a JSMOC run, analyzing what parts of what widgets should be redrawn.

**Parameters**

in	<i>invalidatedArea</i>	The area to redraw, expressed in absolute coordinates.
----	------------------------	--------------------------------------------------------

**7.153.3.16 tearDownScreen()**

```
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\(\)](#)

**7.153.3.17 useSMOCDrawing()**

```
void useSMOCDrawing (
    bool enabled ) [protected]
```

Determines whether to use JSMOC or painter's algorithm for drawing.

#### Parameters

<i>enabled</i>	true if JSMOC should be enabled, false if disabled (meaning painter's algorithm is employed instead).
----------------	-------------------------------------------------------------------------------------------------------

#### 7.153.3.18 usingSMOC()

```
bool usingSMOC ( ) const [inline]
```

#### Returns

true if this screen uses the JSMOC drawing algorithm.

## 7.154 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.*
- virtual void [isScrollableXY](#) (bool &scrollX, bool &scrollY)  
*Is the ClickableContainer scrollable in either direction?*
- void [setScrollbarsVisible](#) (bool newVisible)  
*Sets the visibility of the scrollbars, when the scrollable area is pressed.*
- void [setScrollbarsPermanentlyVisible](#) ()  
*sets the visibility for the scrollbars to be permanent.*
- 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 [handleClickEvent](#) (const [ClickEvent](#) &evt)  
*Handle the click event.*
- virtual void [handleDragEvent](#) (const [DragEvent](#) &evt)  
*Handle the drag event.*
- virtual void [handleGestureEvent](#) (const [GestureEvent](#) &evt)  
*[Gestures](#) generate a scroll animation so these are intercepted in the same manner as drag events.*
- 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.*
- int16\_t [getScrolledX](#) () const  
*Gets the distance scrolled for the x-axis.*
- int16\_t [getScrolledY](#) () const  
*Gets the distance scrolled for the y-axis.*
- 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 always visible.*
- bool [scrollbarsPermanentlyVisible](#)
  - Are scrollbars always 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](#) = 0  
*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

### 7.154.1 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](#)

### 7.154.2 Constructor & Destructor Documentation

#### 7.154.2.1 ScrollableContainer()

```
ScrollableContainer ( )
```

Default constructor.

#### 7.154.2.2 ~ScrollableContainer()

```
~ScrollableContainer ( ) [inline], [virtual]
```

Destructor.

### 7.154.3 Member Function Documentation

#### 7.154.3.1 add()

```
void add (
    Drawable & d ) [virtual]
```

Adds a [Drawable](#) instance as child to this [ScrollableContainer](#).

**Parameters**

<i>in</i>	<i>d</i>	The drawable.
-----------	----------	---------------

Reimplemented from [Container](#).

**7.154.3.2 childGeometryChanged()**

```
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](#).

**7.154.3.3 doScroll()**

```
bool doScroll (
    int16_t deltaX,
    int16_t deltaY ) [protected], [virtual]
```

Method to actually scroll the container. Passing negative values will scroll the items in the [ScrollableContainer](#) up / left, whereas positive values will scroll items down / right.

If the distance is larger than allowed, the deltas are adjusted down to make sure the contained items stay inside view.

**Parameters**

<i>deltaX</i>	The horizontal amount to scroll.
<i>deltaY</i>	The vertical amount to scroll.

**Returns**

did the container actually scroll. The call doScroll(0,0) will always return false.

**7.154.3.4 enableHorizontalScroll()**

```
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**

<i>enable</i>	If true (default), horizontal scrolling is enabled. If false, scrolling is disabled.
---------------	--------------------------------------------------------------------------------------

### 7.154.3.5 enableVerticalScroll()

```
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

<i>enable</i>	If true (default), vertical scrolling is enabled. If false, scrolling is disabled.
---------------	------------------------------------------------------------------------------------

### 7.154.3.6 getContainedArea()

```
Rect getContainedArea ( ) const [virtual]
```

Gets contained area.

#### Returns

The contained area.

Reimplemented from [Container](#).

### 7.154.3.7 getLastChild()

```
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

	<i>x</i>	The x coordinate of the (click) event.
	<i>y</i>	The y coordinate of the (click) event.
out	<i>last</i>	The last child intersecting x,y. <a href="#">ScrollableContainer</a> intercepts these, so returns it self.

Reimplemented from [Container](#).

### 7.154.3.8 getScrolledX()

```
int16_t getScrolledX ( ) const
```

Gets the distance scrolled for the x-axis.

**Returns**

the distance scrolled for the x-axis.

**7.154.3.9 getScrolledY()**

```
int16_t getScrolledY ( ) const
```

Gets the distance scrolled for the y-axis.

**Returns**

the distance scrolled for the y-axis.

**7.154.3.10 getType()**

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

**Returns**

TYPE\_SCROLLABLECONTAINER.

Reimplemented from [Container](#).

**7.154.3.11 getXBorder()**

```
Rect getXBorder (
    const Rect & xBar,
    const Rect & yBar ) const [protected]
```

Gets the area where the horizontal scrollbar can move.

**Parameters**

<i>xBar</i>	The current horizontal scrollbar, supplied for caching reasons.
<i>yBar</i>	The current vertical scrollbar, supplied for caching reasons.

**Returns**

The area.

**7.154.3.12 getXScrollbar()**

```
Rect getXScrollbar ( ) const [protected]
```

Gets x coordinate of the scrollbar.

**Returns**

The horizontal scrollbar area.

**7.154.3.13 getYBorder()**

```
Rect getYBorder (
    const Rect & xBar,
    const Rect & yBar ) const [protected]
```

Gets the area where the vertical scrollbar can move.

**Parameters**

<i>xBar</i>	The current horizontal scrollbar, supplied for caching reasons.
<i>yBar</i>	The current vertical scrollbar, supplied for caching reasons.

**Returns**

The area.

**7.154.3.14 getYScrollbar()**

```
Rect getYScrollbar ( ) const [protected]
```

Gets y coordinate of the scrollbar.

**Returns**

The vertical scrollbar area.

**7.154.3.15 handleClickEvent()**

```
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**

<i>evt</i>	The <a href="#">ClickEvent</a> .
------------	----------------------------------

Reimplemented from [Drawable](#).

**7.154.3.16 handleDragEvent()**

```
void handleDragEvent (
    const DragEvent & evt ) [virtual]
```

Handle the drag event. Initiate a scrolling of the container. Update scrollbars.

#### Parameters

<i>evt</i>	The <a href="#">DragEvent</a> .
------------	---------------------------------

Reimplemented from [Drawable](#).

#### 7.154.3.17 handleGestureEvent()

```
void handleGestureEvent (
    const GestureEvent & evt ) [virtual]
```

[Gestures](#) generate a scroll animation so these are intercepted in the same manner as drag events.

#### Parameters

<i>evt</i>	The <a href="#">GestureEvent</a> .
------------	------------------------------------

Reimplemented from [Drawable](#).

#### 7.154.3.18 handleTickEvent()

```
void handleTickEvent ( ) [virtual]
```

Handle tick events. Used in updating the animation of the scroll.

Reimplemented from [Drawable](#).

#### 7.154.3.19 invalidateScrollbars()

```
void invalidateScrollbars ( ) [protected]
```

Invalidate the scrollbars.

#### 7.154.3.20 isScrollableXY()

```
void isScrollableXY (
    bool & scrollX,
    bool & scrollY ) [inline], [virtual]
```

Is the ClickableContainer scrollable in either direction? Takes the width of the contained elements into account.

#### Parameters

<i>in, out</i>	<i>scrollX</i>	Is the container able to scroll horizontally.
<i>in, out</i>	<i>scrollY</i>	Is the container able to scroll vertically.



### 7.154.3.21 moveChildrenRelative()

```
void moveChildrenRelative (
    int16_t deltaX,
    int16_t deltaY ) [virtual]
```

Moves the scrollable contents relatively.

#### Parameters

<i>deltaX</i>	Horizontal displacement.
<i>deltaY</i>	Vertical displacement.

Reimplemented from [Container](#).

### 7.154.3.22 reset()

```
void reset ( )
```

Resets the x/y coordinates of children to 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.

### 7.154.3.23 setMaxVelocity()

```
void setMaxVelocity (
    uint16_t max ) [inline]
```

Sets the maximum velocity of a scroll due to a swipe.

#### Parameters

<i>max</i>	The maximum velocity of the scroll.
------------	-------------------------------------

### 7.154.3.24 setScrollbarPadding()

```
void setScrollbarPadding (
    uint8_t padding )
```

Sets the amount of space the scrollbar has to its borders.

#### Parameters

<i>padding</i>	The padding.
----------------	--------------

### 7.154.3.25 setScrollbarsAlpha()

```
void setScrollbarsAlpha (
    uint8_t alpha )
```

Sets the alpha value for the scroll bars.

## Parameters

<i>alpha</i>	The alpha value. 255 = completely solid.
--------------	------------------------------------------

**7.154.3.26 setScrollbarsColor()**

```
void setScrollbarsColor (
    colortype color )
```

Sets the color of the scroll bars.

## Parameters

<i>color</i>	The color of the box.
--------------	-----------------------

**7.154.3.27 setScrollbarsPermanentlyVisible()**

```
void setScrollbarsPermanentlyVisible ( )
```

sets the visibility for the scrollbars to be permanent.

**7.154.3.28 setScrollbarsVisible()**

```
void setScrollbarsVisible (
    bool newVisible )
```

Sets the visibility of the scrollbars, when the scrollable area is pressed.

## Parameters

<i>newVisible</i>	If true (default), the scrollbars are visible when scrollable area is pressed. If false, scrollbars are always hidden.
-------------------	------------------------------------------------------------------------------------------------------------------------

**7.154.3.29 setScrollbarWidth()**

```
void setScrollbarWidth (
    uint8_t width )
```

Sets the width of the scrollbar.

## Parameters

<i>width</i>	The width of the scrollbar.
--------------	-----------------------------

## 7.154.3.30 setScrollThreshold()

```
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

<i>t</i>	The new threshold value.
----------	--------------------------

## 7.155 ScrollBase Class Reference

A scroll base class.

```
#include <touchgfx/containers/scrollers/ScrollBase.hpp>
```

## Public Member Functions

- [ScrollBase](#) ()  
*Default constructor.*
- virtual [~ScrollBase](#) ()  
*Destructor.*
- virtual void [setWidth](#) (int16\_t width)  
*Sets width of the [ScrollBase](#).*
- virtual void [setHeight](#) (int16\_t height)  
*Sets height of the [ScrollBase](#).*
- virtual void [setHorizontal](#) (bool horizontal)  
*Sets a horizontal layout.*
- virtual bool [getHorizontal](#) () const  
*Gets the orientation of the drawables.*
- virtual void [setCircular](#) (bool circular)  
*Sets whether the list is circular or not.*
- virtual bool [getCircular](#) () const  
*Gets the circular setting.*
- void [setDrawableSize](#) (int16\_t drawableSize, int16\_t drawableMargin)  
*Sets drawables size.*
- virtual int16\_t [getDrawableSize](#) () const  
*Gets drawable size.*
- virtual int16\_t [getDrawableMargin](#) () const  
*Gets drawable margin.*
- virtual void [setNumberOfItems](#) (int16\_t numberOfItems)  
*Sets number of items in the [DrawableList](#).*
- virtual int16\_t [getNumberOfItems](#) () const  
*Gets number of items in the [DrawableList](#).*
- void [setEasingEquation](#) ([EasingEquation](#) equation)

- Sets easing equation.*

  - void [setAnimationSteps](#) (int16\_t steps)
- Sets animation steps.*

  - uint16\_t [getAnimationSteps](#) () const
- Gets animation steps.*

  - void [setSwipeAcceleration](#) (uint16\_t acceleration)
- Sets swipe acceleration.*

  - uint16\_t [getSwipeAcceleration](#) () const
- Gets swipe acceleration.*

  - void [setMaxSwipeItems](#) (uint16\_t maxItems)
- Sets maximum swipe items.*

  - uint16\_t [getMaxSwipeItems](#) () const
- Gets maximum swipe items.*

  - void [setDragAcceleration](#) (uint16\_t acceleration)
- Sets drag acceleration.*

  - uint16\_t [getDragAcceleration](#) () const
- Gets drag acceleration.*

  - void [allowHorizontalDrag](#) (bool enable)
- Enables horizontal scrolling.*

  - void [allowVerticalDrag](#) (bool enable)
- Enables the vertical scroll.*

  - virtual void [animateToItem](#) (int16\_t itemIndex, int16\_t animationSteps=-1)
- Go to item.*

  - void [setItemSelectedCallback](#) (GenericCallback< int16\_t > &callback)
- Sets [Callback](#) which will be called when the selected item is clicked.*

  - void [setAnimationEndedCallback](#) (GenericCallback<> &callback)
- [Callback](#), called when the set animation ended.*

  - void [setItemPressedCallback](#) (GenericCallback< int16\_t > &callback)
- Set [Callback](#) which will be called when a item is pressed.*

  - bool [isAnimating](#) () const
- Query if this object is animating.*

  - void [stopAnimation](#) ()
- Stops an animation.*

  - virtual void [handleDragEvent](#) (const [DragEvent](#) &evt)
- Handles the drag event described by evt.*

  - virtual void [handleGestureEvent](#) (const [GestureEvent](#) &evt)
- Handles the gesture event described by evt.*

  - virtual void [handleTickEvent](#) ()
- Handles the tick event.*

  - virtual void [itemChanged](#) (int itemIndex)
- Item changed.*

  - virtual void [initialize](#) ()
- Initializes the contents of all drawables.*

## Protected Types

- enum [AnimationState](#) { [NO\\_ANIMATION](#), [ANIMATING\\_GESTURE](#), [ANIMATING\\_DRAG](#) }

*Values that represent animation states.*

## Protected Member Functions

- virtual void [setOffset](#) (int32\_t offset)  
*Sets display offset of first item.*
- virtual int32\_t [getOffset](#) () const  
*Gets display offset of first item.*
- virtual int32\_t [getPositionForItem](#) (int16\_t itemIndex)=0  
*Gets position for an item.*
- int [getNormalizedOffset](#) (int offset) const  
*Gets normalized offset from a given offset.*
- virtual int32\_t [keepOffsetInsideLimits](#) (int32\_t newOffset, int16\_t overShoot) const =0  
*Keep offset inside limits.*
- virtual int32\_t [getNearestAlignedOffset](#) (int32\_t offset) const  
*Gets nearest offset aligned to a multiple of itemSize.*
- virtual void [animateToPosition](#) (int32\_t position, int16\_t steps=-1)  
*Animate to a new position/offset using the given number of steps.*

## Protected Attributes

- [DrawableList](#) list  
*The list.*
- int16\_t [numberOfDrawables](#)  
*Number of drawables.*
- int16\_t [distanceBeforeAlignedItem](#)  
*The distance before aligned item.*
- int16\_t [itemSize](#)  
*Size of the item.*
- uint16\_t [swipeAcceleration](#)  
*The swipe acceleration.*
- uint16\_t [dragAcceleration](#)  
*The drag acceleration.*
- uint16\_t [maxSwipeItems](#)  
*The maximum swipe items.*
- [EasingEquation](#) easingEquation  
*The easing equation.*
- uint16\_t [defaultAnimationSteps](#)  
*The animation steps.*
- [GenericCallback](#)< int16\_t > \* [itemSelectedCallback](#)  
*The item selected callback.*
- [GenericCallback](#) \* [itemLockedInCallback](#)  
*The item locked in callback.*
- [GenericCallback](#) \* [animationEndedCallback](#)  
*The animation ended callback.*
- [GenericCallback](#)< int16\_t > \* [itemPressedCallback](#)  
*The item pressed callback.*
- [AnimationState](#) currentAnimationState  
*The current animation state.*
- int [gestureStep](#)  
*The gesture step.*
- int [gestureStepsTotal](#)  
*The gesture steps total.*

- int [gestureStart](#)  
*The gesture start.*
- int [gestureEnd](#)  
*The gesture end.*
- int16\_t [xClick](#)  
*The click.*
- int16\_t [yClick](#)  
*The click.*
- int32\_t [initialSwipeOffset](#)  
*The initial swipe offset.*
- bool [draggableX](#)  
*Is the container draggable in the horizontal direction.*
- bool [draggableY](#)  
*Is the container draggable in the vertical direction.*

## Additional Inherited Members

### 7.155.1 Detailed Description

A scroll base class with a list of drawables ([DrawableList](#)).

See also

[ScrollWheelBase](#)  
[ScrollList](#)  
[DrawableList](#)

### 7.155.2 Member Enumeration Documentation

#### 7.155.2.1 AnimationState

```
enum AnimationState [protected]
```

Enumerator

<a href="#">NO_ANIMATION</a>	No animation.
<a href="#">ANIMATING_GESTURE</a>	Animating a gesture.
<a href="#">ANIMATING_DRAG</a>	Animating a click+drag.

### 7.155.3 Constructor & Destructor Documentation

#### 7.155.3.1 ScrollBase()

```
ScrollBase ( )
```

Default constructor.

## 7.155.3.2 ~ScrollBase()

```
~ScrollBase ( ) [inline], [virtual]
```

Destructor.

## 7.155.4 Member Function Documentation

## 7.155.4.1 allowHorizontalDrag()

```
void allowHorizontalDrag (
    bool enable )
```

Enables horizontal scrolling to be passed to the children. By default, scrolling in either direction is enabled. This function can be used to explicitly (dis)allow scrolling in the horizontal direction.

## Parameters

<i>enable</i>	If true (default), horizontal scrolling is enabled. If false, scrolling is disabled.
---------------	--------------------------------------------------------------------------------------

## 7.155.4.2 allowVerticalDrag()

```
void allowVerticalDrag (
    bool enable )
```

Enables the vertical scroll to be passed to the children. By default, scrolling in either direction is enabled. This function can be used to explicitly (dis)allow scrolling in the vertical direction.

## Parameters

<i>enable</i>	If true (default), vertical scrolling is enabled. If false, scrolling is disabled.
---------------	------------------------------------------------------------------------------------

## 7.155.4.3 animateToItem()

```
void animateToItem (
    int16_t itemIndex,
    int16_t animationSteps = -1 ) [virtual]
```

Go to item, possibly with animation. The given item index is scrolled into view. If animationSteps is omitted, the default number of animation steps is used. If animationSteps is 0 no animation will be used, otherwise the number of animation steps specified is used.

## Parameters

<i>itemIndex</i>	Zero-based index of the item.
<i>animationSteps</i>	(Optional) The steps to use for the animation. 0 means no animation. If omitted, default animation steps are used.

See also

[setAnimationSteps](#)

#### 7.155.4.4 animateToPosition()

```
void animateToPosition (
    int32_t position,
    int16_t steps = -1 ) [protected], [virtual]
```

Animate to a new position/offset using the given number of steps.

Parameters

<i>position</i>	The new position.
<i>steps</i>	The steps.

Reimplemented in [ScrollWheelBase](#).

#### 7.155.4.5 getAnimationSteps()

```
uint16_t getAnimationSteps ( ) const
```

Gets animation steps as set in [setAnimationSteps](#).

Returns

The animation steps.

See also

[setAnimationSteps](#)

[setEasingEquation](#)

#### 7.155.4.6 getCircular()

```
bool getCircular ( ) const [virtual]
```

Gets the circular setting, previously set using [setCircular\(\)](#).

Returns

True if the list is circular (infinite), false if the list is not circular (finite).

See also

[DrawableList::getCircular](#)

[setCircular](#)



#### 7.155.4.7 getDragAcceleration()

```
uint16_t getDragAcceleration ( ) const
```

Gets drag acceleration (times 10).

##### Returns

The drag acceleration.

##### Note

The reason for multiplying the acceleration by 10 is to avoid introducing floating point arithmetics.

##### See also

[setDragAcceleration](#)

#### 7.155.4.8 getDrawableMargin()

```
int16_t getDrawableMargin ( ) const [virtual]
```

Gets drawable margin as set through the argument in setDrawables().

##### Returns

The drawable margin.

#### 7.155.4.9 getDrawableSize()

```
int16_t getDrawableSize ( ) const [virtual]
```

Gets drawable size as set through the first argument in setDrawables().

##### Returns

The drawable size.

##### See also

[setDrawables](#)

#### 7.155.4.10 getHorizontal()

```
bool getHorizontal ( ) const [virtual]
```

Gets the orientation of the drawables, previously set using setHorizontal.

##### Returns

True if it horizontal, false if it is vertical.

##### See also

[DrawableList::getHorizontal](#)  
[setHorizontal](#)

#### 7.155.4.11 getMaxSwipeItems()

```
uint16_t getMaxSwipeItems ( ) const
```

Gets maximum swipe items as set by `setMaxSwipeItems`.

##### Returns

The maximum swipe items, 0 means "no limit".

##### See also

[setMaxSwipeItems](#)

#### 7.155.4.12 getNearestAlignedOffset()

```
int32_t getNearestAlignedOffset (
    int32_t offset ) const [protected], [virtual]
```

Gets nearest offset aligned to a multiple of `itemSize`.

##### Parameters

<i>offset</i>	The offset.
---------------	-------------

##### Returns

The nearest aligned offset.

Reimplemented in [ScrollList](#).

#### 7.155.4.13 getNormalizedOffset()

```
int getNormalizedOffset (
    int offset ) const [protected]
```

Gets normalized offset from a given offset from 0 down to `-numItems*itemSize`.

##### Parameters

<i>offset</i>	The offset.
---------------	-------------

##### Returns

The normalized offset.

#### 7.155.4.14 getNumberOfItems()

```
int16_t getNumberOfItems ( ) const [virtual]
```

Gets number of items in the [DrawableList](#), as previously set using [setNumberOfItems\(\)](#).

**Returns**

The number of items.

**See also**

[setNumberOfItems](#)  
[DrawableList::getNumberOfItems](#)

**7.155.4.15 getOffset()**

```
int32_t getOffset ( ) const [protected], [virtual]
```

Gets display offset of first item.

**Returns**

The offset.

**7.155.4.16 getPositionForItem()**

```
int32_t getPositionForItem (
    int16_t itemIndex ) [protected], [pure virtual]
```

Get the position for an item. The position should ensure that the item is in view as defined by the semantics of the actual scroll class.

**Parameters**

<i>itemIndex</i>	Zero-based index of the item.
------------------	-------------------------------

**Returns**

The position for item.

Implemented in [ScrollList](#), and [ScrollWheelBase](#).

**7.155.4.17 getSwipeAcceleration()**

```
uint16_t getSwipeAcceleration ( ) const
```

Gets swipe acceleration (times 10).

**Returns**

The swipe acceleration.

**Note**

The reason for multiplying the acceleration by 10 is to avoid introducing floating point arithmetics.

See also

[setSwipeAcceleration](#)

#### 7.155.4.18 handleDragEvent()

```
void handleDragEvent (
    const DragEvent & evt ) [virtual]
```

Handles the drag event described by evt.

Parameters

<i>evt</i>	The event.
------------	------------

Reimplemented from [Drawable](#).

Reimplemented in [ScrollWheelBase](#).

#### 7.155.4.19 handleGestureEvent()

```
void handleGestureEvent (
    const GestureEvent & evt ) [virtual]
```

Handles the gesture event described by evt.

Parameters

<i>evt</i>	The event.
------------	------------

Reimplemented from [Drawable](#).

Reimplemented in [ScrollWheelBase](#).

#### 7.155.4.20 handleTickEvent()

```
void handleTickEvent ( ) [virtual]
```

Handles the tick event.

Reimplemented from [Drawable](#).

#### 7.155.4.21 initialize()

```
void initialize ( ) [inline], [virtual]
```

Initializes the contents of all drawables.

Reimplemented in [ScrollWheelWithSelectionMode](#).

**7.155.4.22 isAnimating()**

```
bool isAnimating ( ) const
```

Query if this object is animating. This can be good to know if `getSelectedItem()` is called, as the result might not be as expected if [isAnimating\(\)](#) returns true.

**Returns**

true if animating, false if not.

**7.155.4.23 itemChanged()**

```
void itemChanged (
    int itemIndex ) [virtual]
```

Inform that an item has change and force all drawables with the given item index to be updated via the callback provided.

**Parameters**

<i>itemIndex</i>	Zero-based index of the changed item.
------------------	---------------------------------------

Reimplemented in [ScrollWheelWithSelectionStyle](#).

**7.155.4.24 keepOffsetInsideLimits()**

```
int32_t keepOffsetInsideLimits (
    int32_t newOffset,
    int16_t overShoot ) const [protected], [pure virtual]
```

**Parameters**

<i>newOffset</i>	The new offset.
<i>overShoot</i>	The over shoot.

**Returns**

An `int32_t`.

Implemented in [ScrollList](#), and [ScrollWheelBase](#).

**7.155.4.25 setAnimationEndedCallback()**

```
void setAnimationEndedCallback (
    GenericCallback<> & callback )
```

[Callback](#), called when the set animation ended.

**Parameters**

<i>in, out</i>	<i>callback</i>	The ended callback.
----------------	-----------------	---------------------

#### 7.155.4.26 setAnimationSteps()

```
void setAnimationSteps (
    int16_t steps )
```

Sets animation steps when moving to a new selected item. The default value is 30.

##### Parameters

<i>steps</i>	The animation steps.
--------------	----------------------

##### See also

[setEasingEquation](#)  
[getAnimationSteps](#)

#### 7.155.4.27 setCircular()

```
void setCircular (
    bool circular ) [virtual]
```

Sets whether the list is circular (infinite) or not. A circular list is a list where the first drawable re-appears after the last item in the list - and the last item in the list appears before the first item in the list.

##### Parameters

<i>circular</i>	True if the list should be circular, false if the list should not be circular.
-----------------	--------------------------------------------------------------------------------

##### See also

[DrawableList::setCircular](#)  
[getCircular](#)

Reimplemented in [ScrollWheelWithSelectionStyle](#).

#### 7.155.4.28 setDragAcceleration()

```
void setDragAcceleration (
    uint16_t acceleration )
```

Sets drag acceleration times 10, so "10" means "1", "15" means "1.5".

##### Parameters

<i>acceleration</i>	The drag acceleration. 10 makes the containers follow the finger, higher values makes the containers move faster.
---------------------	-------------------------------------------------------------------------------------------------------------------

**Note**

The reason for multiplying the acceleration by 10 is to avoid introducing floating point arithmetics.

**See also**

[getDragAcceleration](#)

**7.155.4.29 setDrawableSize()**

```
void setDrawableSize (
    int16_t drawableSize,
    int16_t drawableMargin )
```

Sets drawables size. The drawable is is the size of each drawable in the list (this is enforced by the [DrawableList](#) class). The spacing is the amount of blank to add between each drawable. Half of the space is placed before and half of the space is placed after the drawable. The entire size of an item is thus size + spacing.

**Parameters**

<i>drawableSize</i>	The size of the drawable.
<i>drawableMargin</i>	The margin around drawables (margin before and margin after).

**7.155.4.30 setEasingEquation()**

```
void setEasingEquation (
    EasingEquation equation )
```

Sets easing equation when changing the selected item, for example via swipe or [AnimateTo](#).

**Parameters**

<i>equation</i>	The equation.
-----------------	---------------

**See also**

[setAnimationSteps](#)  
[getAnimationSteps](#)

**7.155.4.31 setHeight()**

```
void setHeight (
    int16_t height ) [virtual]
```

Sets height of the [ScrollBase](#).

**Parameters**

<i>height</i>	The height. The height is propagated to the child(ren).
---------------	---------------------------------------------------------

Reimplemented from [Drawable](#).

Reimplemented in [ScrollWheelWithSelectionStyle](#).

#### 7.155.4.32 setHorizontal()

```
void setHorizontal (
    bool horizontal ) [virtual]
```

Sets a horizontal layout. If horizontal is set true, all drawables are arranged side by side. If horizontal is set false, the drawables are arranged above and below each other (vertically).

##### Parameters

<i>horizontal</i>	True to align drawables horizontal, false to align drawables vertically.
-------------------	--------------------------------------------------------------------------

##### Note

Default value is false, i.e. vertical layout.

##### See also

[DrawableList::setHorizontal](#)  
[getHorizontal](#)

Reimplemented in [ScrollWheelWithSelectionStyle](#).

#### 7.155.4.33 setItemPressedCallback()

```
void setItemPressedCallback (
    GenericCallback< int16_t > & callback )
```

Set [Callback](#) which will be called when a item is pressed.

##### Parameters

in	<i>callback</i>	The callback.
----	-----------------	---------------

#### 7.155.4.34 setItemSelectedCallback()

```
void setItemSelectedCallback (
    GenericCallback< int16_t > & callback )
```

Sets [Callback](#) which will be called when the selected item is clicked.

##### Parameters

in	<i>callback</i>	The callback.
----	-----------------	---------------



#### 7.155.4.35 setMaxSwipeItems()

```
void setMaxSwipeItems (
    uint16_t maxItems )
```

Sets maximum swipe items. Often useful when there are five visible items on the screen and a swipe action should at most swipe the next/previous five items into view to achieve sort of a paging effect.

##### Parameters

<i>maxItems</i>	The maximum items, 0 means "no limit".
-----------------	----------------------------------------

##### See also

[getMaxSwipeItems](#)

#### 7.155.4.36 setNumberOfItems()

```
void setNumberOfItems (
    int16_t items ) [virtual]
```

Sets number of items in the [DrawableList](#). This forces all drawables to be updated to ensure that the content is correct. For example a minute selector might could have 60 items (only some of which are visible at any given time).

##### Parameters

<i>items</i>	Number of items.
--------------	------------------

##### Note

The [DrawableList](#) is refreshed to reflect the change.

Reimplemented in [ScrollWheelWithSelectionStyle](#).

#### 7.155.4.37 setOffset()

```
void setOffset (
    int32_t offset ) [protected], [virtual]
```

Sets display offset of first item.

##### Parameters

<i>offset</i>	The offset.
---------------	-------------

Reimplemented in [ScrollWheelWithSelectionStyle](#).

#### 7.155.4.38 setSwipeAcceleration()

```
void setSwipeAcceleration (
```

```
uint16_t acceleration )
```

Sets swipe acceleration (times 10).

#### Parameters

<i>acceleration</i>	The acceleration times 10, so "60" means "6" and "75" means "7.5".
---------------------	--------------------------------------------------------------------

#### Note

The reason for multiplying the acceleration by 10 is to avoid introducing floating point arithmetics.

#### See also

[getSwipeAcceleration](#)

#### 7.155.4.39 setWidth()

```
void setWidth (
    int16_t width ) [virtual]
```

Sets width of the [ScrollBase](#). The width is propagated to the child(ren).

#### Parameters

<i>width</i>	The width.
--------------	------------

Reimplemented from [Drawable](#).

Reimplemented in [ScrollWheelWithSelectionStyle](#).

#### 7.155.4.40 stopAnimation()

```
void stopAnimation ( )
```

Stops an animation if one is ongoing.

## 7.156 ScrollList Class Reference

A scrolling menu of drawables.

```
#include <touchgfx/containers/scrollers/ScrollList.hpp>
```

### Public Member Functions

- [ScrollList](#) ()  
*Default constructor.*
- virtual [~ScrollList](#) ()  
*Destructor.*
- virtual void [setDrawables](#) ([DrawableListItemsInterface](#) &drawableListItems, [GenericCallback](#)< [Drawable](#)↔  
[ListItemsInterface](#) \*, int16\_t, int16\_t > &updateDrawableCallback)

- Sets the drawables parameters.*

  - void [setWindowSize](#) (int16\_t items)

*Sets window size.*
- void [setPadding](#) (int16\_t paddingBefore, int16\_t paddingAfter)

*Sets distance offset before and after drawables in the [ScrollList](#).*
- int16\_t [getPaddingBefore](#) () const

*Gets distance before first drawable in [ScrollList](#).*
- int16\_t [getPaddingAfter](#) () const

*Gets distance after last drawable in [ScrollList](#).*
- void [setSnapping](#) (bool snap)

*Sets snapping.*
- bool [getSnapping](#) () const

*Gets the current snap setting.*
- int16\_t [getItem](#) (int16\_t drawableIndex)

*Gets an item.*
- virtual void [handleClickEvent](#) (const [ClickEvent](#) &evt)

*Handles the click event described by evt.*

### Protected Member Functions

- virtual int32\_t [getPositionForItem](#) (int16\_t itemIndex)

*Gets position for an item.*
- virtual int32\_t [getNearestAlignedOffset](#) (int32\_t offset) const

*Gets nearest aligned offset.*
- virtual int32\_t [keepOffsetInsideLimits](#) (int32\_t newOffset, int16\_t overShoot) const

*Keep offset inside limits.*

### Protected Attributes

- int16\_t [paddingAfterLastItem](#)

*The distance after last item.*
- bool [snapping](#)

*True to snapping.*
- int [windowSize](#)

*Size of the window.*

### Additional Inherited Members

#### 7.156.1 Detailed Description

A scrolling menu of drawables. To preserve resources, a lot of items can be displayed using only a few drawables. To achieve this, please see [DrawableList](#).

#### 7.156.2 Constructor & Destructor Documentation

### 7.156.2.1 ScrollList()

```
ScrollList ( )
```

Default constructor.

### 7.156.2.2 ~ScrollList()

```
~ScrollList ( ) [inline], [virtual]
```

Destructor.

## 7.156.3 Member Function Documentation

### 7.156.3.1 getItem()

```
int16_t getItem (
    int16_t drawableIndex ) [inline]
```

Gets an item.

#### Parameters

<i>drawableIndex</i>	Zero-based index of the drawable.
----------------------	-----------------------------------

#### Returns

The item.

### 7.156.3.2 getNearestAlignedOffset()

```
int32_t getNearestAlignedOffset (
    int32_t offset ) const [protected], [virtual]
```

#### Parameters

<i>offset</i>	The offset.
---------------	-------------

#### Returns

The nearest aligned offset.

Reimplemented from [ScrollBase](#).

### 7.156.3.3 getPaddingAfter()

```
int16_t getPaddingAfter ( ) const
```

Gets distance after last drawable in [ScrollList](#).

**Returns**

The distance after the last drawable in the [ScrollList](#).

**See also**

[setPadding](#)  
[getPaddingBefore](#)

**7.156.3.4 getPaddingBefore()**

```
int16_t getPaddingBefore ( ) const
```

Gets distance before first drawable in [ScrollList](#).

**Returns**

The distance before.

**See also**

[setPadding](#)  
[getPaddingAfter](#)

**7.156.3.5 getPositionForItem()**

```
int32_t getPositionForItem (
    int16_t itemIndex ) [protected], [virtual]
```

Get the position for an item. The position should ensure that the item is in view as defined by the semantics of the actual scroll class. If the item is already in view, the current offset is returned and not the offset of the given item.

**Parameters**

<i>itemIndex</i>	Zero-based index of the item.
------------------	-------------------------------

**Returns**

The position for item.

Implements [ScrollBase](#).

**7.156.3.6 getSnapping()**

```
bool getSnapping ( ) const
```

Gets the current snap setting.

**Returns**

true if snapping is set, false otherwise.

### 7.156.3.7 handleClickEvent()

```
void handleClickEvent (
    const ClickEvent & evt ) [virtual]
```

#### Parameters

<i>evt</i>	The event.
------------	------------

Reimplemented from [Drawable](#).

### 7.156.3.8 keepOffsetInsideLimits()

```
int32_t keepOffsetInsideLimits (
    int32_t newOffset,
    int16_t overShoot ) const [protected], [virtual]
```

#### Parameters

<i>newOffset</i>	The new offset.
<i>overShoot</i>	The over shoot.

#### Returns

An int32\_t.

Implements [ScrollBase](#).

### 7.156.3.9 setDrawables()

```
void setDrawables (
    DrawableListItemsInterface & drawableListItems,
    GenericCallback< DrawableListItemsInterface *, int16_t, int16_t > & update↵
    DrawableCallback ) [virtual]
```

Sets the drawables parameters. These parameters are

- The access class to the array of drawables
- [Callback](#) to update the contents of a drawable.

#### Parameters

in, out	<i>drawableListItems</i>	Number of drawables allocated.
in	<i>updateDrawableCallback</i>	A callback to update the contents of a drawable.

#### See also

[DrawableList::setDrawables](#)

### 7.156.3.10 setPadding()

```
void setPadding (
    int16_t distanceBefore,
    int16_t distanceAfter )
```

Sets distance offset before and after the "visible" drawables in the [ScrollList](#). This allows the actual area where widgets are placed to have a little extra area where parts of drawables can be seen. For example if the [ScrollList](#) is 200, each drawable is 50 and distance before and distance after are 25, then there is room for three visible drawables inside the [ScrollList](#). When scrolling, part of the scrolled out drawables can be seen before and after the three drawables. Actually  $25/50 = 50\%$  of a drawable can be seen before and after the three drawables in the [ScrollList](#).

#### Parameters

<i>distanceBefore</i>	The distance before the first drawable in the <a href="#">ScrollList</a> .
<i>distanceAfter</i>	The distance after the last drawable in the <a href="#">ScrollList</a> .

#### See also

[getPaddingBefore](#)  
[getPaddingAfter](#)

### 7.156.3.11 setSnapping()

```
void setSnapping (
    bool snap )
```

Set snapping. If snapping is false, the items can flow freely. If snapping is true, the items will snap into place so an item is always in the "selected" spot.

#### Parameters

<i>snap</i>	true to snap.
-------------	---------------

### 7.156.3.12 setWindowSize()

```
void setWindowSize (
    int16_t items )
```

Sets window size. This is the number of items that should always be visible. The default value is 1.

#### Parameters

<i>items</i>	The items.
--------------	------------

**Note**

This only applies to non-circular lists.

## 7.157 ScrollWheel Class Reference

A scroll wheel.

```
#include <touchgfx/containers/scrollers/ScrollWheel.hpp>
```

### Public Member Functions

- [ScrollWheel](#) ()  
*Default constructor.*
- virtual [~ScrollWheel](#) ()  
*Destructor.*
- virtual void [setDrawables](#) ([DrawableListItemsInterface](#) &drawableListItems, [GenericCallback](#)< [DrawableListItemsInterface](#) \*, int16\_t, int16\_t> &updateDrawableCallback)  
*Sets the drawables.*

### Additional Inherited Members

#### 7.157.1 Detailed Description

A scroll wheel is a list of identically styled drawables which can be scrolled through. One of the items in the list is the "selected" one, and scrolling through the list can be done in various ways. The [ScrollWheel](#) uses the [DrawableList](#) to make it possible to handle a huge number of items using only a limited number of drawables by reusing drawables that are no longer in view.

See also

[DrawableList](#)  
[ScrollWheelWithSelectionMode](#)

#### 7.157.2 Constructor & Destructor Documentation

##### 7.157.2.1 ScrollWheel()

```
ScrollWheel ( )
```

Default constructor.

##### 7.157.2.2 ~ScrollWheel()

```
~ScrollWheel ( ) [virtual]
```

Destructor.

#### 7.157.3 Member Function Documentation



## 7.157.3.1 setDrawables()

```
void setDrawables (
    DrawableListItemsInterface & drawableListItems,
    GenericCallback< DrawableListItemsInterface *, int16_t, int16_t > & update←
DrawableCallback ) [virtual]
```

Sets the drawables used by the scroll wheel. The drawables are accessed through a callback that will return the needed drawable and another callback that will put the right data in the drawable.

## Parameters

in, out	<i>drawableListItems</i>	Number of drawables.
in, out	<i>updateDrawableCallback</i>	The update drawable callback.

## 7.158 ScrollWheelBase Class Reference

A scroll wheel base class.

```
#include <touchgfx/containers/scrollers/ScrollWheelBase.hpp>
```

## Public Member Functions

- [ScrollWheelBase](#) ()  
*Default constructor.*
- virtual [~ScrollWheelBase](#) ()  
*Destructor.*
- virtual void [setSelectedItemOffset](#) (int16\_t offset)  
*Sets selected item offset.*
- virtual int16\_t [getSelectedItemOffset](#) () const  
*Gets selected item offset.*
- int [getSelectedItem](#) () const  
*Gets selected item.*
- virtual int32\_t [keepOffsetInsideLimits](#) (int32\_t newOffset, int16\_t overShoot) const  
*Keep offset inside limits.*
- virtual void [handleClickEvent](#) (const [ClickEvent](#) &evt)  
*Handles the click event described by evt.*
- virtual void [handleDragEvent](#) (const [DragEvent](#) &evt)  
*Handles the drag event described by evt.*
- virtual void [handleGestureEvent](#) (const [GestureEvent](#) &evt)  
*Handles the gesture event described by evt.*
- void [setAnimateToCallback](#) ([GenericCallback](#)< int16\_t > &callback)  
*Sets [Callback](#) which will be called when the [ScrollWheel](#) animates to a new item.*

## Protected Member Functions

- virtual int32\_t [getPositionForItem](#) (int16\_t itemIndex)  
*Gets position for an item.*
- virtual void [animateToPosition](#) (int32\_t position, int16\_t steps=-1)  
*Animate to a new position/offset using the given number of steps.*

## Protected Attributes

- [GenericCallback](#)< int16\_t > \* [animateToCallback](#)

*The animate to callback.*

## Additional Inherited Members

### 7.158.1 Detailed Description

A scroll wheel base class. Used by [ScrollWheel](#) and [ScrollWheelWithHighlight](#).

See also

[ScrollWheel](#)

[ScrollWheelWithHighlight](#)

### 7.158.2 Constructor & Destructor Documentation

#### 7.158.2.1 ScrollWheelBase()

```
ScrollWheelBase ( )
```

Default constructor.

#### 7.158.2.2 ~ScrollWheelBase()

```
~ScrollWheelBase ( ) [inline], [virtual]
```

Destructor.

### 7.158.3 Member Function Documentation

#### 7.158.3.1 animateToPosition()

```
void animateToPosition (
    int32_t position,
    int16_t steps = -1 ) [protected], [virtual]
```

Animate to a new position/offset using the given number of steps.

#### Parameters

<i>position</i>	The new position.
<i>steps</i>	(Optional) The number of steps. If steps is <0, the default number of steps is used (this is the default). If steps ==0, there will be no animation, simply a direct skip to the given position.

Reimplemented from [ScrollBase](#).

### 7.158.3.2 getPositionForItem()

```
int32_t getPositionForItem (
    int16_t itemIndex ) [protected], [virtual]
```

Get the position for an item. The position should ensure that the item is in view as defined by the semantics of the actual scroll class.

#### Parameters

<i>itemIndex</i>	Zero-based index of the item.
------------------	-------------------------------

#### Returns

The position for item.

Implements [ScrollBase](#).

### 7.158.3.3 getSelectedItem()

```
int getSelectedItem ( ) const
```

Gets selected item. If an animation is in progress, the item that is being scrolled to is returned, not the item that happens to be flying by at the time.

#### Returns

The selected item.

### 7.158.3.4 getSelectedItemOffset()

```
int16_t getSelectedItemOffset ( ) const [virtual]
```

Gets offset of selected item measured in pixels relative to the start of the widget.

#### Returns

The selected item offset.

#### See also

[setSelectedItemOffset](#)

### 7.158.3.5 handleClickEvent()

```
void handleClickEvent (
    const ClickEvent & evt ) [virtual]
```

Handles the click event described by evt.

#### Parameters

<i>evt</i>	The event.
------------	------------

Reimplemented from [Drawable](#).

#### 7.158.3.6 handleDragEvent()

```
void handleDragEvent (
    const DragEvent & evt ) [virtual]
```

Handles the drag event described by evt.

##### Parameters

<i>evt</i>	The event.
------------	------------

Reimplemented from [ScrollBase](#).

#### 7.158.3.7 handleGestureEvent()

```
void handleGestureEvent (
    const GestureEvent & evt ) [virtual]
```

Handles the gesture event described by evt.

##### Parameters

<i>evt</i>	The event.
------------	------------

Reimplemented from [ScrollBase](#).

#### 7.158.3.8 keepOffsetInsideLimits()

```
int32_t keepOffsetInsideLimits (
    int32_t newOffset,
    int16_t overShoot ) const [virtual]
```

##### Parameters

<i>newOffset</i>	The new offset.
<i>overShoot</i>	The over shoot.

##### Returns

An int32\_t.

Implements [ScrollBase](#).

#### 7.158.3.9 setAnimateToCallback()

```
void setAnimateToCallback (
    GenericCallback< int16_t > & callback )
```

Sets [Callback](#) which will be called when the [ScrollWheel](#) animates to a new item.

#### Parameters

in	<i>callback</i>	The callback.
----	-----------------	---------------

#### 7.158.3.10 setSelectedItemOffset()

```
void setSelectedItemOffset (
    int16_t offset ) [virtual]
```

Sets selected item offset. This is the number of pixels from the start of the widget where the selected item is placed on screen. The offset is the relative x coordinate if the [ScrollWheel](#) is horizontal, otherwise it is the relative y coordinate. If this value is zero, the selected item is placed at the very start of the widget.

#### Parameters

<i>offset</i>	The offset.
---------------	-------------

## 7.159 ScrollWheelWithSelectionStyle Class Reference

A scroll wheel with selection style.

```
#include <touchgfx/containers/scrollers/ScrollWheelWithSelectionStyle.hpp>
```

### Public Member Functions

- [ScrollWheelWithSelectionStyle](#) ()  
*Default constructor.*
- virtual [~ScrollWheelWithSelectionStyle](#) ()  
*Destructor.*
- virtual void [setWidth](#) (int16\_t width)  
*Sets the width.*
- virtual void [setHeight](#) (int16\_t height)  
*Sets the height.*
- virtual void [setHorizontal](#) (bool horizontal)  
*Sets whether the scroll wheel is horizontal or vertical.*
- virtual void [setCircular](#) (bool circular)  
*Sets whether the scroll wheel is circular.*
- virtual void [setNumberOfItems](#) (int16\_t numberOfItems)  
*Sets number of items in the scroll wheel.*
- virtual void [setSelectedItemOffset](#) (int16\_t offset)  
*Sets selected item offset.*
- virtual void [setSelectedItemExtraSize](#) (int16\_t extraSizeBefore, int16\_t extraSizeAfter)  
*Sets selected item extra size.*
- virtual int16\_t [getSelectedItemExtraSizeBefore](#) () const  
*Gets selected item extra size before.*
- virtual int16\_t [getSelectedItemExtraSizeAfter](#) () const  
*Gets selected item extra size after.*

- virtual void [setSelectedItemMargin](#) (int16\_t marginBefore, int16\_t marginAfter)  
*Sets margin around selected item.*
- virtual int16\_t [getSelectedItemMarginBefore](#) () const  
*Gets selected item margin before.*
- virtual int16\_t [getSelectedItemMarginAfter](#) () const  
*Gets selected item margin after.*
- virtual void [setSelectedItemPosition](#) (int16\_t offset, int16\_t extraSizeBefore, int16\_t extraSizeAfter, int16\_t marginBefore, int16\_t marginAfter)  
*Sets the selected item offset.*
- virtual void [setDrawableSize](#) (int16\_t drawableSize, int16\_t drawableMargin)  
*Sets drawable size.*
- virtual void [setDrawables](#) (DrawableListItemsInterface &drawableListItems, GenericCallback< DrawableListItemsInterface \*, int16\_t, int16\_t > &updateDrawableCallback, DrawableListItemsInterface &centerDrawableListItems, GenericCallback< DrawableListItemsInterface \*, int16\_t, int16\_t > &updateCenterDrawableCallback)  
*Setups the widget.*
- virtual void [itemChanged](#) (int itemIndex)  
*Item changed.*
- virtual void [initialize](#) ()  
*Initializes the contents of all drawables.*

## Protected Member Functions

- virtual void [setOffset](#) (int32\_t offset)  
*Sets offset of item 0 relative to the selected item's position.*
- void [refreshDrawableListsLayout](#) ()  
*Refresh drawable lists layout.*

## Protected Attributes

- int16\_t [drawablesInFirstList](#)  
*List of drawables in firsts.*
- [DrawableList](#) list1  
*The center list.*
- [DrawableList](#) list2  
*The last list.*
- int16\_t [extraSizeBeforeSelectedItem](#)  
*The distance before selected item.*
- int16\_t [extraSizeAfterSelectedItem](#)  
*The distance after selected item.*
- int16\_t [marginBeforeSelectedItem](#)  
*The distance before selected item.*
- int16\_t [marginAfterSelectedItem](#)  
*The distance after selected item.*
- [DrawableListItemsInterface](#) \* [drawables](#)  
*The drawables at the beginning and end of the scroll wheel.*
- [DrawableListItemsInterface](#) \* [centerDrawables](#)  
*The drawables at the center of the scroll wheel.*
- [GenericCallback](#)< [DrawableListItemsInterface](#) \*, int16\_t, int16\_t > \* [originalUpdateDrawableCallback](#)  
*The original update drawable callback.*
- [GenericCallback](#)< [DrawableListItemsInterface](#) \*, int16\_t, int16\_t > \* [originalUpdateCenterDrawableCallback](#)  
*The original update center drawable callback.*

## Additional Inherited Members

### 7.159.1 Detailed Description

A scroll wheel with selection style. Similar to an ordinary [ScrollWheel](#), but with a different style for the selected item which can thus be bold, have a different color or similar effect to highlight it.

See also

[DrawableList](#)  
[ScrollWheel](#)

### 7.159.2 Constructor & Destructor Documentation

#### 7.159.2.1 ScrollWheelWithSelectionMode()

```
ScrollWheelWithSelectionMode ( )
```

Default constructor.

#### 7.159.2.2 ~ScrollWheelWithSelectionMode()

```
~ScrollWheelWithSelectionMode ( ) [inline], [virtual]
```

Destructor.

### 7.159.3 Member Function Documentation

#### 7.159.3.1 getSelectedItemExtraSizeAfter()

```
int16_t getSelectedItemExtraSizeAfter ( ) const [virtual]
```

Gets selected item extra size after.

Returns

The selected item extra size after.

See also

[setSelectedItemExtraSize](#)

#### 7.159.3.2 getSelectedItemExtraSizeBefore()

```
int16_t getSelectedItemExtraSizeBefore ( ) const [virtual]
```

Gets selected item extra size before.

**Returns**

The selected item extra size before.

**See also**

[setSelectedItemExtraSize](#)

**7.159.3.3 `getSelectedItemMarginAfter()`**

```
int16_t getItemMarginAfter ( ) const [virtual]
```

Gets selected item margin after.

**Returns**

The selected item margin after.

**See also**

[setSelectedItemMargin](#)

**7.159.3.4 `getSelectedItemMarginBefore()`**

```
int16_t getItemMarginBefore ( ) const [virtual]
```

Gets selected item margin before.

**Returns**

The selected item margin before.

**See also**

[setSelectedItemMargin](#)

**7.159.3.5 `initialize()`**

```
void initialize ( ) [inline], [virtual]
```

Initializes the contents of all drawables.

Reimplemented from [ScrollBase](#).

**7.159.3.6 `itemChanged()`**

```
void itemChanged (
    int itemIndex ) [virtual]
```

Inform that an item has change and force all drawables with the given item index to be updated via the callback provided.



## Parameters

<i>itemIndex</i>	Zero-based index of the changed item.
------------------	---------------------------------------

Reimplemented from [ScrollBase](#).

## 7.159.3.7 refreshDrawableListsLayout()

```
void refreshDrawableListsLayout ( ) [protected]
```

Refresh drawable lists layout. Ensure that the three DrawableLists are places correctly and setup properly. This is typically done after the [ScrollWheelWithSelectionMode](#) has been resized or the size of the selected item is changed.

## 7.159.3.8 setCircular()

```
void setCircular (
    bool circular ) [virtual]
```

Sets whether the scroll wheel is circular. IF the scroll wheel is circular, it can be scrolled infinitely so that the last item appears before the first item, just like the first item appears after the last item in the list.

## Parameters

<i>circular</i>	True to make the scroll wheel circular.
-----------------	-----------------------------------------

## See also

[getCircular](#)

Reimplemented from [ScrollBase](#).

## 7.159.3.9 setDrawables()

```
void setDrawables (
    DrawableListItemsInterface & drawableListItems,
    GenericCallback< DrawableListItemsInterface *, int16_t, int16_t > & update↵
DrawableCallback,
    DrawableListItemsInterface & centerDrawableListItems,
    GenericCallback< DrawableListItemsInterface *, int16_t, int16_t > & update↵
CenterDrawableCallback ) [virtual]
```

Setups the widget. Numerous parameters control the position of the widget, the two scroll lists inside and the values in them.

## Parameters

in, out	<i>drawableListItems</i>	Number of drawables in outer array.
in	<i>updateDrawableCallback</i>	The callback to update a drawable.
in, out	<i>centerDrawableListItems</i>	Number of drawables in center array.
in	<i>updateCenterDrawableCallback</i>	The callback to update a center drawable.

#### 7.159.3.10 setDrawableSize()

```
void setDrawableSize (
    int16_t drawableSize,
    int16_t drawableMargin ) [virtual]
```

Sets drawable size. Each item in the scroll wheel will have a size of the sum of the two numbers, where the `drawableSize` is the size of the drawable in the list and the `drawableMargin` is the margin between each drawable (half of which is placed before the drawable, the rest is placed after the drawable).

##### Parameters

<i>drawableSize</i>	Size of the drawable.
<i>drawableMargin</i>	The drawable margin.

##### See also

[getDrawableSize](#)  
[getDrawableMargin](#)  
[getItemSize](#)

#### 7.159.3.11 setHeight()

```
void setHeight (
    int16_t height ) [virtual]
```

Sets the height. If the scroll wheel is horizontal, the height is propagated to all the drawables.

##### Parameters

<i>height</i>	The height.
---------------	-------------

Reimplemented from [ScrollBase](#).

#### 7.159.3.12 setHorizontal()

```
void setHorizontal (
    bool horizontal ) [virtual]
```

Sets whether the scroll wheel is horizontal or vertical. If the scroll wheel is horizontal, the items are arranged side by side, otherwise they are arranged above and below each other.

##### Parameters

<i>horizontal</i>	True means horizontal, false means vertical.
-------------------	----------------------------------------------

See also

[getHorizontal](#)

Reimplemented from [ScrollBase](#).

#### 7.159.3.13 setNumberOfItems()

```
void setNumberOfItems (
    int16_t numberOfItems ) [virtual]
```

Sets number of items in the scroll wheel. The scroll wheel is refreshed to ensure that everything is displayed properly on the screen.

Parameters

<i>numberOfItems</i>	Number of items.
----------------------	------------------

See also

[getNumberOfItems](#)

Reimplemented from [ScrollBase](#).

#### 7.159.3.14 setOffset()

```
void setOffset (
    int32_t offset ) [protected], [virtual]
```

Sets offset of item 0 relative to the selected item's position.

Parameters

<i>offset</i>	The offset.
---------------	-------------

See also

[getOffset](#)

Reimplemented from [ScrollBase](#).

#### 7.159.3.15 setSelectedItemExtraSize()

```
void setSelectedItemExtraSize (
    int16_t extraSizeBefore,
    int16_t extraSizeAfter ) [virtual]
```

Sets selected item extra size to make the size of the area for the center drawables larger.

Parameters

<i>extraSizeBefore</i>	The extra size before.
<i>extraSizeAfter</i>	The extra size after.

See also

[setSelectedItemOffset](#)

#### 7.159.3.16 setSelectedItemMargin()

```
void setSelectedItemMargin (
    int16_t marginBefore,
    int16_t marginAfter ) [virtual]
```

Sets margin around selected item. This like an invisible area added before and after the selected item (including extra size).

Parameters

<i>marginBefore</i>	The margin before.
<i>marginAfter</i>	The margin after.

See also

[setSelectedItemOffset](#)  
[setSelectedItemExtraSize](#)

#### 7.159.3.17 setSelectedItemPosition()

```
void setSelectedItemPosition (
    int16_t offset,
    int16_t extraSizeBefore,
    int16_t extraSizeAfter,
    int16_t marginBefore,
    int16_t marginAfter ) [virtual]
```

Sets the selected item offset. This is the distance from the beginning of the [ScrollWheel](#) measured in pixels. The distance before and after that should also be drawn using the center drawables - for example to extend area of emphasized elements - can also be specified. Further, if a gap is needed between the "normal" drawables and the center drawables - for example to give the illusion that that items disappear under a graphical element, only to appear in the center.

This is a combination of [setSelectedItemOffset](#), [setSelectedItemExtraSize](#) and [setSelectedItemMargin](#).

Parameters

<i>offset</i>	The offset of the selected item.
<i>extraSizeBefore</i>	The extra size before the selected item.
<i>extraSizeAfter</i>	The extra size after the selected item.
<i>marginBefore</i>	The margin before the selected item.
<i>marginAfter</i>	The margin after the selected item.

See also

[setSelectedItemOffset](#)  
[setSelectedItemExtraSize](#)  
[setSelectedItemMargin](#)

## 7.159.3.18 setSelectItemOffset()

```
void setSelectItemOffset (
    int16_t offset ) [virtual]
```

Sets selected item offset. This is the number of pixels from the start of the widget where the selected item is placed on screen. The offset is the relative x coordinate if the [ScrollWheel](#) is horizontal, otherwise it is the relative y coordinate. If this value is zero, the selected item is placed at the very start of the widget.

## Parameters

<i>offset</i>	The offset.
---------------	-------------

## 7.159.3.19 setWidth()

```
void setWidth (
    int16_t width ) [virtual]
```

Sets the width. If the scroll wheel is vertical, the width is propagated to all the drawables.

## Parameters

<i>width</i>	The width.
--------------	------------

Reimplemented from [ScrollBase](#).

## 7.160 SDL2TouchController Class Reference

[TouchController](#) for the simulator.

```
#include <platform/driver/touch/SDL2TouchController.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.*

## 7.160.1 Detailed Description

[TouchController](#) for the simulator.

See also

[TouchController](#)

## 7.160.2 Member Function Documentation

### 7.160.2.1 init()

```
void init ( ) [virtual]
```

Initializes touch controller.

Implements [TouchController](#).

### 7.160.2.2 sampleTouch()

```
bool sampleTouch (
    int32_t & x,
    int32_t & y ) [virtual]
```

Checks whether the touch screen is being touched, and if so, what coordinates.

#### Parameters

out	<i>x</i>	The x position of the touch
out	<i>y</i>	The y position of the touch

#### Returns

True if a touch has been detected, otherwise false.

Implements [TouchController](#).

## 7.161 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.*

### 7.161.1 Detailed Description

[TouchController](#) for the simulator.

#### See also

[TouchController](#)

### 7.161.2 Member Function Documentation

## 7.161.2.1 init()

```
void init ( ) [virtual]
```

Initializes touch controller.

Implements [TouchController](#).

## 7.161.2.2 sampleTouch()

```
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](#).

## 7.162 Shape&lt; POINTS &gt; 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.*

## Protected Member Functions

- 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

### 7.162.1 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

<i>POINTS</i>	The number of points in the given shape.
<i>T</i>	The type of the points used for the shape. Must be int or float.

See also

[CanvasWidget](#)

### 7.162.2 Constructor & Destructor Documentation

#### 7.162.2.1 ~Shape()

```
~Shape ( ) [inline], [virtual]
```

Virtual Destructor.

### 7.162.3 Member Function Documentation

#### 7.162.3.1 getCacheX()

```
CWRUtil::Q5 getCacheX (
    int i ) const [inline], [protected], [virtual]
```

Gets cached x coordinate of a corner.

#### Parameters

<i>i</i>	Zero-based index of the corner.
----------	---------------------------------



**Returns**

The cached x coordinate.

Implements [AbstractShape](#).

**7.162.3.2 getCacheY()**

```
CWRUtil::Q5 getCacheY (  
    int i ) const [inline], [protected], [virtual]
```

Gets cached y coordinate of a corner.

**Parameters**

<i>i</i>	Zero-based index of the corner.
----------	---------------------------------

**Returns**

The cached y coordinate.

Implements [AbstractShape](#).

**7.162.3.3 getCornerX()**

```
CWRUtil::Q5 getCornerX (  
    int i ) const [inline], [virtual]
```

Gets the x coordinate of a corner.

**Parameters**

<i>i</i>	Zero-based index of the corner.
----------	---------------------------------

**Returns**

The corner x coordinate.

Implements [AbstractShape](#).

**7.162.3.4 getCornerY()**

```
CWRUtil::Q5 getCornerY (  
    int i ) const [inline], [virtual]
```

Gets the y coordinate of a corner.

**Parameters**

<i>i</i>	Zero-based index of the corner.
----------	---------------------------------

**Returns**

The corner y coordinate.

Implements [AbstractShape](#).

**7.162.3.5 getNumPoints()**

```
int getNumPoints ( ) const [inline], [virtual]
```

Gets number points used to make up the shape.

**Returns**

The number points.

Implements [AbstractShape](#).

**7.162.3.6 setCache()**

```
void setCache (
    int i,
    CWRUtil::Q5 x,
    CWRUtil::Q5 y ) [inline], [protected], [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>i</i>	Zero-based index of the corner.
<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

Implements [AbstractShape](#).

**7.162.3.7 setCorner()**

```
void setCorner (
    int i,
    CWRUtil::Q5 x,
    CWRUtil::Q5 y ) [inline], [virtual]
```

Sets a corner of the shape in Q5 format.

**Parameters**

<i>i</i>	Zero-based index of the corner.
<i>x</i>	The x coordinate in Q5 format.
<i>y</i>	The y coordinate in Q5 format.

Implements [AbstractShape](#).

## 7.163 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.*

### 7.163.1 Detailed Description

```
template<typename 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

<a href="#">T</a>	Generic type parameter, either int or float.
-------------------	----------------------------------------------

See also

[setAbstractShape\(\)](#)

## 7.164 SlideMenu Class Reference

[SlideMenu](#) is a container that has the functionality of being either collapsed or expanded.

```
#include <touchgfx/containers/SlideMenu.hpp>
```

### Public Types

- enum [State](#) { **COLLAPSED**, **EXPANDED** }  
*Values that represent the [SlideMenu](#) states.*
- enum [ExpandDirection](#) { **SOUTH**, **NORTH**, **EAST**, **WEST** }  
*Values that represent the expand directions.*

### Public Member Functions

- [SlideMenu](#) ()  
*Default constructor.*
- virtual [~SlideMenu](#) ()  
*Destructor.*
- virtual void [setup](#) ([SlideMenu::ExpandDirection](#) newExpandDirection, const [Bitmap](#) &backgroundBMP, const [Bitmap](#) &stateChangeButtonBMP, const [Bitmap](#) &stateChangeButtonPressedBMP)

Setup the [SlideMenu](#) by positioning the `stateChangeButton` next to background image relative to the expand direction and center it in the other dimension.

- virtual void [setup](#) ([SlideMenu::ExpandDirection](#) newExpandDirection, const [Bitmap](#) &backgroundBMP, const [Bitmap](#) &stateChangeButtonBMP, const [Bitmap](#) &stateChangeButtonPressedBMP, int16\_t backgroundX, int16\_t backgroundY, int16\_t stateChangeButtonX, int16\_t stateChangeButtonY)

Setup method for the [SlideMenu](#). Positioning of the background image and the `stateChangeButton` is done by stating the X and Y coordinates for the elements (relative to the [SlideMenu](#)).

- virtual void [setExpandDirection](#) ([SlideMenu::ExpandDirection](#) newExpandDirection)

Sets the expand direction.

- virtual [SlideMenu::ExpandDirection](#) [getExpandDirection](#) () const

Gets the expand direction.

- virtual void [setVisiblePixelsWhenCollapsed](#) (int16\_t visiblePixels)

Sets the amount of visible pixels when collapsed.

- virtual int16\_t [setVisiblePixelsWhenCollapsed](#) () const

Gets the visible pixels when collapsed.

- virtual void [setHiddenPixelsWhenExpanded](#) (int16\_t hiddenPixels)

Sets the amount of hidden pixels when expanded.

- virtual int16\_t [getHiddenPixelsWhenExpanded](#) () const

Gets the hidden pixels when expanded.

- virtual void [setExpandedStateTimeout](#) (uint16\_t timeout)

Sets the expanded state timeout in ticks.

- virtual uint16\_t [getExpandedStateTimeout](#) () const

Gets expanded state timeout.

- virtual void [setAnimationDuration](#) (uint16\_t duration)

Sets the animation duration.

- virtual uint16\_t [getAnimationDuration](#) () const

Gets the animation duration.

- virtual void [setAnimationEasingEquation](#) ([EasingEquation](#) animationEasingEquation)

Sets the animation easing equation.

- virtual [EasingEquation](#) [getAnimationEasingEquation](#) () const

Gets the animation easing equation.

- virtual void [setState](#) ([SlideMenu::State](#) newState)

Sets the state of the [SlideMenu](#). No animation is performed.

- virtual void [animateToState](#) ([SlideMenu::State](#) newState)

Animate to the given state.

- virtual [SlideMenu::State](#) [getState](#) ()

Gets the current state.

- virtual void [resetExpandedStateTimer](#) ()

Resets the expanded state timer.

- virtual uint16\_t [getExpandedStateTimer](#) () const

Gets the expanded state timer.

- virtual int16\_t [getBackgroundX](#) () const

Gets the background x coordinate.

- virtual int16\_t [getBackgroundY](#) () const

Gets the background y coordinate.

- virtual int16\_t [getStateChangeButtonX](#) () const

Gets the state change button x coordinate.

- virtual int16\_t [getStateChangeButtonY](#) () const

Gets the state change button y coordinate.

- virtual void [setStateChangedCallback](#) ([GenericCallback](#)< const [SlideMenu](#) & > &callback)

Set the state changed callback. This callback is called when the state change button is clicked.

- virtual void [setStateChangedAnimationEndedCallback](#) ([GenericCallback](#)< const [SlideMenu](#) & > &callback)  
*Set the state change animation ended callback. This callback is called when a state change animation has ended.*
- virtual void [add](#) ([Drawable](#) &d)  
*Adds a drawable to the container.*
- virtual void [remove](#) ([Drawable](#) &d)  
*Removes the drawable from the container.*

### Protected Member Functions

- void [stateChangeButtonClickedHandler](#) (const [AbstractButton](#) &button)  
*Handler for the state change button clicked event.*
- void [animationEndedHandler](#) (const [MoveAnimator](#)< [Container](#) > &container)  
*Handler for the state change animation ended event.*
- virtual void [handleTickEvent](#) ()  
*Handles the tick event.*
- virtual int16\_t [getCollapsedXCoordinate](#) ()  
*Gets the x coordinate for the collapsed state.*
- virtual int16\_t [getCollapsedYCoordinate](#) ()  
*Gets the y coordinate for the collapsed state.*
- virtual int16\_t [getExpandedXCoordinate](#) ()  
*Gets the x coordinate for the expanded state.*
- virtual int16\_t [getExpandedYCoordinate](#) ()  
*Gets the y coordinate for the expanded state.*

### Protected Attributes

- [MoveAnimator](#)< [Container](#) > [menuContainer](#)  
*The container holding the actual menu items. This is the container that performs the state change animation.*
- [Button](#) [stateChangeButton](#)  
*The state change button that toggles the [SlideMenu](#) state.*
- [Image](#) [background](#)  
*The background of the [SlideMenu](#).*
- [Callback](#)< [SlideMenu](#), const [AbstractButton](#) & > [onStateChangeButtonClicked](#)  
*The local state changed button clicked callback.*
- [Callback](#)< [SlideMenu](#), const [MoveAnimator](#)< [Container](#) > & > [animationEndedCallback](#)  
*The local state changed animation ended callback.*
- [GenericCallback](#)< const [SlideMenu](#) &> \* [stateChangedCallback](#)  
*The public state changed button clicked callback.*
- [GenericCallback](#)< const [SlideMenu](#) &> \* [stateChangedAnimationEndedCallback](#)  
*The public state changed animation ended callback.*
- [SlideMenu::State](#) [currentState](#)  
*The current state of the [SlideMenu](#).*
- [SlideMenu::ExpandDirection](#) [expandDirection](#)  
*The expand direction of the [SlideMenu](#).*
- [EasingEquation](#) [animationEquation](#)  
*The easing equation used for the state change animation.*
- int16\_t [visiblePixelsWhenCollapsed](#)  
*The number of visible pixels when collapsed.*
- int16\_t [hiddenPixelsWhenExpanded](#)  
*The number of hidden pixels when expanded.*

- uint16\_t [expandedStateTimeout](#)  
*The expanded state timeout.*
- uint16\_t [expandedStateTimer](#)  
*The timer that counts towards the expandedStateTimeout. If reached the [SlideMenu](#) will animate to COLLAPSED.*
- uint16\_t [animationDuration](#)  
*The animation duration of the state change animation.*

## Additional Inherited Members

### 7.164.1 Detailed Description

[SlideMenu](#) is a container that has the functionality of being either collapsed or expanded. The [SlideMenu](#) consists of a background and a activate button that toggles the SlideMenus collapsed/expanded state.

The relative positions of the background and state change button is configurable as is the direction in which the [SlideMenu](#) expands and collapses. How much of the [SlideMenu](#) that is visible when collapsed can be set with the setVisiblePixelsWhenCollapsed(..) method. It is, of course, important that the state change button is accessible when collapsed. The [SlideMenu](#) will animate back to the collapsed state after a expandedStateTimeout is reached. The timer can be reset, for example when the user interacts with elements in the list. Use the resetExpanded↔StateTimer(..) method for this.

Actual menu elements are added normally using the add(..) method and are positioned relative to the [SlideMenu](#).

### 7.164.2 Member Function Documentation

#### 7.164.2.1 add()

```
void add (
    Drawable & d ) [virtual]
```

##### Parameters

<i>d</i>	The drawable to add.
----------	----------------------

Reimplemented from [Container](#).

#### 7.164.2.2 animateToState()

```
void animateToState (
    SlideMenu::State newState ) [virtual]
```

##### Parameters

<i>newState</i>	The new state of the <a href="#">SlideMenu</a> .
-----------------	--------------------------------------------------

### 7.164.2.3 animationEndedHandler()

```
void animationEndedHandler (
    const MoveAnimator< Container > & container ) [protected]
```

#### Parameters

<i>container</i>	The menuContainer.
------------------	--------------------

### 7.164.2.4 getAnimationDuration()

```
uint16_t getAnimationDuration ( ) const [virtual]
```

#### Returns

The animation duration.

### 7.164.2.5 getAnimationEasingEquation()

```
EasingEquation getAnimationEasingEquation ( ) const [virtual]
```

#### Returns

The animation easing equation.

### 7.164.2.6 getBackgroundX()

```
int16_t getBackgroundX ( ) const [virtual]
```

#### Returns

The background x coordinate.

### 7.164.2.7 getBackgroundY()

```
int16_t getBackgroundY ( ) const [virtual]
```

#### Returns

The background y coordinate.

### 7.164.2.8 getCollapsedXCoordinate()

```
int16_t getCollapsedXCoordinate ( ) [protected], [virtual]
```

**Returns**

The collapsed x coordinate.

**7.164.2.9 getCollapsedYCoordinate()**

```
int16_t getCollapsedYCoordinate ( ) [protected], [virtual]
```

**Returns**

The collapsed y coordinate.

**7.164.2.10 getExpandDirection()**

```
SlideMenu::ExpandDirection getExpandDirection ( ) const [virtual]
```

**Returns**

The expand direction.

**7.164.2.11 getExpandedStateTimeout()**

```
uint16_t getExpandedStateTimeout ( ) const [virtual]
```

**Returns**

The expanded state timeout.

**7.164.2.12 getExpandedStateTimer()**

```
uint16_t getExpandedStateTimer ( ) const [virtual]
```

**Returns**

The expanded state timer.

**7.164.2.13 getExpandedXCoordinate()**

```
int16_t getExpandedXCoordinate ( ) [protected], [virtual]
```

**Returns**

The expanded x coordinate.



**7.164.2.14 getExpandedYCoordinate()**

```
int16_t getExpandedYCoordinate ( ) [protected], [virtual]
```

**Returns**

The expanded y coordinate.

**7.164.2.15 getHiddenPixelsWhenExpanded()**

```
int16_t getHiddenPixelsWhenExpanded ( ) const [virtual]
```

**Returns**

The hidden pixels when expanded.

**7.164.2.16 getState()**

```
SlideMenu::State getState ( ) [virtual]
```

**Returns**

The current state.

**7.164.2.17 getStateChangeButtonX()**

```
int16_t getStateChangeButtonX ( ) const [virtual]
```

**Returns**

The state change button x coordinate.

**7.164.2.18 getStateChangeButtonY()**

```
int16_t getStateChangeButtonY ( ) const [virtual]
```

**Returns**

The state change button y coordinate.

**7.164.2.19 getVisiblePixelsWhenCollapsed()**

```
int16_t getVisiblePixelsWhenCollapsed ( ) const [virtual]
```

**Returns**

The visible pixels when collapsed.

**7.164.2.20 remove()**

```
void remove (
    Drawable & d ) [virtual]
```

**Parameters**

<i>d</i>	The drawable to remove.
----------	-------------------------

Reimplemented from [Container](#).

**7.164.2.21 resetExpandedStateTimer()**

```
void resetExpandedStateTimer ( ) [virtual]
```

Resets the expanded state timer. The [SlideMenu](#) will animate to the COLLAPSED state after a number of ticks (set with `setExpandedStateTimeout(..)`). This method resets this timer.

**7.164.2.22 setAnimationDuration()**

```
void setAnimationDuration (
    uint16_t duration ) [virtual]
```

**Parameters**

<i>duration</i>	The animation duration.
-----------------	-------------------------

**7.164.2.23 setAnimationEasingEquation()**

```
void setAnimationEasingEquation (
    EasingEquation animationEasingEquation ) [virtual]
```

**Parameters**

<i>animationEasingEquation</i>	The animation easing equation.
--------------------------------	--------------------------------

**7.164.2.24 setExpandDirection()**

```
void setExpandDirection (
    SlideMenu::ExpandDirection newExpandDirection ) [virtual]
```

**Parameters**

<i>newExpandDirection</i>	The new expand direction.
---------------------------	---------------------------

**7.164.2.25 setExpandedStateTimeout()**

```
void setExpandedStateTimeout (
    uint16_t timeout ) [virtual]
```

Sets the expanded state timeout in ticks. The [SlideMenu](#) will animate to the COLLAPSED state when this number of ticks has been executed while the [SlideMenu](#) is in the EXPANDED state. The timer can be reset with the [resetExpandedStateTimer](#) method.

**Parameters**

<i>timeout</i>	The timeout in ticks.
----------------	-----------------------

**7.164.2.26 setHiddenPixelsWhenExpanded()**

```
void setHiddenPixelsWhenExpanded (
    int16_t hiddenPixels ) [virtual]
```

**Parameters**

<i>hiddenPixels</i>	The hidden pixels.
---------------------	--------------------

**7.164.2.27 setState()**

```
void setState (
    SlideMenu::State newState ) [virtual]
```

**Parameters**

<i>newState</i>	The new state of the <a href="#">SlideMenu</a> .
-----------------	--------------------------------------------------

**7.164.2.28 setStateChangedAnimationEndedCallback()**

```
void setStateChangedAnimationEndedCallback (
    GenericCallback< const SlideMenu & > & callback ) [virtual]
```

**Parameters**

<i>callback</i>	The callback.
-----------------	---------------

**7.164.2.29 setStateChangedCallback()**

```
void setStateChangedCallback (
    GenericCallback< const SlideMenu & > & callback ) [virtual]
```

## Parameters

<i>callback</i>	The callback.
-----------------	---------------

7.164.2.30 **setup()** [1/2]

```
void setup (
    SlideMenu::ExpandDirection newExpandDirection,
    const Bitmap & backgroundBMP,
    const Bitmap & stateChangeButtonBMP,
    const Bitmap & stateChangeButtonPressedBMP ) [virtual]
```

Setup the [SlideMenu](#) by positioning the stateChangeButton next to background image relative to the expand direction and center it in the other dimension. The width and height of the [SlideMenu](#) will be automatically set to span both elements. Default values are: expandedStateTimeout = 200, visiblePixelsWhenCollapsed = 0, hiddenPixelsWhenExpanded = 0, animationDuration = 10, animationEquation = cubicEaseInOut.

## Parameters

<i>newExpandDirection</i>	The new expand direction.
<i>backgroundBMP</i>	The background bitmap.
<i>stateChangeButtonBMP</i>	The state change button bitmap.
<i>stateChangeButtonPressedBMP</i>	The state change button pressed bitmap.

7.164.2.31 **setup()** [2/2]

```
void setup (
    SlideMenu::ExpandDirection newExpandDirection,
    const Bitmap & backgroundBMP,
    const Bitmap & stateChangeButtonBMP,
    const Bitmap & stateChangeButtonPressedBMP,
    int16_t backgroundX,
    int16_t backgroundY,
    int16_t stateChangeButtonX,
    int16_t stateChangeButtonY ) [virtual]
```

Setup method for the [SlideMenu](#). Positioning of the background image and the stateChangeButton is done by stating the X and Y coordinates for the elements (relative to the [SlideMenu](#)). The width and height of the [SlideMenu](#) will be automatically set to span both elements. Default values are: expandedStateTimeout = 200, visiblePixelsWhenCollapsed = 0, hiddenPixelsWhenExpanded = 0, animationDuration = 10, animationEquation = cubicEaseInOut.

## Parameters

<i>newExpandDirection</i>	The new expand direction.
<i>backgroundBMP</i>	The background bitmap.
<i>stateChangeButtonBMP</i>	The state change button bitmap.
<i>stateChangeButtonPressedBMP</i>	The state change button pressed bitmap.
<i>backgroundX</i>	The background x coordinate.
<i>backgroundY</i>	The background y coordinate.
<i>stateChangeButtonX</i>	The state change button x coordinate.
<i>stateChangeButtonY</i>	The state change button y coordinate.

**7.164.2.32 setVisiblePixelsWhenCollapsed()**

```
void setVisiblePixelsWhenCollapsed (
    int16_t visiblePixels ) [virtual]
```

**Parameters**

<i>visiblePixels</i>	The visible pixels.
----------------------	---------------------

**7.164.2.33 stateChangeButtonClickedHandler()**

```
void stateChangeButtonClickedHandler (
    const AbstractButton & button ) [protected]
```

**Parameters**

<i>button</i>	The state change button.
---------------	--------------------------

**7.165 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.*
- virtual void [setupHorizontalSlider](#) (uint16\_t backgroundX, uint16\_t backgroundY, uint16\_t indicatorY, uint16\_t indicatorMinX, uint16\_t indicatorMaxX)  
*Sets up the slider in horizontal mode.*

- virtual void [setupVerticalSlider](#) (uint16\_t backgroundX, uint16\_t backgroundY, uint16\_t indicatorX, uint16\_t indicatorMinY, uint16\_t indicatorMaxY)  
*Sets up the slider in vertical mode.*
- virtual uint16\_t [getIndicatorMin](#) () const  
*Gets indicator minimum.*
- virtual uint16\_t [getIndicatorMax](#) () const  
*Gets indicator maximum.*
- virtual void [setValueRange](#) (int minValue, int maxValue, int newValue)  
*Sets the value range of the slider.*
- virtual uint16\_t [getMinValue](#) () const  
*Gets the minimum value.*
- virtual uint16\_t [getMaxValue](#) () const  
*Gets the maximum value.*
- virtual void [setValueRange](#) (int minValue, int maxValue)  
*Sets the value range of the slider.*
- virtual 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.*
- virtual void [updateIndicatorPosition](#) (int16\_t position)  
*Updates the indicator position described by position.*
- virtual int16\_t [valueToPosition](#) (int value) const  
*Translate a value in the value range to a corresponding indicator position.*
- virtual int [positionToValue](#) (int16\_t position) const  
*Translate a position in the indicator position range to the corresponding value.*
- virtual uint16\_t [getIndicatorRadius](#) () const  
*Gets the indicator radius.*
- virtual int [getIndicatorPositionRangeSize](#) () const  
*Gets the indicator position range size.*
- virtual int [getValueRangeSize](#) () const  
*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

### 7.165.1 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.

### 7.165.2 Constructor & Destructor Documentation

#### 7.165.2.1 Slider()

```
Slider ( )
```

Default constructor. Set the value range to default 0-100.

### 7.165.3 Member Function Documentation

#### 7.165.3.1 `getIndicatorMax()`

```
uint16_t getIndicatorMax ( ) const [inline], [virtual]
```

Gets indicator maximum previous set using `setupHorizontalSlider` or `setupVerticalSlider`.

##### Returns

The calculated indicator maximum.

##### See also

[setupHorizontalSlider](#)  
[setupVerticalSlider](#)  
[getIndicatorMin](#)

#### 7.165.3.2 `getIndicatorMin()`

```
uint16_t getIndicatorMin ( ) const [inline], [virtual]
```

Gets indicator minimum previous set using `setupHorizontalSlider` or `setupVerticalSlider`.

##### Returns

The indicator minimum.

##### See also

[setupHorizontalSlider](#)  
[setupVerticalSlider](#)  
[getIndicatorMax](#)

#### 7.165.3.3 `getIndicatorPositionRangeSize()`

```
int getIndicatorPositionRangeSize ( ) const [protected], [virtual]
```

##### Returns

The indicator position range size.

#### 7.165.3.4 `getIndicatorRadius()`

```
uint16_t getIndicatorRadius ( ) const [protected], [virtual]
```

##### Returns

The the indicator radius.



#### 7.165.3.5 getMaxValue()

```
uint16_t getMaxValue ( ) const [inline], [virtual]
```

Gets the maximum value previously set using setValueRange.

##### Returns

The maximum value.

##### See also

[setValueRange](#)  
[getMinValue](#)

#### 7.165.3.6 getMinValue()

```
uint16_t getMinValue ( ) const [inline], [virtual]
```

Gets the minimum value previously set using setValueRange.

##### Returns

The minimum value.

##### See also

[setValueRange](#)  
[getMaxValue](#)

#### 7.165.3.7 getType()

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

##### Returns

TYPE\_BUTTON.

Reimplemented from [Container](#).

#### 7.165.3.8 getValue()

```
int getValue ( ) [inline]
```

Gets the current value represented by the indicator.

##### Returns

The current value.

#### 7.165.3.9 `getValueRangeSize()`

```
int getValueRangeSize ( ) const [protected], [virtual]
```

##### Returns

The value range size.

#### 7.165.3.10 `handleClickEvent()`

```
void handleClickEvent (
    const ClickEvent & evt ) [protected], [virtual]
```

##### Parameters

<i>evt</i>	The event.
------------	------------

Reimplemented from [Drawable](#).

#### 7.165.3.11 `handleDragEvent()`

```
void handleDragEvent (
    const DragEvent & evt ) [protected], [virtual]
```

##### Parameters

<i>evt</i>	The event.
------------	------------

Reimplemented from [Drawable](#).

#### 7.165.3.12 `positionToValue()`

```
int positionToValue (
    int16_t position ) const [protected], [virtual]
```

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

<i>position</i>	The position.
-----------------	---------------

##### Returns

The value that corresponds to the coordinate.

7.165.3.13 `setBitmaps()` [1/2]

```
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 `sliderBackgroundSelected`.

## Parameters

<i>sliderBackground</i>	The slider background with the slider range unselected.
<i>sliderBackgroundSelected</i>	The slider background with the slider range selected.
<i>indicator</i>	The indicator.

7.165.3.14 `setBitmaps()` [2/2]

```
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 `sliderBackgroundSelected`.

## Parameters

<i>sliderBackground</i>	The slider background.
<i>sliderBackgroundSelected</i>	The slider background selected.
<i>indicator</i>	The indicator.

7.165.3.15 `setNewValueCallback()`

```
void setNewValueCallback (
    GenericCallback< const Slider &, int > & callback ) [inline]
```

Associates an action to be performed when the slider changes its value.

## Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">Slider</a> and the current value of the slider.
-----------------	----------------------------------------------------------------------------------------------------------------------------------------

See also

[GenericCallback](#)

#### 7.165.3.16 setStartValueCallback()

```
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

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">Slider</a> and the current value of the slider at interaction start.
-----------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------

See also

[GenericCallback](#)

#### 7.165.3.17 setStopValueCallback()

```
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

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">Slider</a> and the current value of the slider at interaction end.
-----------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------

See also

[GenericCallback](#)

#### 7.165.3.18 setupHorizontalSlider()

```
void setupHorizontalSlider (
    uint16_t backgroundX,
    uint16_t backgroundY,
    uint16_t indicatorY,
    uint16_t indicatorMinX,
    uint16_t indicatorMaxX ) [virtual]
```

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

<i>backgroundX</i>	The background x coordinate inside the slider.
<i>backgroundY</i>	The background y coordinate inside the slider.
<i>indicatorY</i>	The indicator y coordinate inside the slider.
<i>indicatorMinX</i>	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 <code>indicatorMaxX</code> .
<i>indicatorMaxX</i>	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 <code>indicatorMinX</code> .

#### 7.165.3.19 `setupVerticalSlider()`

```
void setupVerticalSlider (
    uint16_t backgroundX,
    uint16_t backgroundY,
    uint16_t indicatorX,
    uint16_t indicatorMinY,
    uint16_t indicatorMaxY ) [virtual]
```

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

<i>backgroundX</i>	The background x coordinate inside the slider.
<i>backgroundY</i>	The background y coordinate inside the slider.
<i>indicatorX</i>	The indicator x coordinate inside the slider.
<i>indicatorMinY</i>	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 <code>indicatorMaxX</code> .
<i>indicatorMaxY</i>	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 <code>indicatorMinX</code> .

#### 7.165.3.20 `setValue()`

```
void setValue (
    int value ) [virtual]
```

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

<i>value</i>	The value.
--------------	------------

#### 7.165.3.21 setValueRange() [1/2]

```
void setValueRange (
    int minValue,
    int maxValue,
    int newValue ) [virtual]
```

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

<i>minValue</i>	The minimum value. Must be less than maxValue.
<i>maxValue</i>	The maximum value. Must be greater than minValue.
<i>newValue</i>	The new value.

#### 7.165.3.22 setValueRange() [2/2]

```
void setValueRange (
    int minValue,
    int maxValue ) [virtual]
```

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

<i>minValue</i>	The minimum value. Must be less than maxValue.
<i>maxValue</i>	The maximum value. Must be greater than minValue.

## 7.165.3.23 updateIndicatorPosition()

```
void updateIndicatorPosition (
    int16_t position ) [protected], [virtual]
```

Updates the indicator position described by position. Calls the newSliderValueCallback with the new value.

## Parameters

<i>position</i>	The position (x coordinate in horizontal mode and y coordinate in vertical mode).
-----------------	-----------------------------------------------------------------------------------

## 7.165.3.24 valueToPosition()

```
int16_t valueToPosition (
    int value ) const [protected], [virtual]
```

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

<i>value</i>	The value.
--------------	------------

## Returns

The coordinate that corresponds to the value.

## 7.166 SlideTransition&lt; templateDirection &gt; 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.

### 7.166.1 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

<i>templateDirection</i>	Type of the template direction.
--------------------------	---------------------------------

See also

[Transition](#)

### 7.166.2 Constructor & Destructor Documentation

#### 7.166.2.1 SlideTransition()

```
SlideTransition (
    const uint8_t transitionSteps = 20 ) [inline]
```

Constructor.

#### Parameters

<i>transitionSteps</i>	Number of steps in the transition animation.
------------------------	----------------------------------------------

#### 7.166.2.2 ~SlideTransition()

```
~SlideTransition ( ) [inline], [virtual]
```

Destructor.

### 7.166.3 Member Function Documentation



**7.166.3.1 handleTickEvent()**

```
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](#).

**7.166.3.2 init()**

```
void init ( ) [inline], [virtual]
```

Initializes this object.

See also

[Transition::init\(\)](#)

Reimplemented from [Transition](#).

**7.166.3.3 initMoveDrawable()**

```
void initMoveDrawable (
    Drawable & d ) [inline], [protected], [virtual]
```

Moves the [Drawable](#) to its initial position.

Parameters

in	<i>d</i>	The <a href="#">Drawable</a> to move.
----	----------	---------------------------------------

**7.166.3.4 tearDown()**

```
void tearDown ( ) [inline], [virtual]
```

Tear down.

See also

[Transition::teadDown\(\)](#)

Reimplemented from [Transition](#).

**7.166.3.5 tickMoveDrawable()**

```
void tickMoveDrawable (
    Drawable & d ) [inline], [protected], [virtual]
```

Moves the [Drawable](#).

## Parameters

in	<i>d</i>	The <a href="#">Drawable</a> to move.
----	----------	---------------------------------------

## 7.167 [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.*

### 7.167.1 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

<i>T</i>	specifies the type to enable the Snap behavior to.
----------	----------------------------------------------------

#### See also

[Draggable<T>](#)

### 7.167.2 Constructor & Destructor Documentation

#### 7.167.2.1 [Snapper\(\)](#)

```
Snapper ( ) [inline]
```

Default constructor.

#### 7.167.2.2 ~Snapper()

```
~Snapper ( ) [inline], [virtual]
```

Destructor.

### 7.167.3 Member Function Documentation

#### 7.167.3.1 handleClickEvent()

```
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

<i>evt</i>	The click event.
------------	------------------

#### 7.167.3.2 handleDragEvent()

```
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

<i>evt</i>	The drag event.
------------	-----------------

Reimplemented from [Draggable< T >](#).

#### 7.167.3.3 setDragAction()

```
void setDragAction (
    GenericCallback< const DragEvent & > & callback ) [inline]
```

Associates an action to be performed when the [Snapper](#) is dragged.

##### Parameters

<i>callback</i>	The callback will be executed with the <a href="#">DragEvent</a> .
-----------------	--------------------------------------------------------------------

See also

[GenericCallback](#)

#### 7.167.3.4 setSnappedAction()

```
void setSnappedAction (
    GenericCallback<> & callback ) [inline]
```

Associates an action to be performed when the [Snapper](#) is snapped.

Parameters

in	<i>callback</i>	The callback to be executed on snap.
----	-----------------	--------------------------------------

See also

[GenericCallback](#)

#### 7.167.3.5 setSnapPosition()

```
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

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

## 7.168 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.*
- virtual void [makeSnapshot](#) (const [BitmapId](#) bmp)  
*Makes a snapshot of the area the [SnapshotWidget](#) currently to a bitmap.*
- void [setAlpha](#) (const uint8\_t a)  
*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

### 7.168.1 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. Note: The snapshot must be taken from a byte aligned position. On BPP=4, this means on even positions, x=0, 2, 4, 8,... On BPP=2, this means on positions, x= 0, 4, 8, 12,... On BPP=1, this means on positions, x= 0, 8, 16,...

See also

[Widget](#)

### 7.168.2 Constructor & Destructor Documentation

#### 7.168.2.1 SnapshotWidget()

[SnapshotWidget](#) ( )

Default constructor.

#### 7.168.2.2 ~SnapshotWidget()

[~SnapshotWidget](#) ( ) [virtual]

Destructor.

### 7.168.3 Member Function Documentation

**7.168.3.1 draw()**

```
void draw (
    const Rect & invalidatedArea ) const [virtual]
```

Draws the [SnapshotWidget](#). It supports partial drawing, so it only redraws the area described by `invalidatedArea`.

**Parameters**

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

Implements [Drawable](#).

**7.168.3.2 getAlpha()**

```
uint8_t getAlpha ( ) const [inline]
```

Gets the current alpha value.

**Returns**

The alpha value.

**7.168.3.3 getSolidRect()**

```
Rect getSolidRect ( ) const [virtual]
```

Gets solid rectangle.

**Returns**

The solid rectangle.

Implements [Drawable](#).

**7.168.3.4 getType()**

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

**Returns**

TYPE\_SNAPSHOTWIDGET.

Reimplemented from [Widget](#).

**7.168.3.5 makeSnapshot()** [1/2]

```
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](#). The snapshot is stored in Animation Storage.

## 7.168.3.6 makeSnapshot() [2/2]

```
void makeSnapshot (
    const BitmapId bmp ) [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](#). The snapshot is stored in the provided dynamic bitmap.

## Parameters

<i>bmp</i>	The target dynamic bitmap.
------------	----------------------------

## 7.168.3.7 setAlpha()

```
void setAlpha (
    const uint8_t a ) [inline]
```

Sets the alpha value.

## Parameters

<i>a</i>	The alpha value.
----------	------------------

## 7.169 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, uint8\_t alpha, [Alignment](#) alignment, uint16\_t linespace, [TextRotation](#) rotation, [TextDirection](#) textDirection, uint8\_t indentation, [WideTextAction](#) wideTextAction=WIDE\_TEXT\_NONE)  
*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).*

- `uint16_t` [linespace](#)  
*Line space in pixels for multiline strings. Default is 0.*
- `uint8_t` [alpha](#)  
*8-bit alpha value. Default is 255 (solid).*
- `uint8_t` [indentation](#)  
*Indentation of text inside rectangle. Text will start this far from the left/right edge.*
- [WideTextAction](#) [wideTextAction](#)  
*What to do with wide text lines.*

### 7.169.1 Detailed Description

The visual elements when writing a string.

### 7.169.2 Constructor & Destructor Documentation

#### 7.169.2.1 StringVisuals() [1/2]

```
StringVisuals ( ) [inline]
```

Construct an empty [StringVisuals](#) object.

#### 7.169.2.2 StringVisuals() [2/2]

```
StringVisuals (
    const Font * font,
    colortype color,
    uint8_t alpha,
    Alignment alignment,
    uint16_t linespace,
    TextRotation rotation,
    TextDirection textDirection,
    uint8_t indentation,
    WideTextAction wideTextAction = WIDE_TEXT_NONE ) [inline]
```

#### Parameters

<i>font</i>	The <a href="#">Font</a> with which to draw the text.
<i>color</i>	The color with which to draw the text.
<i>alpha</i>	Alpha blending. Default value is 255 (solid)
<i>alignment</i>	How to align the text.
<i>linespace</i>	<a href="#">Line</a> space in pixels between each line, in case the text contains newline characters.
<i>rotation</i>	How to rotate the text.
<i>textDirection</i>	The text direction.
<i>indentation</i>	The indentation of the text from the left and right of the text area rectangle.
<i>wideTextAction</i>	What to do with lines longer than the width of the <a href="#">TextArea</a> .



## 7.170 SwipeContainer Class Reference

A swipe container.

```
#include <touchgfx/containers/SwipeContainer.hpp>
```

### Public Member Functions

- virtual void [handleTickEvent](#) ()  
*Called periodically by the framework if the [Drawable](#) instance has subscribed to timer ticks.*
- virtual void [handleClickEvent](#) (const [touchgfx::ClickEvent](#) &evt)  
*Defines the event handler interface for ClickEvents.*
- virtual void [handleDragEvent](#) (const [touchgfx::DragEvent](#) &evt)  
*Defines the event handler interface for DragEvents.*
- virtual void [handleGestureEvent](#) (const [touchgfx::GestureEvent](#) &evt)  
*Defines the event handler interface for GestureEvents.*
- virtual void [add](#) ([touchgfx::Drawable](#) &page)  
*Adds a page to the container.*
- virtual void [remove](#) ([Drawable](#) &page)  
*Removes the page from the container.*
- virtual void [setSwipeCutoff](#) (uint16\_t cutoff)  
*Set the swipe cutoff.*
- void [setPageIndicatorXY](#) (int16\_t x, int16\_t y)  
*Sets the x and y position of the page indicator.*
- void [setPageIndicatorXYWithCenteredX](#) (int16\_t x, int16\_t y)  
*Sets the x and y position of the page indicator.*
- void [setPageIndicatorBitmaps](#) (const [touchgfx::Bitmap](#) &normalPage, const [touchgfx::Bitmap](#) &highlightedPage)  
*Sets the bitmaps that are used by the page indicator.*
- void [setEndSwipeElasticWidth](#) (uint16\_t width)  
*When dragging either one of the end pages a part of the background will become visible until the user stop dragging and the end page swipes back to its position. The width of this area is set by this method.*
- uint8\_t [getNumberOfPages](#) ()  
*Gets number of pages.*
- void [setSelectedPage](#) (uint8\_t pageIndex)  
*Sets the selected page.*

### Additional Inherited Members

#### 7.170.1 Detailed Description

See also

[touchgfx::Container](#).

#### 7.170.2 Member Function Documentation

### 7.170.2.1 add()

```
void add (
    touchgfx::Drawable & page ) [virtual]
```

Adds a page to the container.

#### Parameters

<i>in, out</i>	<i>page</i>	The page to add.
----------------	-------------	------------------

#### Note

All pages must have the same width and height.

Reimplemented from [Container](#).

### 7.170.2.2 getNumberOfPages()

```
uint8_t getNumberOfPages ( ) [inline]
```

Gets number of pages.

#### Returns

The number of pages.

### 7.170.2.3 handleClickEvent()

```
virtual void handleClickEvent (
    const touchgfx::ClickEvent & evt ) [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

<i>evt</i>	The <a href="#">ClickEvent</a> received from the <a href="#">HAL</a> .
------------	------------------------------------------------------------------------

Reimplemented from [Drawable](#).

### 7.170.2.4 handleDragEvent()

```
virtual void handleDragEvent (
    const touchgfx::DragEvent & evt ) [virtual]
```

Defines the event handler interface for DragEvents. The event is only received if the drawable is touchable.

#### Parameters

<i>evt</i>	The <a href="#">DragEvent</a> received from the <a href="#">HAL</a> .
------------	-----------------------------------------------------------------------

Reimplemented from [Drawable](#).

#### 7.170.2.5 handleGestureEvent()

```
virtual void handleGestureEvent (
    const touchgfx::GestureEvent & evt ) [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

<i>evt</i>	The <a href="#">GestureEvent</a> received from the <a href="#">HAL</a> .
------------	--------------------------------------------------------------------------

Reimplemented from [Drawable](#).

#### 7.170.2.6 handleTickEvent()

```
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](#).

#### 7.170.2.7 remove()

```
void remove (
    Drawable & page ) [virtual]
```

Removes the page from the container.

##### Parameters

<i>in, out</i>	<i>page</i>	The page to remove.
----------------	-------------	---------------------

##### Note

This is safe to call even if page is not a page (in which case nothing happens).

Reimplemented from [Container](#).

#### 7.170.2.8 setEndSwipeElasticWidth()

```
void setEndSwipeElasticWidth (
    uint16_t width )
```

**Parameters**

<i>width</i>	The width in pixels.
--------------	----------------------

**7.170.2.9 setPageIndicatorBitmaps()**

```
void setPageIndicatorBitmaps (
    const touchgfx::Bitmap & normalPage,
    const touchgfx::Bitmap & highlightedPage )
```

Sets the bitmaps that are used by the page indicator.

**Parameters**

<i>normalPage</i>	The normal page.
<i>highlightedPage</i>	The highlighted page.

**7.170.2.10 setPageIndicatorXY()**

```
void setPageIndicatorXY (
    int16_t x,
    int16_t y )
```

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

**7.170.2.11 setPageIndicatorXYWithCenteredX()**

```
void setPageIndicatorXYWithCenteredX (
    int16_t x,
    int16_t y )
```

Sets the x and y position of the page indicator. The value specified as x will be the center coordinate of the page indicators.

**Parameters**

<i>x</i>	The center x coordinate.
<i>y</i>	The y coordinate.

**Note**

This method should not be used until all pages have been added, the `setPageIndicatorBitmaps` has been called and the page indicator therefore has the correct width.

## 7.170.2.12 setSelectedPage()

```
void setSelectedPage (
    uint8_t pageIndex )
```

Sets the selected page.

## Parameters

<i>pageIndex</i>	Zero-based index of the page. Range from 0 to numberOfPages-1.
------------------	----------------------------------------------------------------

## 7.170.2.13 setSwipeCutoff()

```
void setSwipeCutoff (
    uint16_t cutoff ) [virtual]
```

Set the swipe cutoff which indicates how far you should drag a page before it results in a page change.

## Parameters

<i>cutoff</i>	The cutoff in pixels.
---------------	-----------------------

## 7.171 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 line spacing of the [TextArea](#).*
- uint16\_t [getLinespacing](#) () const

- Gets the line spacing 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](#).*
  - void [resizeHeightToCurrentText](#) ()
 

*Sets the hight of the [TextArea](#).*
  - void [setWidthTextAction](#) ([WideTextAction](#) action)
 

*Sets wide text action.*
  - [WideTextAction](#) [getWideTextAction](#) () const
 

*Gets wide text action.*
  - virtual uint16\_t [getType](#) () const
 

*For GUI testing only.*

## Protected Attributes

- [TypedText](#) [typedText](#)

*The [TypedText](#) to display.*
- [color\\_t](#) [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.*
- [TextRotation](#) [rotation](#)

*The text rotation to use.*
- [WideTextAction](#) [wideTextAction](#)

*What to do if the text is wider than the text area.*

## Additional Inherited Members

### 7.171.1 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.

### 7.171.2 Constructor & Destructor Documentation

#### 7.171.2.1 TextArea()

```
TextArea ( ) [inline]
```

Create an empty [TextArea](#). Default color is black.

### 7.171.3 Member Function Documentation

#### 7.171.3.1 draw()

```
void draw (
    const Rect & area ) const [virtual]
```

Draws the text. Called automatically.

#### Parameters

<code>area</code>	The invalidated area.
-------------------	-----------------------

Implements [Drawable](#).

Reimplemented in [TextAreaWithTwoWildcards](#), and [TextAreaWithOneWildcard](#).

#### 7.171.3.2 getAlpha()

```
uint8_t getAlpha ( ) const [inline]
```

Gets the alpha value of the text.

**Returns**

The alpha value. 255 = completely solid. 0 = invisible.

**7.171.3.3 getColor()**

```
colortype getColor ( ) const [inline]
```

Gets the color of the text.

**Returns**

The color to used for drawing the text.

**7.171.3.4 getIndentation()**

```
uint8_t getIndentation ( ) [inline]
```

Gets the indentation.

**Returns**

The indentation.

**See also**

[setIndetation](#)

**7.171.3.5 getLinespacing()**

```
uint16_t getLinespacing ( ) const [inline]
```

Gets the line spacing of the [TextArea](#).

**Returns**

The line spacing.

**7.171.3.6 getRotation()**

```
TextRotation getRotation ( ) const [inline]
```

Gets rotation of the text in the [TextArea](#).

**Returns**

The rotation of the text.



### 7.171.3.7 getSolidRect()

```
Rect getSolidRect ( ) const [inline], [virtual]
```

Gets solid rectangle.

#### Returns

the largest solid rectangle for this widget. For a [TextArea](#), this is an empty area.

Implements [Drawable](#).

### 7.171.3.8 getTextHeight()

```
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](#).

### 7.171.3.9 getTextWidth()

```
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](#).

### 7.171.3.10 getType()

```
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](#).

#### 7.171.3.11 `getTypedText()`

```
TypedText getTypedText ( ) const [inline]
```

Gets the [TypedText](#) of the text area.

##### Returns

The currently used [TypedText](#).

#### 7.171.3.12 `getWideTextAction()`

```
WideTextAction getWideTextAction ( ) const [inline]
```

Gets wide text action preciously set using `setWideTextAction`.

##### Returns

current `WideTextAction` setting.

##### See also

[setWideTextAction](#)

#### 7.171.3.13 `resizeHeightToCurrentText()`

```
void resizeHeightToCurrentText ( )
```

Sets the height of the [TextArea](#) to match the height of the current associated text for the current selected language. This is especially useful for texts with WordWrap enabled.

Please note that if the current text rotation is either 90 or 270 degrees, the width of the text area will be set and not the height, as the text is rotated.

##### See also

[resizeToCurrentText](#)  
[setWordWrap](#)  
[setRotation](#)

#### 7.171.3.14 `resizeToCurrentText()`

```
void resizeToCurrentText ( )
```

Sets the dimensions of the [TextArea](#) to match the width and height of the current associated text for the current selected language.

Please note that if the current text rotation is either 90 or 270 degrees, the width of the text area will be set to the height of the text and vice versa, as the text is rotated.

##### See also

[setRotation](#)  
[resizeHeightToCurrentText](#)

### 7.171.3.15 setAlpha()

```
void setAlpha (
    uint8_t alpha ) [inline]
```

Sets the alpha value of the text.

#### Parameters

<i>alpha</i>	The alpha value. 255 = completely solid. 0 = invisible.
--------------	---------------------------------------------------------

### 7.171.3.16 setBaselineY()

```
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

<i>baselineY</i>	The y coordinate of the baseline.
------------------	-----------------------------------

### 7.171.3.17 setColor()

```
void setColor (
    colortype color ) [inline]
```

Sets the color of the text.

#### Parameters

<i>color</i>	The color to use.
--------------	-------------------

### 7.171.3.18 setIndentation()

```
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

<i>indent</i>	The indentation from left (when left aligned text) and right (when right aligned text).
---------------	-----------------------------------------------------------------------------------------

## See also

`getMaxPixelsLeft`

7.171.3.19 `setLinespacing()`

```
void setLinespacing (
    uint16_t space ) [inline]
```

Sets the line spacing of the [TextArea](#).

## Parameters

<i>space</i>	The line spacing of use in the <a href="#">TextArea</a> .
--------------	-----------------------------------------------------------

7.171.3.20 `setRotation()`

```
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 `TEXT_ROTATE_180` and `TEXT_ROTATE_270` is further rotate 90 degrees clockwise.

## Parameters

<i>rotation</i>	The rotation of the text.
-----------------	---------------------------

7.171.3.21 `setTypedText()`

```
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

<i>t</i>	The <a href="#">TypedText</a> for this widget to display.
----------	-----------------------------------------------------------

## 7.171.3.22 setWideTextAction()

```
void setWideTextAction (
    WideTextAction action ) [inline]
```

Sets wide text action. Defines what to do if a line of text is wider than the text area. Default action is WIDE\_TEXT↔\_NONE which means that text lines are only broken if there is a newline in the text.

If wrapping is enabled and the text would occupy more lines than the size of the [TextArea](#), the last line will get an ellipsis to signal that some text is missing. The character used for ellipsis is taken from the text spreadsheet.

## Parameters

<i>action</i>	The action to perform for wide lines of text.
---------------	-----------------------------------------------

## See also

[WideTextAction](#)  
[getWideTextAction](#)  
[resizeHeightToCurrentText](#)

## 7.171.3.23 setXBaselineY()

```
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

<i>x</i>	The x coordinate of the <a href="#">TextArea</a> .
<i>baselineY</i>	The y coordinate of the baseline.

## 7.172 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

### 7.172.1 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](#)

### 7.172.2 Constructor & Destructor Documentation

#### 7.172.2.1 TextAreaWithOneWildcard()

```
TextAreaWithOneWildcard ( ) [inline]
```

Create an empty text area.

Note

No text can be displayed until a font is set. Default color is black.

### 7.172.3 Member Function Documentation

#### 7.172.3.1 draw()

```
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

<i>area</i>	The invalidated area.
-------------	-----------------------

Reimplemented from [TextArea](#).

#### 7.172.3.2 getTextHeight()

```
int16_t getTextHeight ( ) [inline], [virtual]
```

Gets text height.

##### Returns

The text height.

Reimplemented from [TextArea](#).

#### 7.172.3.3 getTextWidth()

```
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](#).

#### 7.172.3.4 getType()

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

##### Returns

TYPE\_TEXTAREAWITHONEWILDCARD.

Reimplemented from [TextArea](#).

#### 7.172.3.5 getWildcard()

```
const Unicode::UnicodeChar * getWildcard ( ) const [inline]
```

Gets the wildcard in the text.

##### Returns

The wildcard used in the text.

### 7.172.3.6 setWildcard()

```
void setWildcard (
    const Unicode::UnicodeChar * value ) [inline]
```

Sets the wildcard in the text. Must be a zero-terminated [UnicodeChar](#) array.

#### Parameters

<i>value</i>	A pointer to the <a href="#">UnicodeChar</a> to set the wildcard to.
--------------	----------------------------------------------------------------------

## 7.173 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

#### 7.173.1 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](#)

## 7.173.2 Constructor & Destructor Documentation

### 7.173.2.1 TextAreaWithTwoWildcards()

```
TextAreaWithTwoWildcards ( ) [inline]
```

Create an empty text area.

#### Note

No text can be displayed until a font is set. Default color is black.

## 7.173.3 Member Function Documentation

### 7.173.3.1 draw()

```
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

<i>area</i>	The invalidated area.
-------------	-----------------------

Reimplemented from [TextArea](#).

### 7.173.3.2 getTextHeight()

```
int16_t getTextHeight ( ) [inline], [virtual]
```

Gets text height.

#### Returns

The text height.

Reimplemented from [TextArea](#).

### 7.173.3.3 getTextWidth()

```
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](#).

**7.173.3.4 getType()**

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

**Returns**

TYPE\_TEXTAREAWITHTWOWILDCARDS.

Reimplemented from [TextArea](#).

**7.173.3.5 getWildcard1()**

```
const Unicode::UnicodeChar * getWildcard1 ( ) const [inline]
```

Gets the first wildcard in the text.

**Returns**

The first wildcard from a [TextArea](#) with two wildcards.

**7.173.3.6 getWildcard2()**

```
const Unicode::UnicodeChar * getWildcard2 ( ) const [inline]
```

Gets the second wildcard in the text.

**Returns**

The second wildcard from a [TextArea](#) with two wildcards.

**7.173.3.7 setWildcard1()**

```
void setWildcard1 (
    const Unicode::UnicodeChar * value ) [inline]
```

Sets the first wildcard in the text. Must be a zero-terminated UnicodeChar array.

**Parameters**

<i>value</i>	A pointer to the UnicodeChar to set the wildcard to.
--------------	------------------------------------------------------

## 7.173.3.8 setWildcard2()

```
void setWildcard2 (
    const Unicode::UnicodeChar * value ) [inline]
```

Sets the second wildcard in the text. Must be a zero-terminated UnicodeChar array.

## Parameters

<i>value</i>	A pointer to the UnicodeChar to set the wildcard to.
--------------	------------------------------------------------------

## 7.174 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,...) const  
*Gets the total height needed by the text.*

## Additional Inherited Members

## 7.174.1 Detailed Description

Base class for TextAreas displaying texts with one or more wildcards.

## See also

[TextAreaWithOneWildcard](#)  
[TextAreaWithTwoWildcards](#)

## 7.174.2 Constructor &amp; Destructor Documentation

## 7.174.2.1 TextAreaWithWildcardBase()

```
TextAreaWithWildcardBase ( ) [inline]
```

Create an empty text area.

## Note

No text can be displayed until a font is set. Default color is black.

## 7.174.3 Member Function Documentation

### 7.174.3.1 calculateTextHeight()

```
int16_t calculateTextHeight (
    const Unicode::UnicodeChar * format,
    ... ) const
```

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

<i>format</i>	The text containing s wildcards.
...	Variable arguments providing additional information.

#### Returns

the total height needed by the text.

## 7.175 TextButtonStyle< T > Class Template Reference

A text button style.

```
#include <touchgfx/containers/buttons/TextButtonStyle.hpp>
```

### Public Member Functions

- [TextButtonStyle](#) ()  
*Default constructor.*
- virtual [~TextButtonStyle](#) ()  
*Destructor.*
- void [setText](#) ([TypedText](#) t)  
*Sets a text.*
- void [setTextX](#) (int16\_t x)  
*Sets text x coordinate.*
- void [setTextY](#) (int16\_t y)  
*Sets text y coordinate.*
- void [setTextXY](#) (int16\_t x, int16\_t y)  
*Sets text xy.*
- void [setTextPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets text position.*
- void [setTextRotation](#) ([TextRotation](#) rotation)  
*Sets text rotation.*
- void [setTextColors](#) ([colortype](#) newColorReleased, [colortype](#) newColorPressed)  
*Sets text colors.*

### Protected Member Functions

- virtual void [handlePressedUpdated](#) ()  
*Handles the pressed updated.*
- virtual void [handleAlphaUpdated](#) ()  
*Handles the alpha updated.*

## Protected Attributes

- [TextArea](#) `text`  
*The text.*
- [colortype](#) `colorReleased`  
*The color released.*
- [colortype](#) `colorPressed`  
*The color pressed.*

### 7.175.1 Detailed Description

```
template<class T>
class touchgfx::TextButtonStyle< T >
```

An text button style. This class is supposed to be used with one of the `ButtonTrigger` classes to create a functional button. This class will show a text in one of two colors depending on the state of the button (pressed or released).

The [TextButtonStyle](#) does not set the size of the enclosing container (normally [AbstractButtonContainer](#)). The size must be set manually.

To get a background behind the text, use [TextButtonStyle](#) together with e.g. [ImageButtonStyle](#): `TextButtonStyle<ImageButtonStyle<ClickButtonTrigger> > myButton;`

The position of the text can be adjusted with `setTextXY` (default is centered).

#### Template Parameters

<code>T</code>	Generic type parameter. Typically a <a href="#">AbstractButtonContainer</a> subclass.
----------------	---------------------------------------------------------------------------------------

See also

[AbstractButtonContainer](#)

### 7.175.2 Member Function Documentation

#### 7.175.2.1 `setText()`

```
void setText (
    TypedText t ) [inline]
```

#### Parameters

<code>t</code>	A <a href="#">TypedText</a> to process.
----------------	-----------------------------------------

#### 7.175.2.2 `setTextColors()`

```
void setTextColors (
    colortype newColorReleased,
    colortype newColorPressed ) [inline]
```

## Parameters

<i>newColorReleased</i>	The new color released.
<i>newColorPressed</i>	The new color pressed.

## 7.175.2.3 setTextPosition()

```
void setTextPosition (
    int16_t x,
    int16_t y,
    int16_t width,
    int16_t height ) [inline]
```

## Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>width</i>	The width of the text.
<i>height</i>	The height of the text.

## 7.175.2.4 setTextRotation()

```
void setTextRotation (
    TextRotation rotation ) [inline]
```

## Parameters

<i>rotation</i>	The rotation.
-----------------	---------------

## 7.175.2.5 setTextX()

```
void setTextX (
    int16_t x ) [inline]
```

## Parameters

<i>x</i>	The x coordinate.
----------	-------------------

## 7.175.2.6 setTextXY()

```
void setTextXY (
    int16_t x,
    int16_t y ) [inline]
```

## Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

## 7.175.2.7 setTextY()

```
void setTextY (
    int16_t y ) [inline]
```

## Parameters

<i>y</i>	The y coordinate.
----------	-------------------

## 7.176 TextProgress Class Reference

A text progress.

```
#include <touchgfx/containers/progress_indicators/TextProgress.hpp>
```

## Public Member Functions

- [TextProgress](#) ()  
*Default constructor.*
- virtual [~TextProgress](#) ()  
*Destructor.*
- virtual void [setProgressIndicatorPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets the position and dimension of the text progress indicator.*
- virtual void [setTypedText](#) (const [TypedText](#) &t)  
*Sets the typed text.*
- virtual [TypedText](#) [getTypedText](#) () const  
*Gets the typed text.*
- virtual void [setColor](#) ([colortype](#) color)  
*Sets the color.*
- virtual [colortype](#) [getColor](#) () const  
*Gets the color.*
- virtual void [setAlpha](#) (uint8\_t alpha)  
*Sets the alpha.*
- virtual uint8\_t [getAlpha](#) () const  
*Gets the alpha.*
- virtual void [setValue](#) (int value)  
*Sets a value.*
- virtual void [setNumberOfDecimals](#) (uint16\_t numberOfDecimals)  
*Sets number of decimals.*
- virtual uint16\_t [getNumberOfDecimals](#) () const  
*Gets number of decimals.*

## Protected Attributes

- [TextAreaWithOneWildcard](#) `textArea`  
*The text area.*
- [Unicode::UnicodeChar](#) `textBuffer` [9]  
*Room for 100.0000.*
- `uint16_t` [decimals](#)  
*The number of decimals.*

## Additional Inherited Members

### 7.176.1 Detailed Description

A text progress will display progress as a number with a given number of decimals.

#### Note

The implementation does not use floating point variables to calculate the progress.

### 7.176.2 Constructor & Destructor Documentation

#### 7.176.2.1 `TextProgress()`

```
TextProgress ( )
```

Default constructor.

#### 7.176.2.2 `~TextProgress()`

```
~TextProgress ( ) [virtual]
```

Destructor.

### 7.176.3 Member Function Documentation

#### 7.176.3.1 `getAlpha()`

```
uint8_t getAlpha ( ) const [virtual]
```

Gets the alpha of the text area.

#### Returns

The alpha.

#### See also

[setAlpha](#)  
[TextArea::getAlpha](#)



### 7.176.3.2 getColor()

```
color_t getColor ( ) const [virtual]
```

Gets the color of the text in the used text area.

#### Returns

The color.

### 7.176.3.3 getNumberOfDecimals()

```
uint16_t getNumberOfDecimals ( ) const [virtual]
```

Gets number of decimals.

#### Returns

The number of decimals.

#### See also

[setNumberOfDecimals](#)

### 7.176.3.4 getTypedText()

```
TypedText getTypedText ( ) const [virtual]
```

Gets the typed text.

#### Returns

The typed text.

#### See also

[setTypedText](#)

### 7.176.3.5 setAlpha()

```
void setAlpha (
    uint8_t alpha ) [virtual]
```

Sets the alpha of the text area.

#### Parameters

<i>alpha</i>	The alpha.
--------------	------------

See also

[getAlpha](#)  
[TextArea::setAlpha](#)

#### 7.176.3.6 setColor()

```
void setColor (
    color_t color ) [virtual]
```

Sets the color of the text in the used text area.

Parameters

<i>color</i>	The color.
--------------	------------

See also

[getColor](#)  
[TextArea::setColor](#)

#### 7.176.3.7 setNumberOfDecimals()

```
void setNumberOfDecimals (
    uint16_t numberOfDecimals ) [virtual]
```

Sets number of decimals when displaying progress.

Parameters

<i>numberOfDecimals</i>	Number of decimals. Only up to two decimals is supported.
-------------------------	-----------------------------------------------------------

See also

[getNumberOfDecimals](#)

#### 7.176.3.8 setProgressIndicatorPosition()

```
void setProgressIndicatorPosition (
    int16_t x,
    int16_t y,
    int16_t width,
    int16_t height ) [virtual]
```

Sets the position and dimension of the text progress indicator relative to the background image.

Parameters

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

## Parameters

<i>width</i>	The width of the text progress indicator.
<i>height</i>	The height of the text progress indicator.

Reimplemented from [AbstractProgressIndicator](#).

## 7.176.3.9 setTypedText()

```
void setTypedText (
    const TypedText & t ) [virtual]
```

Sets the typed text. The text should have one wildcard and could for example "< progress>%".

## Parameters

<i>t</i>	The <a href="#">TypedText</a> to process.
----------	-------------------------------------------

## See also

[getTypedText](#)

## 7.176.3.10 setValue()

```
virtual void setValue (
    int value ) [virtual]
```

Sets the current value in the range (min..max) set by [setRange\(\)](#). Values lower than min are mapped to min, values higher than max are mapped to max.

## Parameters

<i>value</i>	The value.
--------------	------------

Reimplemented from [AbstractProgressIndicator](#).

## 7.177 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](#) ()

*Gets the next character.*

- [Unicode::UnicodeChar getNextLigature](#) ([TextDirection](#) direction)

*Gets the next ligature.*

## 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.*

### 7.177.1 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.

### 7.177.2 Constructor & Destructor Documentation

#### 7.177.2.1 TextProvider()

```
TextProvider ( )
```

Empty constructor. The user must call [initialize\(\)](#) before characters can be provided.

### 7.177.3 Member Function Documentation

#### 7.177.3.1 getNextChar()

```
Unicode::UnicodeChar getNextChar ( )
```

Gets the next character.

#### Returns

The next character of the expanded string or 0 if end of string i reached.

#### See also

[TextProvider::getNextLigature\(\)](#)

#### 7.177.3.2 getNextLigature()

```
Unicode::UnicodeChar getNextLigature (
    TextDirection direction )
```

Gets the next ligature. For most languages this is the same as [getNextChar\(\)](#) but eg. Arabic has different ligatures for each character.

**Note**

Functions [getNextLigature\(\)](#) and [getNextChar\(\)](#) will advance through the same buffer and mixing the use of those functions is not recommended and may cause undesired results. Instead create two TextProviders and user [getNextChar\(\)](#) on one and [getNextLigature\(\)](#) on the other.

**Parameters**

<i>direction</i>	The direction.
------------------	----------------

**Returns**

The next character of the expanded string or 0 if end of string i reached.

**See also**

[TextProvider::getNextChar\(\)](#)

**7.177.3.3 initialize()**

```
void initialize (
    const Unicode::UnicodeChar * format,
    va_list pArg )
```

Initializes the [TextProvider](#). Each '\2' character in the format is replaced by one UnicodeChar\* argument from pArg.

**Parameters**

<i>format</i>	The string to format.
<i>pArg</i>	Format arguments in the form of a va_list.

## 7.178 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.*

### 7.178.1 Detailed Description

Class for setting language and getting texts. The language set will determine which texts will be used.

### 7.178.2 Member Function Documentation

#### 7.178.2.1 `getLanguage()`

```
static LanguageId getLanguage ( ) [inline], [static]
```

Gets the current language.

##### Returns

The id of the language.

#### 7.178.2.2 `getText()`

```
const Unicode::UnicodeChar * getText (
    TypedTextId id ) const [inline]
```

Get text in the set language.

##### Parameters

<i>id</i>	The id of the text to lookup.
-----------	-------------------------------

##### Returns

The text.

#### 7.178.2.3 `setLanguage()`

```
static void setLanguage (
    LanguageId id ) [static]
```

Sets the current language for texts.

##### Parameters

<i>id</i>	The id of the language.
-----------	-------------------------

## 7.179 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.*
- uint8\_t [getAlpha](#) () const  
*Gets the current alpha 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](#) () const  
*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.*

### 7.179.1 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](#)

### 7.179.2 Member Enumeration Documentation

### 7.179.2.1 RenderingAlgorithm

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.

## 7.179.3 Constructor & Destructor Documentation

### 7.179.3.1 ~TextureMapper()

[~TextureMapper](#) ( ) [virtual]

Destructor.

## 7.179.4 Member Function Documentation

### 7.179.4.1 applyTransformation()

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.

### 7.179.4.2 draw()

```
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

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

#### See also

[Drawable::draw\(\)](#)

Implements [Drawable](#).

### 7.179.4.3 drawTriangle()

```
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

	<i>invalidatedArea</i>	The invalidated area.
<i>in, out</i>	<i>fb</i>	If non-null, the fb.
	<i>triangleXs</i>	The triangle xs.
	<i>triangleYs</i>	The triangle ys.
	<i>triangleZs</i>	The triangle zs.
	<i>triangleUs</i>	The triangle us.
	<i>triangleVs</i>	The triangle vs.

#### 7.179.4.4 getAlpha()

```
uint8_t getAlpha ( ) const [inline]
```

Gets the current alpha value.

#### Returns

The current alpha value.

#### See also

[setAlpha](#)

#### 7.179.4.5 getBitmap()

```
Bitmap getBitmap ( ) const [inline]
```

Gets the bitmap for the image.

#### Returns

the bitmap.

#### 7.179.4.6 getBitmapPositionX()

```
float getBitmapPositionX ( ) const [inline], [virtual]
```

Gets bitmap position x coordinate.

#### Returns

The bitmap position x coordinate.

**7.179.4.7 getBitmapPositionY()**

```
float getBitmapPositionY ( ) const [inline], [virtual]
```

Gets bitmap position y coordinate.

**Returns**

The bitmap position y coordinate.

**7.179.4.8 getBoundingRect()**

```
Rect getBoundingRect ( ) const [protected]
```

Gets bounding rectangle of the transformed bitmap.

**Returns**

The bounding rectangle.

**7.179.4.9 getCameraDistance()**

```
float getCameraDistance ( ) const [inline], [virtual]
```

Gets camera distance.

**Returns**

The camera distance.

**7.179.4.10 getCameraX()**

```
float getCameraX ( ) const [inline], [virtual]
```

Gets camera x coordinate.

**Returns**

The camera x coordinate.

**7.179.4.11 getCameraY()**

```
float getCameraY ( ) const [inline], [virtual]
```

Gets camera y coordinate.

**Returns**

The camera y coordinate.

**7.179.4.12 getOrigoX()**

```
float getOrigoX ( ) const [inline], [virtual]
```

Gets transformation origo x coordinate.

**Returns**

The transformation origo x coordinate.

**7.179.4.13 getOrigoY()**

```
float getOrigoY ( ) const [inline], [virtual]
```

Gets transformation origo y coordinate.

**Returns**

The transformation origo y coordinate.

**7.179.4.14 getOrigoZ()**

```
float getOrigoZ ( ) const [inline], [virtual]
```

Gets transformation origo z coordinate.

**Returns**

The transformation origo z coordinate.

**7.179.4.15 getRenderingAlgorithm()**

```
RenderingAlgorithm getRenderingAlgorithm ( ) const [inline], [virtual]
```

Gets the algorithm used when rendering.

**Returns**

The algorithm used when rendering.

**7.179.4.16 getScale()**

```
float getScale ( ) const [inline], [virtual]
```

Gets the scale.

**Returns**

The scale.

#### 7.179.4.17 `getSolidRect()`

```
Rect getSolidRect ( ) const [virtual]
```

Gets solid rectangle.

##### Returns

largest possible solid rect.

##### See also

[Drawable::getSolidRect\(\)](#)

Implements [Drawable](#).

#### 7.179.4.18 `getType()`

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

##### Returns

TYPE\_TEXTUREMAPPER.

Reimplemented from [Widget](#).

#### 7.179.4.19 `getX0()`

```
float getX0 ( ) const [inline], [virtual]
```

Get the x coordinate of the top left corner of the transformed bitmap.

##### Returns

The X0 coordinate.

#### 7.179.4.20 `getX1()`

```
float getX1 ( ) const [inline], [virtual]
```

Get the x coordinate of the top right corner of the transformed bitmap.

##### Returns

The X1 coordinate.

**7.179.4.21 getX2()**

```
float getX2 ( ) const [inline], [virtual]
```

Get the x coordinate of the bottom right of the transformed bitmap.

**Returns**

The X2 coordinate.

**7.179.4.22 getX3()**

```
float getX3 ( ) const [inline], [virtual]
```

Get the x coordinate of the bottom left corner of the transformed bitmap.

**Returns**

The X3 coordinate.

**7.179.4.23 getXAngle()**

```
float getXAngle ( ) const [inline], [virtual]
```

Get x angle.

**Returns**

The x angle.

**7.179.4.24 getY0()**

```
float getY0 ( ) const [inline], [virtual]
```

Get the y coordinate of the top left corner of the transformed bitmap.

**Returns**

The Y0 coordinate.

**7.179.4.25 getY1()**

```
float getY1 ( ) const [inline], [virtual]
```

Get the y coordinate of the top right corner of the transformed bitmap.

**Returns**

The Y1 coordinate.



**7.179.4.26 getY2()**

```
float getY2 ( ) const [inline], [virtual]
```

Get the y coordinate of the bottom right corner of the transformed bitmap.

**Returns**

The Y2 coordinate.

**7.179.4.27 getY3()**

```
float getY3 ( ) const [inline], [virtual]
```

Get the y coordinate of the bottom left corner of the transformed bitmap.

**Returns**

The Y3 coordinate.

**7.179.4.28 getYAngle()**

```
float getYAngle ( ) const [inline], [virtual]
```

Get y angle.

**Returns**

The y angle.

**7.179.4.29 getZ0()**

```
float getZ0 ( ) const [inline], [virtual]
```

Get the z coordinate of the top left corner of the transformed bitmap.

**Returns**

The Z0 coordinate.

**7.179.4.30 getZ1()**

```
float getZ1 ( ) const [inline], [virtual]
```

Get the z coordinate of the top right corner of the transformed bitmap.

**Returns**

The Z1 coordinate.

**7.179.4.31 getZ2()**

```
float getZ2 ( ) const [inline], [virtual]
```

Get the z coordinate of the bottom right corner of the transformed bitmap.

**Returns**

The Z2 coordinate.

**7.179.4.32 getZ3()**

```
float getZ3 ( ) const [inline], [virtual]
```

Get the z coordinate of the bottom left corner of the transformed bitmap.

**Returns**

The Z3 coordinate.

**7.179.4.33 getZAngle()**

```
float getZAngle ( ) const [inline], [virtual]
```

Get z angle.

**Returns**

The z angle.

**7.179.4.34 lookupRenderVariant()**

```
RenderingVariant lookupRenderVariant ( ) const [protected]
```

Returns the rendering variant based on the bitmap format, alpha value and rendering algorithm.

**Returns**

The RenderingVariant.

**7.179.4.35 setAlpha()**

```
void setAlpha (
    uint8_t a ) [inline]
```

Sets the global alpha blending value.

**Parameters**

<i>a</i>	new alpha.
----------	------------

**7.179.4.36 setBitmap()**

```
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**

<i>bmp</i>	The bitmap to be used by the widget.
------------	--------------------------------------

**7.179.4.37 setBitmapPosition() [1/2]**

```
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**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

**7.179.4.38 setBitmapPosition() [2/2]**

```
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**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

**7.179.4.39 setCamera()**

```
void setCamera (
    float x,
    float y ) [inline], [virtual]
```

Sets the camera coordinate.

**Parameters**

<i>x</i>	The x coordinate for the camera.
<i>y</i>	The y coordinate for the camera.

**7.179.4.40 setCameraDistance()**

```
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**

<i>d</i>	The new camera distance.
----------	--------------------------

**7.179.4.41 setOrigo() [1/2]**

```
void setOrigo (  
    float x,  
    float y,  
    float z ) [inline], [virtual]
```

Sets the transformation origo.

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>z</i>	The z coordinate.

**7.179.4.42 setOrigo() [2/2]**

```
void setOrigo (  
    float x,  
    float y ) [inline], [virtual]
```

Sets the transformation origo.

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

**7.179.4.43 setRenderingAlgorithm()**

```
void setRenderingAlgorithm (
    RenderingAlgorithm algorithm ) [inline], [virtual]
```

Sets the algorithm to be used. Default setting is NEAREST\_NEIGHBOR.

**Parameters**

<i>algorithm</i>	The algorithm to use when rendering.
------------------	--------------------------------------

**7.179.4.44 setScale()**

```
void setScale (
    float scale ) [virtual]
```

Sets the scale of the image.

**Parameters**

<i>scale</i>	The new scale value.
--------------	----------------------

**7.179.4.45 updateAngles()**

```
void updateAngles (
    float xAngle,
    float yAngle,
    float zAngle ) [virtual]
```

Updates the angles of the image.

**Parameters**

<i>xAngle</i>	The new x Angle.
<i>yAngle</i>	The new y Angle.
<i>zAngle</i>	The new x Angle.

**7.179.4.46 updateXAngle()**

```
void updateXAngle (
    float xAngle ) [inline], [virtual]
```

Updates the x coordinate angle described by xAngle.

**Parameters**

<i>xAngle</i>	The new x angle.
---------------	------------------

**7.179.4.47 updateYAngle()**

```
void updateYAngle (
    float yAngle ) [inline], [virtual]
```

Updates the y coordinate angle described by yAngle.

**Parameters**

<i>yAngle</i>	The new y angle.
---------------	------------------

**7.179.4.48 updateZAngle()**

```
void updateZAngle (
    float zAngle ) [inline], [virtual]
```

Updates the z coordinate angle described by zAngle.

**Parameters**

<i>zAngle</i>	The new z angle.
---------------	------------------

**7.180 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.*

**7.181 TiledImage Class Reference**

Simple widget capable of showing a tiled bitmap.

```
#include <touchgfx/widgets/TiledImage.hpp>
```

## Public Member Functions

- [TiledImage](#) (const [Bitmap](#) &bmp=[Bitmap](#)())  
*Default Constructor.*
- virtual void [setBitmap](#) (const [Bitmap](#) &bmp)  
*Sets the bitmap ID for this [TiledImage](#).*
- virtual void [setOffset](#) (int16\_t x, int16\_t y)  
*Sets an offset into the bitmap where the tile drawing should start.*
- virtual void [setXOffset](#) (int16\_t x)  
*Sets x offset into the bitmap where the tile drawing should start.*
- virtual void [setYOffset](#) (int16\_t y)  
*Sets y offset into the bitmap where the tile drawing should start.*
- virtual void [getOffset](#) (int16\_t &x, int16\_t &y)  
*Gets the offset into the bitmap where the tile drawing should start.*
- virtual int16\_t [getXOffset](#) ()  
*Get x coordinate offset.*
- virtual int16\_t [getYOffset](#) ()  
*Get y coordinate offset.*
- virtual void [draw](#) (const [Rect](#) &invalidatedArea) const  
*Draws the image.*
- virtual [Rect](#) [getSolidRect](#) () const  
*Gets the largest solid (non-transparent) rectangle.*
- virtual uint16\_t [getType](#) () const  
*For GUI testing only.*

## Protected Attributes

- int16\_t [xOffset](#)  
*The X offset into the bitmap to start drawing.*
- int16\_t [yOffset](#)  
*The Y offset into the bitmap to start drawing.*

## Additional Inherited Members

### 7.181.1 Detailed Description

Simple widget capable of showing a tiled bitmap. This means that when [TiledImage](#) is larger than the provided [Bitmap](#), the [Bitmap](#) is repeated over and over horizontally and vertically. The bitmap can be alpha-blended with the background and have areas of transparency.

See also

[Image](#)

### 7.181.2 Constructor & Destructor Documentation

### 7.181.2.1 TiledImage()

```
TiledImage (
    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

<i>bmp</i>	The bitmap to display.
------------	------------------------

### 7.181.3 Member Function Documentation

#### 7.181.3.1 draw()

```
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

<i>invalidatedArea</i>	The rectangle to draw, with coordinates relative to this drawable.
------------------------	--------------------------------------------------------------------

Reimplemented from [Image](#).

#### 7.181.3.2 getOffset()

```
void getOffset (
    int16_t & x,
    int16_t & y ) [virtual]
```

Gets the offset into the bitmap where the tile drawing should start. Please note that the offsets set using `setOffset` have been normalized.

## Parameters

out	<i>x</i>	The x coordinate offset.
out	<i>y</i>	The y coordinate offset.

## See also

[getXOffset](#)  
[getYOffset](#)

#### 7.181.3.3 getSolidRect()

```
Rect getSolidRect ( ) const [virtual]
```

Gets the largest solid (non-transparent) rectangle. This value is pre-calculated by the image converter tool.

**Returns**

The largest solid (non-transparent) rectangle.

Reimplemented from [Image](#).

**7.181.3.4 getType()**

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

**Returns**

TYPE\_IMAGE.

Reimplemented from [Image](#).

**7.181.3.5 getXOffset()**

```
int16_t getXOffset ( ) [virtual]
```

**Returns**

The x coordinate offset.

**See also**

[getYOffset](#)  
[getOffset](#)

**7.181.3.6 getYOffset()**

```
int16_t getYOffset ( ) [virtual]
```

**Returns**

The y coordinate offset.

**See also**

[getXOffset](#)  
[getOffset](#)

**7.181.3.7 setBitmap()**

```
void setBitmap (
    const Bitmap & bmp ) [virtual]
```

Sets the bitmap ID for this [TiledImage](#). Updates the width and height of this widget to match that of the bitmap.

**Parameters**

<i>bmp</i>	The bitmap instance.
------------	----------------------

**See also**[Bitmap](#)

Reimplemented from [Image](#).

**7.181.3.8 setOffset()**

```
void setOffset (
    int16_t x,
    int16_t y ) [virtual]
```

Sets an offset into the bitmap where the tile drawing should start.

**Parameters**

<i>x</i>	The x coordinate offset.
<i>y</i>	The y coordinate offset.

**See also**[setXOffset](#)[setYOffset](#)**7.181.3.9 setXOffset()**

```
void setXOffset (
    int16_t x ) [virtual]
```

Sets x offset into the bitmap where the tile drawing should start.

**Parameters**

<i>x</i>	The x coordinate offset.
----------	--------------------------

**See also**[setYOffset](#)[setOffset](#)**7.181.3.10 setYOffset()**

```
void setYOffset (
    int16_t y ) [virtual]
```

Sets y offset into the bitmap where the tile drawing should start.

## Parameters

y	The y coordinate offset.
---	--------------------------

## See also

[setXOffset](#)  
[setOffset](#)

## 7.182 TiledImageButtonStyle< T > Class Template Reference

A tiled image button style.

```
#include <touchgfx/containers/buttons/TiledImageButtonStyle.hpp>
```

### Public Member Functions

- [TiledImageButtonStyle](#) ()  
*Default constructor.*
- virtual [~TiledImageButtonStyle](#) ()  
*Destructor.*
- virtual void [setWidth](#) (int16\_t width)  
*Sets a width.*
- virtual void [setHeight](#) (int16\_t height)  
*Sets a height.*
- virtual void [setTileBitmaps](#) (const [Bitmap](#) &bmpReleased, const [Bitmap](#) &bmpPressed)  
*Sets tile bitmaps.*
- virtual void [setTileOffset](#) (int16\_t x, int16\_t y)  
*Sets an offset into the bitmap where the tile drawing should start.*

### Protected Member Functions

- virtual void [handlePressedUpdated](#) ()  
*Handles the pressed updated.*
- virtual void [handleAlphaUpdated](#) ()  
*Handles the alpha updated.*

### Protected Attributes

- [TiledImage](#) tiledImage  
*The tiled image.*
- [Bitmap](#) upTile  
*The image to display when button is released.*
- [Bitmap](#) downTile  
*The image to display when button is pressed.*

### 7.182.1 Detailed Description

```
template<class T>
class touchgfx::TiledImageButtonStyle< T >
```

An tiled image button style. This class is supposed to be used with one of the `ButtonTrigger` classes to create a functional button. This class will show one of two tiled images depending on the state of the button (pressed or released).

The `TiledImageButtonStyle` does not set the size of the enclosing container (normally `AbstractButtonContainer`) to the size of the pressed `Bitmap`. This can be overridden by calling `setWidth/setHeight` after setting the bitmaps.

#### Template Parameters

<code>T</code>	Generic type parameter. Typically a <code>AbstractButtonContainer</code> subclass.
----------------	------------------------------------------------------------------------------------

See also

[AbstractButtonContainer](#)

### 7.182.2 Member Function Documentation

#### 7.182.2.1 `setHeight()`

```
void setHeight (
    int16_t height ) [inline], [virtual]
```

#### Parameters

<i>height</i>	The height.
---------------	-------------

#### 7.182.2.2 `setTileBitmaps()`

```
void setTileBitmaps (
    const Bitmap & bmpReleased,
    const Bitmap & bmpPressed ) [inline], [virtual]
```

#### Parameters

<i>bmpReleased</i>	The bitmap released.
<i>bmpPressed</i>	The bitmap pressed.

#### 7.182.2.3 `setTileOffset()`

```
void setTileOffset (
    int16_t x,
    int16_t y ) [inline], [virtual]
```

Sets an offset into the bitmap where the tile drawing should start.

#### Parameters

<i>x</i>	The x coordinate offset.
<i>y</i>	The y coordinate offset.

#### 7.182.2.4 setWidth()

```
void setWidth (
    int16_t width ) [inline], [virtual]
```

#### Parameters

<i>width</i>	The width.
--------------	------------

## 7.183 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

#### 7.183.1 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](#)

## 7.183.2 Constructor & Destructor Documentation

### 7.183.2.1 ToggleButton()

```
ToggleButton ( )
```

Default constructor.

## 7.183.3 Member Function Documentation

### 7.183.3.1 forceState()

```
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

<i>activeState</i>	If true, display the <code>bmpPressed</code> bitmap when not pressed. If false display the <code>bmpReleased</code> bitmap.
--------------------	-----------------------------------------------------------------------------------------------------------------------------

### 7.183.3.2 getState()

```
bool getState ( ) const [inline]
```

Gets the state.

#### Returns

true if state is currently active.

### 7.183.3.3 getType()

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

#### Returns

TYPE\_TOGGLEBUTTON.

Reimplemented from [Button](#).



## 7.183.3.4 handleClickEvent()

```
void handleClickEvent (
    const ClickEvent & event ) [virtual]
```

Overrides handleClickEvent in order to swap the bitmaps after being clicked.

## Parameters

<i>event</i>	The event to handle.
--------------	----------------------

Reimplemented from [AbstractButton](#).

## 7.183.3.5 setBitmaps()

```
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

<i>bmpReleased</i>	The bitmap to show in the "normal" state, ie when button is not pressed.
<i>bmpPressed</i>	The bitmap to show when the button is pressed.

## See also

[Button::setBitmaps](#)

Reimplemented from [Button](#).

## 7.184 ToggleButtonTrigger Class Reference

A toggle button trigger.

```
#include <touchgfx/containers/buttons/ToggleButtonTrigger.hpp>
```

## Public Member Functions

- [ToggleButtonTrigger](#) ()  
*Default constructor.*
- virtual [~ToggleButtonTrigger](#) ()  
*Destructor.*
- void [forceState](#) (bool activeState)  
*Force the button into a specific state.*
- void [setToggleCanceled](#) (bool isToggleCanceled)

*Sets toggle canceled.*

- bool [getToggleCanceled](#) ()

*Gets toggle canceled.*

- virtual void [handleClickEvent](#) (const [ClickEvent](#) &event)

*Handles the click event described by event.*

## Protected Attributes

- bool [toggleCanceled](#)

*True if toggle canceled.*

## Additional Inherited Members

### 7.184.1 Detailed Description

A toggle button trigger. This trigger will create a button that reacts on clicks. This means it will call the action when it gets a touch released event.

The [ToggleButtonTrigger](#) will stay in pressed state until it is clicked again.

The [ToggleButtonTrigger](#) can be combined with one or more of the ButtonStyle classes to create a functional button.

### 7.184.2 Member Function Documentation

#### 7.184.2.1 forceState()

```
void forceState (
    bool activeState ) [inline]
```

Use this function to force the button in one of the two possible states. If button is forced to the active state, then [AbstractButtonContainer](#) will be in a pressed state.

#### Parameters

<i>activeState</i>	If true, the <a href="#">AbstractButtonContainer</a> will appear pressed.
--------------------	---------------------------------------------------------------------------

#### 7.184.2.2 getToggleCanceled()

```
bool getToggleCanceled ( ) [inline]
```

#### Returns

True if it succeeds, false if it fails.

#### 7.184.2.3 handleClickEvent()

```
void handleClickEvent (
    const ClickEvent & event ) [inline], [virtual]
```

## Parameters

<i>event</i>	The event.
--------------	------------

Reimplemented from [Drawable](#).

## 7.184.2.4 setToggleCanceled()

```
void setToggleCanceled (
    bool isToggleCanceled ) [inline]
```

## Parameters

<i>isToggleCanceled</i>	True if is toggle canceled, false if not.
-------------------------	-------------------------------------------

## 7.185 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

## 7.185.1 Detailed Description

Invisible widget used to capture touch events. The [TouchArea](#) consumes drag events without the widget it self moving.

See also

[AbstractButton](#)

## 7.185.2 Constructor & Destructor Documentation

### 7.185.2.1 TouchArea()

```
TouchArea ( ) [inline]
```

Default constructor.

## 7.185.3 Member Function Documentation

### 7.185.3.1 draw()

```
void draw (
    const Rect & invalidatedArea ) const [inline], [virtual]
```

A [TouchArea](#) will not draw anything.

#### Parameters

<i>invalidatedArea</i>	The region of the toucharea to draw. Ignored.
------------------------	-----------------------------------------------

Implements [Drawable](#).

### 7.185.3.2 getSolidRect()

```
Rect getSolidRect ( ) const [inline], [virtual]
```

A [TouchArea](#) has no solid rectangle.

#### Returns

an empty rect.

Implements [Drawable](#).

### 7.185.3.3 getType()

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

**Returns**

TYPE\_TOUCHAREA.

Reimplemented from [AbstractButton](#).

**7.185.3.4 handleClickEvent()**

```
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**

<i>event</i>	The event to handle.
--------------	----------------------

Reimplemented from [AbstractButton](#).

**7.185.3.5 handleDragEvent()**

```
void handleDragEvent (
    const DragEvent & evt ) [inline], [virtual]
```

A [TouchArea](#) will not move when dragged.

**Parameters**

<i>evt</i>	The event to handle. Ignored.
------------	-------------------------------

Reimplemented from [Drawable](#).

**7.185.3.6 setPressedAction()**

```
void setPressedAction (
    GenericCallback< const AbstractButton & > & callback ) [inline]
```

Associates an action to be performed when the [TouchArea](#) is pressed.

**Parameters**

<i>callback</i>	The callback is given a reference to this touch area.
-----------------	-------------------------------------------------------

## 7.186 TouchButtonTrigger Class Reference

A touch button trigger.

```
#include <touchgfx/containers/buttons/TouchButtonTrigger.hpp>
```

## Public Member Functions

- [TouchButtonTrigger](#) ()  
*Default constructor.*
- virtual [~TouchButtonTrigger](#) ()  
*Destructor.*
- virtual void [handleClickEvent](#) (const [ClickEvent](#) &event)  
*Handles the click event described by event.*

## Additional Inherited Members

### 7.186.1 Detailed Description

A touch button trigger. This trigger will create a button that reacts on touches. This means it will call the action when it gets a touch pressed event.

The [TouchButtonTrigger](#) can be combined with one or more of the [ButtonStyle](#) classes to create a functional button.

### 7.186.2 Member Function Documentation

#### 7.186.2.1 handleClickEvent()

```
void handleClickEvent (
    const ClickEvent & event ) [inline], [virtual]
```

#### Parameters

<i>event</i>	The event.
--------------	------------

Reimplemented from [Drawable](#).

## 7.187 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.*

### 7.187.1 Detailed Description

Class [TouchCalibration](#) is responsible for translating coordinates ([Point](#)) based on matrix of calibration values.

## 7.187.2 Member Function Documentation

### 7.187.2.1 setCalibrationMatrix()

```
static void setCalibrationMatrix (
    const Point * ref,
    const Point * scr ) [static]
```

Initializes the calibration matrix based on reference and measured values.

#### Parameters

<i>ref</i>	Pointer to array of three reference points.
<i>scr</i>	Pointer to array of three measured points.

### 7.187.2.2 translatePoint()

```
static void translatePoint (
    Point & p ) [static]
```

Translates the specified point using the matrix. If matrix has not been initialized, p is not modified.

#### Parameters

<i>in, out</i>	<i>p</i>	The point to translate.
----------------	----------	-------------------------

## 7.188 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.*

### 7.188.1 Detailed Description

Basic Touch Controller interface.

### 7.188.2 Constructor & Destructor Documentation

### 7.188.2.1 ~TouchController()

```
~TouchController ( ) [inline], [virtual]
```

Destructor.

## 7.188.3 Member Function Documentation

### 7.188.3.1 init()

```
void init ( ) [pure virtual]
```

Initializes touch controller.

Implemented in [I2CTouchController](#), [SDL2TouchController](#), [SDLTouchController](#), and [NoTouchController](#).

### 7.188.3.2 sampleTouch()

```
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	<i>x</i>	The x position of the touch
out	<i>y</i>	The y position of the touch

#### Returns

True if a touch has been detected, otherwise false.

Implemented in [I2CTouchController](#), [NoTouchController](#), [SDL2TouchController](#), and [SDLTouchController](#).

## 7.189 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.*

### 7.189.1 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](#)

### 7.189.2 Constructor & Destructor Documentation

#### 7.189.2.1 Transition()

```
Transition ( ) [inline]
```

Constructs the [Transition](#).

#### 7.189.2.2 ~Transition()

```
~Transition ( ) [inline], [virtual]
```

Destructor.

### 7.189.3 Member Function Documentation

#### 7.189.3.1 handleTickEvent()

```
void handleTickEvent ( ) [inline], [virtual]
```

Called for every tick when transitioning. Base does nothing.

Reimplemented in [CoverTransition< templateDirection >](#), [SlideTransition< templateDirection >](#), and [NoTransition](#).

### 7.189.3.2 init()

```
void init ( ) [inline], [virtual]
```

Initializes the transition. Called after the c.tor. when the application changes the transition. Base version does nothing.

Reimplemented in [CoverTransition< templateDirection >](#), and [SlideTransition< templateDirection >](#).

### 7.189.3.3 isDone()

```
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.

### 7.189.3.4 setScreenContainer()

```
void setScreenContainer (
    Container & cont ) [inline], [virtual]
```

Sets the screen container. Is used by [Screen](#) to enable the transition to access the container.

#### Parameters

in	<i>cont</i>	The container the transition should have access to.
----	-------------	-----------------------------------------------------

### 7.189.3.5 tearDown()

```
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 [CoverTransition< templateDirection >](#), and [SlideTransition< templateDirection >](#).

## 7.190 TwoWildcardTextButtonStyle< T > Class Template Reference

A wildcard text button style.

```
#include <TwoWildcardTextButtonStyle.hpp>
```

## Public Member Functions

- [TwoWildcardTextButtonStyle](#) ()  
*Default constructor.*
- virtual [~TwoWildcardTextButtonStyle](#) ()  
*Destructor.*
- void [setTwoWildcardText](#) ([TypedText](#) t)  
*Sets wildcard text.*
- void [setTwoWildcardTextX](#) (int16\_t x)  
*Sets wildcard text x coordinate.*
- void [setTwoWildcardTextY](#) (int16\_t y)  
*Sets wildcard text y coordinate.*
- void [setTwoWildcardTextXY](#) (int16\_t x, int16\_t y)  
*Sets wildcard text xy.*
- void [setTwoWildcardTextPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets text position.*
- void [setTwoWildcardTextRotation](#) ([TextRotation](#) rotation)  
*Sets wildcard text rotation.*
- void [setWildcardTextBuffer1](#) (const [Unicode::UnicodeChar](#) \*value)  
*Sets the first wildcard in the text.*
- void [setWildcardTextBuffer2](#) (const [Unicode::UnicodeChar](#) \*value)  
*Sets the second wildcard in the text.*
- void [setTwoWildcardTextColors](#) ([colortype](#) newColorReleased, [colortype](#) newColorPressed)  
*Sets wild card text colors.*

## Protected Member Functions

- virtual void [handlePressedUpdated](#) ()  
*Handles the pressed updated.*
- virtual void [handleAlphaUpdated](#) ()  
*Handles the alpha updated.*

## Protected Attributes

- [TextAreaWithTwoWildcards](#) twoWildcardText  
*The wildcard text.*
- [colortype](#) colorReleased  
*The color released.*
- [colortype](#) colorPressed  
*The color pressed.*

### 7.190.1 Detailed Description

```
template<class T>
class touchgfx::TwoWildcardTextButtonStyle< T >
```

An wildcard text button style. This class is supposed to be used with one of the [ButtonTrigger](#) classes to create a functional button. This class will show a text with a wildcard in one of two colors depending on the state of the button (pressed or released).

The [TwoWildcardTextButtonStyle](#) does not set the size of the enclosing container (normally [AbstractButton](#)↔[Container](#)). The size must be set manually.

To get a background behind the text, use [TwoWildcardTextButtonStyle](#) together with e.g. [ImageButtonStyle](#): [TwoWildcardTextButtonStyle](#)<[ImageButtonStyle](#)<[ClickButtonTrigger](#)> > myButton;

The position of the text can be adjusted with `setTwoWildcardTextXY` (default is centered).

#### Template Parameters

<i>T</i>	Generic type parameter. Typically a <a href="#">AbstractButtonContainer</a> subclass.
----------	---------------------------------------------------------------------------------------

#### See also

[AbstractButtonContainer](#)

#### Template Parameters

<i>T</i>	Generic type parameter.
----------	-------------------------

## 7.190.2 Member Function Documentation

### 7.190.2.1 setTwoWildcardText()

```
void setTwoWildcardText (
    TypedText t ) [inline]
```

#### Parameters

<i>t</i>	A <a href="#">TypedText</a> to process.
----------	-----------------------------------------

### 7.190.2.2 setTwoWildcardTextColors()

```
void setTwoWildcardTextColors (
    color_t newColorReleased,
    color_t newColorPressed ) [inline]
```

#### Parameters

<i>newColorReleased</i>	The new color released.
<i>newColorPressed</i>	The new color pressed.

### 7.190.2.3 setTwoWildcardTextPosition()

```
void setTwoWildcardTextPosition (
    int16_t x,
    int16_t y,
    int16_t width,
```

```
int16_t height ) [inline]
```

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>width</i>	The width of the text.
<i>height</i>	The height of the text.

**7.190.2.4 setTwoWildcardTextRotation()**

```
void setTwoWildcardTextRotation (
    TextRotation rotation ) [inline]
```

**Parameters**

<i>rotation</i>	The rotation.
-----------------	---------------

**7.190.2.5 setTwoWildcardTextX()**

```
void setTwoWildcardTextX (
    int16_t x ) [inline]
```

**Parameters**

<i>x</i>	The x coordinate.
----------	-------------------

**7.190.2.6 setTwoWildcardTextXY()**

```
void setTwoWildcardTextXY (
    int16_t x,
    int16_t y ) [inline]
```

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

**7.190.2.7 setTwoWildcardTextY()**

```
void setTwoWildcardTextY (
    int16_t y ) [inline]
```

## Parameters

<code>y</code>	The y coordinate.
----------------	-------------------

7.190.2.8 `setWildcardTextBuffer1()`

```
void setWildcardTextBuffer1 (
    const Unicode::UnicodeChar * value ) [inline]
```

Sets the first wildcard in the text. Must be a zero-terminated UnicodeChar array.

## Parameters

<code>value</code>	A pointer to the UnicodeChar to set the wildcard to.
--------------------	------------------------------------------------------

7.190.2.9 `setWildcardTextBuffer2()`

```
void setWildcardTextBuffer2 (
    const Unicode::UnicodeChar * value ) [inline]
```

Sets the second wildcard in the text. Must be a zero-terminated UnicodeChar array.

## Parameters

<code>value</code>	A pointer to the UnicodeChar to set the wildcard to.
--------------------	------------------------------------------------------

## 7.191 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.*

#### 7.191.1 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](#)

#### 7.191.2 Constructor & Destructor Documentation

##### 7.191.2.1 TypedText()

```
TypedText (
    const TypedTextId id = TYPED\_TEXT\_INVALID ) [inline], [explicit]
```

Construct a typed text.

Parameters

<i>id</i>	The id of the <a href="#">TypedText</a> .
-----------	-------------------------------------------

#### 7.191.3 Member Function Documentation

##### 7.191.3.1 getAlignment()

```
Alignment getAlignment ( ) const [inline]
```

Gets the alignment associated with this [TypedText](#).

**Returns**

The alignment.

**7.191.3.2 getFont()**

```
const Font * getFont ( ) const [inline]
```

Gets the font associated with this [TypedText](#).

**Returns**

The font.

**7.191.3.3 getId()**

```
TypedTextId getId ( ) const [inline]
```

Gets the id of the typed text.

**Returns**

The id.

**7.191.3.4 getText()**

```
const Unicode::UnicodeChar * getText ( ) const [inline]
```

Gets the text associated with this [TypedText](#).

**Returns**

The text.

**7.191.3.5 getTextDirection()**

```
TextDirection getTextDirection ( ) const [inline]
```

Gets the text direction associated with this [TypedText](#).

**Returns**

The alignment.

**7.191.3.6 hasValidId()**

```
bool hasValidId ( ) const [inline]
```

Has the [TypedText](#) been set to a proper value.



**Returns**

Is the id valid.

**7.191.3.7 registerTexts()**

```
static void registerTexts (
    const Texts * t ) [inline], [static]
```

Registers an array of texts.

**Parameters**

<i>t</i>	The array of texts.
----------	---------------------

**7.191.3.8 registerTypedTextDatabase()**

```
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**

<i>data</i>	A reference to the <a href="#">TypedTextData</a> storage array.
<i>f</i>	The fonts associated with the array.
<i>n</i>	The number of typed texts in the array.

**7.192 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.*

**7.192.1 Detailed Description**

The data structure for typed texts.

## 7.193 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.*

### 7.193.1 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.

### 7.193.2 Constructor & Destructor Documentation

#### 7.193.2.1 ~UIEventListener()

```
~UIEventListener ( ) [inline], [virtual]
```

Destructor.

### 7.193.3 Member Function Documentation

#### 7.193.3.1 handleClickEvent()

```
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

<i>event</i>	The event data.
--------------	-----------------

Reimplemented in [Application](#).

#### 7.193.3.2 handleDragEvent()

```
void handleDragEvent (
    const DragEvent & event ) [inline], [virtual]
```

This handler is invoked when a drag event has been detected by the system.

##### Parameters

<i>event</i>	The event data.
--------------	-----------------

Reimplemented in [Application](#).

#### 7.193.3.3 handleGestureEvent()

```
void handleGestureEvent (
    const GestureEvent & event ) [inline], [virtual]
```

This handler is invoked when a gesture event has been detected by the system.

##### Parameters

<i>event</i>	The event data.
--------------	-----------------

Reimplemented in [Application](#).

#### 7.193.3.4 handleKeyEvent()

```
void handleKeyEvent (
    uint8_t c ) [inline], [virtual]
```

This handler is invoked when a key (or button) event has been detected by the system.

##### Parameters

<i>c</i>	The key or button pressed.
----------	----------------------------

Reimplemented in [Application](#).

#### 7.193.3.5 handlePendingScreenTransition()

```
void handlePendingScreenTransition ( ) [inline], [virtual]
```

This handler is invoked when a change screen event is pending.

Reimplemented in [Application](#), and [MVPApplication](#).

### 7.193.3.6 handleTickEvent()

```
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](#).

## 7.194 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 to UnicodeChar version.*
- static uint16\_t [strncpy](#) ([UnicodeChar](#) \*RESTRICT dst, const char \*RESTRICT src, uint16\_t maxchars)  
*Copy a string to a destination buffer, char to UnicodeChar 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](#) \* [vsnprintf](#) ([UnicodeChar](#) \*dst, uint16\_t dstSize, const [UnicodeChar](#) \*format, va\_list pArg)  
*Variant of snprintf.*
- static [UnicodeChar](#) \* [snprintf](#) ([UnicodeChar](#) \*dst, uint16\_t dstSize, const char \*format,...)  
*Variant of snprintf.*
- static [UnicodeChar](#) \* [vsnprintf](#) ([UnicodeChar](#) \*dst, uint16\_t dstSize, const char \*format, va\_list pArg)  
*Variant of snprintf.*
- static [UnicodeChar](#) \* [snprintfFloats](#) ([UnicodeChar](#) \*dst, uint16\_t dstSize, const [UnicodeChar](#) \*format, const float \*values)  
*Variant of snprintf for floats only.*
- static [UnicodeChar](#) \* [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 `uint16_t fromUTF8 (const uint8_t *utf8, UnicodeChar *dst, uint16_t maxchars)`  
*Convert a string from `utf8` to unicode.*
- static `uint16_t toUTF8 (const UnicodeChar *unicode, uint8_t *utf8, uint16_t maxbytes)`  
*Converts a string from unicode to `utf8`.*

### 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.*

## 7.194.1 Member Typedef Documentation

### 7.194.1.1 UnicodeChar

`uint16_t UnicodeChar`

Use the `UnicodeChar` typename when referring to strings.

## 7.194.2 Member Function Documentation

### 7.194.2.1 atoi()

```
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

<code>s</code>	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.

### 7.194.2.2 fromUTF8()

```
static uint16_t fromUTF8 (
    const uint8_t * utf8,
    UnicodeChar * dst,
    uint16_t maxchars ) [static]
```

Convert a string from utf8 to unicode. The conversion stops if there is no more room in the destination or if the terminating zero character has been converted.

#### Parameters

	<i>utf8</i>	The UTF8 string.
out	<i>dst</i>	The destination buffer for the converted string.
	<i>maxchars</i>	The maximum number of chars that the dst array can hold.

#### Returns

The number of characters successfully converted from utf8 to unicode including the terminating zero.

### 7.194.2.3 itoa()

```
static void itoa (
    int32_t value,
    UnicodeChar * buffer,
    uint16_t bufferSize,
    int radix ) [static]
```

Integer to ASCII conversion.

#### Parameters

	<i>value</i>	to convert.
out	<i>buffer</i>	to place result in.
	<i>bufferSize</i>	Size of buffer (number of 16-bit values).
	<i>radix</i>	to use (8 for octal, 10 for decimal, 16 for hex)

### 7.194.2.4 snprintf() [1/2]

```
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), %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	<i>dst</i>	Buffer for the formatted string.
	<i>dstSize</i>	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	<i>format</i>	The format string.
	...	The values to insert in the format string.

**Returns**

pointer to the first element in the buffer where the formatted string is placed.

**See also**

[snprintfFloat](#), [snprintfFloats](#)

**7.194.2.5 snprintf()** [2/2]

```
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), %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

```
%[flags][width][.precision]X
```

Where flags can be:

```
'-': left justify the field (see width).
'+': force sign.
' ': insert space if value is positive.
'0': left pad with zeros instead of spaces (see width).
```

Where width is the desired width of the output. If the value is larger, more characters may be generated, but not more than the parameter dstSize. If width is '\*' the actual width is read from the parameters passed to this function.

Where precision is the number of number of digits after the decimal point, default is 3. Use "%.f" to not generate any numbers after the decimal point. If precision is '\*' the actual precision is read from the parameters passed to this function.

**Parameters**

out	<i>dst</i>	Buffer for the formatted string.
	<i>dstSize</i>	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	<i>format</i>	The format string.
	...	The values to insert in the format string.

**Returns**

pointer to the first element in the buffer where the formatted string is placed.

**Note**

%f is not supported by this function because floats are converted to doubles when given as parameters in a variable argument list (va\_list). Use snprintfFloat or snprintfFloats instead.

**Warning**

The format string is internally copied from at char\* to a UnicodeChar\*. This buffer has a limit of 63 characters, so if the format is longer than 63 characters, the caller must do this copying to prevent an assert from triggering:

```
touchgfx::Unicode::UnicodeChar tmpfmt[200];
touchgfx::Unicode::strncpy(tmpfmt, "Very, very, very, very, very, very, very,
very, very long format %i", 200);
touchgfx::Unicode::snprintf(dst, dstSize, tmpfmt, ...);
```

**See also**

[snprintfFloat](#), [snprintfFloats](#)

**7.194.2.6 snprintfFloat() [1/2]**

```
static UnicodeChar * snprintfFloat (
    UnicodeChar * dst,
    uint16_t dstSize,
    const UnicodeChar * format,
    const float value ) [inline], [static]
```

Variant of snprintf for one float only.

The number format supports only one %f with flags/modifiers. The following is supported:

```
%[flags][width][.precision]f
```

Where flags can be:

```
'-': left justify the field (see width).
'+': force sign.
' ': insert space if value is positive.
'#': insert decimal point even if there are not decimals.
'0': left pad with zeros instead of spaces (see width).
```

Where width is the desired width of the output. If the value is larger, more characters may be generated, but not more than the parameter dstSize.

Where precision is the number of number of digits after the decimal point, default is 3. Use "%.f" to not generate any numbers after the decimal point.

```
Unicode::UnicodeChar buffer[20];
Unicode::snprintfFloat(buffer, 20, "%6.4f", 3.14159f);
// buffer="3.1416" Unicode::snprintfFloat(buffer, 20, "%#6.f", 3.14159f);
// buffer=" 3." Unicode::snprintfFloat(buffer, 20, "%6f", 3.14159f);
// buffer=" 3.142" Unicode::snprintfFloat(buffer, 20, "%+06.f", 3.14159f);
// buffer="+00003"
```

If more control over the output is needed, see snprintfFloats which can have more than a single "%f" in the string and also supports "\*" in place of a number.



## Parameters

out	<i>dst</i>	Buffer for the formatted string.
	<i>dstSize</i>	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	<i>format</i>	The format string containing exactly one occurrence of f.
	<i>value</i>	The floating point value to insert for f.

## Returns

pointer to the first element in the buffer where the formatted string is placed.

## See also

[snprintf](#), [snprintfFloats](#)

## 7.194.2.7 snprintfFloat() [2/2]

```
static UnicodeChar * snprintfFloat (
    UnicodeChar * dst,
    uint16_t dstSize,
    const char * format,
    const float value ) [inline], [static]
```

Variant of snprintf for one float only.

The number format supports only one %f with flags/modifiers. The following is supported:

```
%[flags][width][.precision]f
```

Where flags can be:

```
'-': left justify the field (see width).
'+': force sign.
' ': insert space if value is positive.
'#': insert decimal point even if there are not decimals.
'0': left pad with zeros instead of spaces (see width).
```

Where width is the desired width of the output. If the value is larger, more characters may be generated, but not more than the parameter dstSize.

Where precision is the number of number of digits after the decimal point, default is 3. Use "%.f" to not generate any numbers after the decimal point.

```
Unicode::UnicodeChar buffer[20];
Unicode::snprintfFloat(buffer, 20, "%6.4f", 3.14159f);
// buffer="3.1416"
Unicode::snprintfFloat(buffer, 20, "%#6.f", 3.14159f);
// buffer=" 3."
Unicode::snprintfFloat(buffer, 20, "%6f", 3.14159f);
// buffer=" 3.142"
Unicode::snprintfFloat(buffer, 20, "%+06.f", 3.14159f);
// buffer="+00003"
```

If more control over the output is needed, see snprintfFloats which can have more than a single "%f" in the string and also supports "\*" in place of a number.

## Parameters

out	<i>dst</i>	Buffer for the formatted string.
	<i>dstSize</i>	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	<i>format</i>	The format string containing exactly on occurrence of f.
	<i>value</i>	The floating point value to insert for f.

## Returns

pointer to the first element in the buffer where the formatted string is placed.

## Warning

The format string is internally copied from at char\* to a UnicodeChar\*. This buffer has a limit of 63 characters, so if the format is longer than 63 characters, the caller must do this copying to prevent an assert from triggering:

```
touchgfx::Unicode::UnicodeChar tmpfmt[200];
touchgfx::Unicode::strncpy(tmpfmt, "Very, very, very, very, very, very, very, very,
    very, very long format %f", 200);
touchgfx::Unicode::snprintfFloats(dst, dstSize, tmpfmt, values);
```

## See also

[snprintf](#), [snprintfFloats](#)

## 7.194.2.8 snprintfFloats() [1/2]

```
static UnicodeChar * snprintfFloats (
    UnicodeChar * dst,
    uint16_t dstSize,
    const UnicodeChar * format,
    const float * values ) [static]
```

Variant of snprintf for floats only.

The format supports several %f with flags/modifiers. The following is supported:

```
%[flags][width][.precision]f
```

Where flags can be:

```
'-': left justify the field (see width).
'+': force sign.
' ': insert space if value is positive
'#': insert decimal point even if there are not decimals
'0': left pad with zeros instead of spaces (see width)
```

Where width is the desired width of the output. If the value is larger, more characters may be generated, but not more than the parameter dstSize. If width is '\*' the actual width is read from the list of values passed to this function.

Where precision is the number of number of digits after the decimal point, default is

1. Use "%.f" to not generate any numbers after the decimal point. If precision is '\*' the actual precision is read from the list of values passed to this function.

```
float param1[3] = { 6.0f, 4.0f, 3.14159f };
Unicode::snprintfFloats(buffer, 20, "%*.*f", param1);
// buffer="3.1416"
float param2[2] = { 3.14159f, -123.4f };
Unicode::snprintfFloats(buffer, 20, "%f %f", param2);
// buffer="3.142 -123.400"
```

## Parameters

out	<i>dst</i>	Buffer for the formatted string.
	<i>dstSize</i>	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	<i>format</i>	The format string containing f's.
in	<i>values</i>	The floating point values to insert for f. The number of elements in the array must match the number of f's in the format string.

## Returns

pointer to the first element in the buffer where the formatted string is placed.

## See also

[snprintf](#), [snprintfFloat](#)

7.194.2.9 `snprintfFloats()` [2/2]

```
static UnicodeChar * snprintfFloats (
    UnicodeChar * dst,
    uint16_t dstSize,
    const char * format,
    const float * values ) [static]
```

Variant of `snprintf` for floats only.

The format supports several %f with flags/modifiers. The following is supported:

```
%[flags][width][.precision]f
```

Where flags can be:

```
'-': left justify the field (see width).
'+': force sign.
' ': insert space if value is positive.
'#': insert decimal point even if there are not decimals.
'0': left pad with zeros instead of spaces (see width).
```

Where width is the desired width of the output. If the value is larger, more characters may be generated, but not more than the parameter `dstSize`. If width is '\*' the actual width is read from the list of values passed to this function.

Where precision is the number of number of digits after the decimal point, default is

1. Use "%f" to not generate any numbers after the decimal point. If precision is '\*' the actual precision is read from the list of values passed to this function.

```
float param1[3] = { 6.0f, 4.0f, 3.14159f };
Unicode::snprintfFloats(buffer, 20, "%*.f", param1);
// buffer="3.1416"
float param2[2] = { 3.14159f, -123.4f };
Unicode::snprintfFloats(buffer, 20, "%f %f", param2);
// buffer="3.142 -123.400"
```

## Parameters

out	<i>dst</i>	Buffer for the formatted string.
	<i>dstSize</i>	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	<i>format</i>	The format string containing f's.
in	<i>values</i>	The floating point values to insert for f. The number of elements in the array must match the number of f's in the format string.

**Returns**

pointer to the first element in the buffer where the formatted string is placed.

@warning The format string is internally copied from `char*` to a `UnicodeChar*`. This buffer has a limit of 63 characters, so if the format is longer than 63 characters, the caller must do this copying to prevent an assert from triggering:

```
touchgfx::Unicode::UnicodeChar tmpfmt[200];
touchgfx::Unicode::strncpy(tmpfmt, "Very, very, very, very, very, very, very,
    very, very long format %f", 200);
touchgfx::Unicode::snprintfFloats(dst, dstSize, tmpfmt, values);
```

**See also**

[snprintf](#), [snprintfFloat](#)

**7.194.2.10 strlen()** [1/2]

```
static uint16_t strlen (
    const UnicodeChar * str ) [static]
```

Gets the length of a 0-terminated unicode string.

**Parameters**

<i>str</i>	The string in question.
------------	-------------------------

**Returns**

Length of string.

**7.194.2.11 strlen()** [2/2]

```
static uint16_t strlen (
    const char * str ) [static]
```

Gets the length of a 0-terminated string.

**Parameters**

<i>str</i>	The string.
------------	-------------

**Returns**

Length of string.

**7.194.2.12 strncmp()**

```
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

<i>str1</i>	The first string.
<i>str2</i>	The second string.
<i>maxchars</i>	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.

#### 7.194.2.13 strncmp\_ignore\_white\_spaces()

```
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

<i>str1</i>	The first string.
<i>str2</i>	The second string.
<i>maxchars</i>	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.

#### 7.194.2.14 strncpy() [1/2]

```
static uint16_t strncpy (  
    UnicodeChar *RESTRICT dst,  
    const UnicodeChar *RESTRICT src,  
    uint16_t maxchars ) [static]
```

Copy a string to a destination buffer, UnicodeChar to UnicodeChar version. Stops if it encounters a zero-termination, in which case the zero-termination is included in the destination string. Otherwise copies maxchars.

**Parameters**

out	<i>dst</i>	The destination buffer. Must have a size of at least maxchars.
in	<i>src</i>	The source string (UnicodeChars)
	<i>maxchars</i>	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 UnicodeChars of src, the string placed in destination will NOT be zero-terminated!

**7.194.2.15 strncpy()** [2/2]

```
static uint16_t strncpy (  
    UnicodeChar *RESTRICT dst,  
    const char *RESTRICT src,  
    uint16_t maxchars ) [static]
```

Copy a string to a destination buffer, char to UnicodeChar version. Stops if it encounters a zero-termination, in which case the zero-termination is included in the destination string. Otherwise copies maxchars.

**Parameters**

out	<i>dst</i>	The destination buffer. Must have a size of at least maxchars.
in	<i>src</i>	The source string as an array of chars.
	<i>maxchars</i>	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 destination will NOT be zero-terminated!

**7.194.2.16 toUTF8()**

```
static uint16_t toUTF8 (  
    const UnicodeChar * unicode,  
    uint8_t * utf8,  
    uint16_t maxbytes ) [static]
```

Converts a string from unicode to utf8. The conversion stops if there is no more room in the destination or if the terminating zero character has been converted. U+10000 through U+10FFFF are skipped.

## Parameters

	<i>unicode</i>	The unicode string.
out	<i>utf8</i>	The destination buffer for the converted string.
	<i>maxbytes</i>	The maximum number of bytes that the utf8 array can hold.

## Returns

The number of characters successfully converted from unicode to utf8 including the terminating zero.

## 7.194.2.17 utoa()

```
static void utoa (
    uint32_t value,
    UnicodeChar * buffer,
    uint16_t bufferSize,
    int radix ) [static]
```

Integer to ASCII conversion.

## Parameters

	<i>value</i>	to convert.
out	<i>buffer</i>	to place result in.
	<i>bufferSize</i>	Size of buffer (number of 16-bit values).
	<i>radix</i>	to use (8 for octal, 10 for decimal, 16 for hex)

## 7.194.2.18 vsnprintf() [1/2]

```
static UnicodeChar * vsnprintf (
    UnicodeChar * dst,
    uint16_t dstSize,
    const UnicodeChar * format,
    va_list pArg ) [static]
```

Variant of snprintf. See snprintf for details.

## Parameters

out	<i>dst</i>	Buffer for the formatted string.
	<i>dstSize</i>	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	<i>format</i>	The format string.
	<i>pArg</i>	The values to insert in the format string.

## Returns

pointer to the first element in the buffer where the formatted string is placed.



See also

[snprintf](#)

#### 7.194.2.19 vsnprintf() [2/2]

```
static UnicodeChar * vsnprintf (
    UnicodeChar * dst,
    uint16_t dstSize,
    const char * format,
    va_list pArg ) [static]
```

Variant of snprintf. See snprintf for details.

##### Parameters

out	<i>dst</i>	Buffer for the formatted string.
	<i>dstSize</i>	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	<i>format</i>	The format string.
	<i>pArg</i>	The values to insert in the format string.

##### Returns

pointer to the first element in the buffer where the formatted string is placed.

See also

[snprintf](#)

## 7.195 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](#).*
- T [quickRemoveAt](#) (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.*

### 7.195.1 Detailed Description

```
template<class T, uint16_t capacity>
class touchgfx::Vector< T, capacity >
```

A very simple container class using pre-allocated memory.

#### Template Parameters

<i>T</i>	The type of objects this container works on.
<i>capacity</i>	The maximum number of objects this container can store.

### 7.195.2 Constructor & Destructor Documentation

#### 7.195.2.1 Vector()

```
Vector ( ) [inline]
```

Default constructor. Constructs an empty vector.

### 7.195.3 Member Function Documentation

#### 7.195.3.1 add()

```
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

<i>e</i>	The element to add to the <code>Vector</code> .
----------	-------------------------------------------------

### 7.195.3.2 clear()

```
void clear ( ) [inline]
```

Clears the contents of the container. It does not destruct any of the elements in the [Vector](#).

### 7.195.3.3 contains()

```
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

<i>elem</i>	The element.
-------------	--------------

#### Returns

true if the [Vector](#) contains the element, false otherwise.

### 7.195.3.4 isEmpty()

```
bool isEmpty ( ) const [inline]
```

Query if this object is empty.

#### Returns

true if the [Vector](#) contains no elements.

### 7.195.3.5 maxCapacity()

```
uint16_t maxCapacity ( ) const [inline]
```

Query the maximum capacity of the vector.

#### Returns

The capacity the [Vector](#) was initialized with.

### 7.195.3.6 operator[]() [1/2]

```
T & operator[] (
    uint16_t idx ) [inline]
```

Index operator.

#### Parameters

<i>idx</i>	The index of the element to obtain.
------------	-------------------------------------

**Returns**

A reference to the element placed at index `idx`.

**7.195.3.7 operator[]()** [2/2]

```
const T & operator[] (
    uint16_t idx ) const [inline]
```

Const version of the index operator.

**Parameters**

<i>idx</i>	The index of the element to obtain.
------------	-------------------------------------

**Returns**

A const reference to the element placed at index `idx`.

**7.195.3.8 quickRemoveAt()**

```
T quickRemoveAt (
    uint16_t index ) [inline]
```

Removes an element at the specified index of the [Vector](#). The last element in the list is moved to the position where the element is removed.

**Parameters**

<i>index</i>	The index to remove.
--------------	----------------------

**Returns**

The value of the removed element.

**7.195.3.9 remove()**

```
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**

<i>e</i>	The element to remove from the <a href="#">Vector</a> .
----------	---------------------------------------------------------

## 7.195.3.10 removeAt()

```
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

<i>index</i>	The index to remove.
--------------	----------------------

## Returns

The value of the removed element.

## 7.195.3.11 reverse()

```
void reverse ( ) [inline]
```

Reverses the ordering of the elements in the [Vector](#).

## 7.195.3.12 size()

```
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](#).

## 7.196 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

## 7.196.1 Detailed Description

This class represents a homogeneous 3D vector.

See also

quadruple

## 7.196.2 Constructor & Destructor Documentation

### 7.196.2.1 Vector4() [1/2]

```
Vector4 ( ) [inline]
```

Default constructor.

### 7.196.2.2 Vector4() [2/2]

```
Vector4 (
    float x,
    float y,
    float z ) [inline]
```

Constructor.

Parameters

<i>x</i>	The x value.
<i>y</i>	The y value.
<i>z</i>	The z value.

## 7.196.3 Member Function Documentation

### 7.196.3.1 crossProduct()

```
Vector4 crossProduct (
    const Vector4 & operand ) [inline]
```

Cross product.

Parameters

<i>operand</i>	The second operand.
----------------	---------------------

Returns

The result of the operation.

## 7.197 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

### 7.197.1 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

<a href="#">T</a>	The type of <a href="#">Presenter</a> associated with this view.
-------------------	------------------------------------------------------------------

#### See also

[Screen](#)

### 7.197.2 Constructor & Destructor Documentation

#### 7.197.2.1 View()

```
View ( ) [inline]
```

Default constructor.

### 7.197.3 Member Function Documentation

#### 7.197.3.1 bind()

```
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	<i>presenter</i>	The specific <a href="#">Presenter</a> to be associated with the <a href="#">View</a> .
----	------------------	-----------------------------------------------------------------------------------------

## 7.198 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

#### 7.198.1 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](#)

#### 7.198.2 Constructor & Destructor Documentation

##### 7.198.2.1 [Widget](#)()

```
Widget ( ) [inline]
```

Default constructor.

##### 7.198.2.2 [~Widget](#)()

```
~Widget ( ) [inline], [virtual]
```

Destructor.



### 7.198.3 Member Function Documentation

#### 7.198.3.1 getLastChild()

```
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

	<i>x</i>	The point of intersection expressed in coordinates relative to the parent.
	<i>y</i>	The y coordinate.
out	<i>last</i>	Result will be placed here.

Implements [Drawable](#).

#### 7.198.3.2 getType()

```
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](#), [BoxWithBorder](#), [TiledImage](#), [TextAreaWithOneWildcard](#), [ButtonWithLabel](#), [Box](#), [ButtonWithIcon](#), [Image](#), [SnapshotWidget](#), [Button](#), [TouchArea](#), [ToggleButton](#), and [AbstractButton](#).

## 7.199 WildcardTextStyle< T > Class Template Reference

A wildcard text button style.

```
#include <WildcardTextStyle.hpp>
```

### Public Member Functions

- [WildcardTextStyle \( \)](#)  
*Default constructor.*
- virtual [~WildcardTextStyle \( \)](#)

*Destructor.*

- void [setWildcardText](#) ([TypedText](#) t)  
*Sets wildcard text.*
- void [setWildcardTextX](#) (int16\_t x)  
*Sets wildcard text x coordinate.*
- void [setWildcardTextY](#) (int16\_t y)  
*Sets wildcard text y coordinate.*
- void [setWildcardTextXY](#) (int16\_t x, int16\_t y)  
*Sets wildcard text xy.*
- void [setWildcardTextPosition](#) (int16\_t x, int16\_t y, int16\_t width, int16\_t height)  
*Sets text position.*
- void [setWildcardTextRotation](#) ([TextRotation](#) rotation)  
*Sets wildcard text rotation.*
- void [setWildcardTextBuffer](#) (const [Unicode::UnicodeChar](#) \*buffer)  
*Sets wildcard text buffer.*
- void [setWildcardTextColors](#) ([colortype](#) newColorReleased, [colortype](#) newColorPressed)  
*Sets wild card text colors.*

### Protected Member Functions

- virtual void [handlePressedUpdated](#) ()  
*Handles the pressed updated.*
- virtual void [handleAlphaUpdated](#) ()  
*Handles the alpha updated.*

### Protected Attributes

- [TextAreaWithOneWildcard](#) [wildcardText](#)  
*The wildcard text.*
- [colortype](#) [colorReleased](#)  
*The color released.*
- [colortype](#) [colorPressed](#)  
*The color pressed.*

## 7.199.1 Detailed Description

```
template<class T>
class touchgfx::WildcardTextButtonStyle< T >
```

An wildcard text button style. This class is supposed to be used with one of the [ButtonTrigger](#) classes to create a functional button. This class will show a text with a wildcard in one of two colors depending on the state of the button (pressed or released).

The [WildcardTextButtonStyle](#) does not set the size of the enclosing container (normally [AbstractButtonContainer](#)). The size must be set manually.

To get a background behind the text, use [WildcardTextButtonStyle](#) together with e.g. [ImageButtonStyle](#): [WildcardTextButtonStyle](#)<[ImageButtonStyle](#)<[ClickButtonTrigger](#)> > myButton;

The position of the text can be adjusted with [setTextXY](#) (default is centered).

#### Template Parameters

<i>T</i>	Generic type parameter. Typically a <a href="#">AbstractButtonContainer</a> subclass.
----------	---------------------------------------------------------------------------------------

See also

[AbstractButtonContainer](#)

#### Template Parameters

<i>T</i>	Generic type parameter.
----------	-------------------------

### 7.199.2 Member Function Documentation

#### 7.199.2.1 setWildcardText()

```
void setWildcardText (
    TypedText t ) [inline]
```

##### Parameters

<i>t</i>	A <a href="#">TypedText</a> to process.
----------	-----------------------------------------

#### 7.199.2.2 setWildcardTextBuffer()

```
void setWildcardTextBuffer (
    const Unicode::UnicodeChar * buffer ) [inline]
```

##### Parameters

<i>in, out</i>	<i>buffer</i>	If non-null, the buffer.
----------------	---------------	--------------------------

#### 7.199.2.3 setWildcardTextColors()

```
void setWildcardTextColors (
    colortype newColorReleased,
    colortype newColorPressed ) [inline]
```

##### Parameters

<i>newColorReleased</i>	The new color released.
<i>newColorPressed</i>	The new color pressed.

#### 7.199.2.4 setWildcardTextPosition()

```
void setWildcardTextPosition (
    int16_t x,
```

```
    int16_t y,  
    int16_t width,  
    int16_t height ) [inline]
```

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.
<i>width</i>	The width of the text.
<i>height</i>	The height of the text.

**7.199.2.5 setWildcardTextRotation()**

```
void setWildcardTextRotation (  
    TextRotation rotation ) [inline]
```

**Parameters**

<i>rotation</i>	The rotation.
-----------------	---------------

**7.199.2.6 setWildcardTextX()**

```
void setWildcardTextX (  
    int16_t x ) [inline]
```

**Parameters**

<i>x</i>	The x coordinate.
----------	-------------------

**7.199.2.7 setWildcardTextXY()**

```
void setWildcardTextXY (  
    int16_t x,  
    int16_t y ) [inline]
```

**Parameters**

<i>x</i>	The x coordinate.
<i>y</i>	The y coordinate.

**7.199.2.8 setWildcardTextY()**

```
void setWildcardTextY (  
    int16_t y ) [inline]
```

## Parameters

<code>y</code>	The y coordinate.
----------------	-------------------

## 7.200 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](#) zoom↵  
Mode=FIXED\_LEFT\_AND\_TOP, [touchgfx::EasingEquation](#) widthProgressionEquation=&[touchgfx::Easing](#)↵  
[Equations::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 end↵  
Height, uint16\_t duration, [ZoomMode](#) zoomMode=FIXED\_LEFT\_AND\_TOP, [EasingEquation](#) x↵  
ProgressionEquation=&[touchgfx::EasingEquations::linearEaseNone](#), [EasingEquation](#) yProgression↵  
Equation=&[touchgfx::EasingEquations::linearEaseNone](#), [EasingEquation](#) widthProgressionEquation=&[touchgfx](#)↵  
::[EasingEquations::linearEaseNone](#), [EasingEquation](#) heightProgressionEquation=&[touchgfx::Easing](#)↵  
[Equations::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 bool [isZoomAnimationRunning](#) () 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.*
- void [startTimerAndSetParameters](#) (int16\_t endWidth, int16\_t endHeight, uint16\_t duration, [ZoomMode](#) zoomMode, [EasingEquation](#) widthProgressionEquation, [EasingEquation](#) heightProgressionEquation)  
*Starts timer and set parameters.*
- 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

### 7.200.1 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](#)

### 7.200.2 Member Enumeration Documentation

#### 7.200.2.1 States

```
enum States [protected]
```

#### Enumerator

ANIMATE_ZOOM	Zoom animation state.
ANIMATE_ZOOM_AND_MOVE	Zoom and move animation state.
NO_ANIMATION	No animation state.

### 7.200.3 Constructor & Destructor Documentation

#### 7.200.3.1 ZoomAnimationImage()

```
ZoomAnimationImage ( )
```

Creates and initialize the [ZoomAnimationImage](#).

#### 7.200.3.2 ~ZoomAnimationImage()

```
~ZoomAnimationImage ( ) [virtual]
```

Destroys the [ZoomAnimationImage](#).



## 7.200.4 Member Function Documentation

### 7.200.4.1 getAlpha()

```
uint8_t getAlpha ( ) const [virtual]
```

Gets the current alpha value.

#### Returns

The current alpha value.

### 7.200.4.2 getAnimationDelay()

```
uint16_t getAnimationDelay ( ) const [virtual]
```

Gets the current animation delay.

#### Returns

The current animation delay. Expressed in ticks.

### 7.200.4.3 getLargeBitmap()

```
Bitmap getLargeBitmap ( ) const [inline]
```

Gets the large bitmap.

#### Returns

the large bitmap.

### 7.200.4.4 getScalingMode()

```
ScalableImage::ScalingAlgorithm getScalingMode ( ) [virtual]
```

Gets the scaling algorithm of the [ScalableImage](#).

#### Returns

the scaling algorithm used.

### 7.200.4.5 getSmallBitmap()

```
Bitmap getSmallBitmap ( ) const [inline]
```

Gets the small bitmap.

**Returns**

the small bitmap.

**7.200.4.6   getType()**

```
uint16_t getType ( ) const [inline], [virtual]
```

For GUI testing only. Returns type of this drawable.

**Returns**

TYPE\_ZOOMANIMATIONIMAGE.

Reimplemented from [Container](#).

**7.200.4.7   handleTickEvent()**

```
void handleTickEvent ( ) [virtual]
```

The tick handler.

Reimplemented from [Drawable](#).

**7.200.4.8   isRunning()**

```
bool isRunning ( ) const [virtual]
```

Is there currently an animation running.

**Returns**

true if there is an animation running.

**7.200.4.9   isZoomAnimationRunning()**

```
bool isZoomAnimationRunning ( ) const [virtual]
```

Is there currently an animation running.

**Returns**

true if there is an animation running.

**7.200.4.10   setAlpha()**

```
void setAlpha (
    uint8_t alpha ) [virtual]
```

Sets the alpha channel for the image.

## Parameters

<i>alpha</i>	The alpha value. 255 = completely solid.
--------------	------------------------------------------

## 7.200.4.11 setAnimationDelay()

```
void setAnimationDelay (
    uint16_t delay ) [virtual]
```

Sets a delay on animations done by the [ZoomAnimationImage](#). Defaults to 0.

## Parameters

<i>delay</i>	The delay in ticks.
--------------	---------------------

## 7.200.4.12 setAnimationEndedCallback()

```
void setAnimationEndedCallback (
    touchgfx::GenericCallback< const ZoomAnimationImage & > & callback ) [inline]
```

Associates an action to be performed when the animation ends.

## Parameters

<i>callback</i>	The callback to be executed. The callback will be given a reference to the <a href="#">ZoomAnimationImage</a> .
-----------------	-----------------------------------------------------------------------------------------------------------------

## See also

[GenericCallback](#)

## 7.200.4.13 setBitmaps()

```
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

<i>smallBitmap</i>	The image in the smallest resolution.
<i>largeBitmap</i>	The image in the largest resolution.

#### 7.200.4.14 setCurrentState()

```
void setCurrentState (
    States state ) [protected], [virtual]
```

Sets the current animation state.

##### Parameters

<i>state</i>	The new state.
--------------	----------------

#### 7.200.4.15 setDimension()

```
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

<i>width</i>	The new width.
<i>height</i>	The new height.

#### 7.200.4.16 setHeight()

```
void setHeight (
    int16_t height ) [virtual]
```

Sets the height of the image. Chooses the optimal rendering method afterwards. The image is automatically invalidated.

##### Parameters

<i>height</i>	The new height.
---------------	-----------------

Reimplemented from [Drawable](#).

#### 7.200.4.17 setPosition()

```
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

<i>x</i>	The x coordinate of this <a href="#">Drawable</a> .
<i>y</i>	The y coordinate of this <a href="#">Drawable</a> .
<i>width</i>	The width of this <a href="#">Drawable</a> .
<i>height</i>	The height of this <a href="#">Drawable</a> .

Reimplemented from [Drawable](#).

## 7.200.4.18 setScalingMode()

```
void setScalingMode (
    ScalableImage::ScalingAlgorithm mode ) [virtual]
```

Sets the scaling algorithm of the [ScalableImage](#).

## Parameters

<i>mode</i>	The new mode.
-------------	---------------

## 7.200.4.19 setWidth()

```
void setWidth (
    int16_t width ) [virtual]
```

Sets the width of the image. Chooses the optimal rendering method afterwards The image is automatically invalidated.

## Parameters

<i>width</i>	The new width.
--------------	----------------

Reimplemented from [Drawable](#).

## 7.200.4.20 startTimerAndSetParameters()

```
void startTimerAndSetParameters (
    int16_t endWidth,
    int16_t endHeight,
    uint16_t duration,
    ZoomMode zoomMode,
    EasingEquation widthProgressionEquation,
    EasingEquation heightProgressionEquation ) [protected]
```

Starts timer and set parameters. Contains code shared between [startZoomAnimation\(\)](#) and [startZoomAndMoveAnimation\(\)](#). If both delay and duration is zero, the end position and size is applied and the animation is ended.

## Parameters

<i>endWidth</i>	The end width.
-----------------	----------------

## Parameters

<i>endHeight</i>	The end height.
<i>duration</i>	The duration.
<i>zoomMode</i>	The zoom mode.
<i>widthProgressionEquation</i>	The width progression equation.
<i>heightProgressionEquation</i>	The height progression equation.

## 7.200.4.21 startZoomAndMoveAnimation()

```
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

<i>endX</i>	The X position of the image at animation end. Relative to the container or view that holds the <a href="#">ZoomAnimationImage</a> .
<i>endY</i>	The Y position of the image at animation end. Relative to the container or view that holds the <a href="#">ZoomAnimationImage</a> .
<i>endWidth</i>	The width of the image at animation end.
<i>endHeight</i>	The height of the image at animation end.
<i>duration</i>	The duration of the animation measured in ticks.
<i>zoomMode</i>	The zoom mode that will be used during the animation. Default = FIXED_LEFT_AND_TOP.
<i>xProgressionEquation</i>	The equation that describes the development of the X position during the animation. Default = <a href="#">EasingEquations::linearEaseNone</a> .
<i>yProgressionEquation</i>	The equation that describes the development of the Y position during the animation. Default = <a href="#">EasingEquations::linearEaseNone</a> .
<i>widthProgressionEquation</i>	The equation that describes the development of the width during the animation. Default = <a href="#">EasingEquations::linearEaseNone</a> .
<i>heightProgressionEquation</i>	The equation that describes the development of the height during the animation. Default = <a href="#">EasingEquations::linearEaseNone</a> .

## 7.200.4.22 startZoomAnimation()

```
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 endWidth 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

<i>endWidth</i>	The width of the image at animation end.
<i>endHeight</i>	The height of the image at animation end.
<i>duration</i>	The duration of the animation measured in ticks.
<i>zoomMode</i>	The zoom mode that will be used during the animation. Default = FIXED_LEFT_AND_TOP.
<i>widthProgressionEquation</i>	The equation that describes the development of the width during the animation. Default = <a href="#">EasingEquations::linearEaseNone</a> .
<i>heightProgressionEquation</i>	The equation that describes the development of the height during the animation. Default = <a href="#">EasingEquations::linearEaseNone</a> .

## 7.200.4.23 updateRenderingMethod()

```
void updateRenderingMethod ( ) [protected], [virtual]
```

Chooses the optimal rendering of the image given the current width and height.

## 7.200.4.24 updateZoomAnimationDeltaXY()

```
void updateZoomAnimationDeltaXY ( ) [protected], [virtual]
```

Calculates the change in X and Y caused by the zoom animation given the current ZoomMode.

# Index

- ~AbstractClock
  - touchgfx::AbstractClock, [68](#)
- ~AbstractDirectionProgress
  - touchgfx::AbstractDirectionProgress, [71](#)
- ~AbstractPainter
  - touchgfx::AbstractPainter, [72](#)
- ~AbstractPartition
  - touchgfx::AbstractPartition, [87](#)
- ~AbstractProgressIndicator
  - touchgfx::AbstractProgressIndicator, [94](#)
- ~AbstractShape
  - touchgfx::AbstractShape, [100](#)
- ~AnimationTextureMapper
  - touchgfx::AnimationTextureMapper, [124](#)
- ~Box
  - touchgfx::Box, [147](#)
- ~BoxProgress
  - touchgfx::BoxProgress, [150](#)
- ~Button
  - touchgfx::Button, [160](#)
- ~ButtonController
  - touchgfx::ButtonController, [162](#)
- ~Canvas
  - touchgfx::Canvas, [180](#)
- ~CanvasWidget
  - touchgfx::CanvasWidget, [183](#)
- ~CircleProgress
  - touchgfx::CircleProgress, [206](#)
- ~ClickEvent
  - touchgfx::ClickEvent, [214](#)
- ~Container
  - touchgfx::Container, [226](#)
- ~CoverTransition
  - touchgfx::CoverTransition, [231](#)
- ~DMA\_Interface
  - touchgfx::DMA\_Interface, [248](#)
- ~DMA\_Queue
  - touchgfx::DMA\_Queue, [253](#)
- ~DigitalClock
  - touchgfx::DigitalClock, [241](#)
- ~DragEvent
  - touchgfx::DragEvent, [256](#)
- ~Draggable
  - touchgfx::Draggable, [259](#)
- ~Drawable
  - touchgfx::Drawable, [262](#)
- ~DrawableList
  - touchgfx::DrawableList, [275](#)
- ~DrawableListItems
  - touchgfx::DrawableListItems, [283](#)
- ~DrawableListItemsInterface
  - touchgfx::DrawableListItemsInterface, [285](#)
- ~Event
  - touchgfx::Event, [305](#)
- ~FadeAnimator
  - touchgfx::FadeAnimator, [307](#)
- ~FontProvider
  - touchgfx::FontProvider, [319](#)
- ~GenericCallback
  - touchgfx::GenericCallback, [321](#)
  - touchgfx::GenericCallback< T1, T2, void >, [323](#)
  - touchgfx::GenericCallback< T1, void, void >, [325](#)
  - touchgfx::GenericCallback< void >, [326](#)
- ~HAL
  - touchgfx::HAL, [341](#)
- ~I2C
  - touchgfx::I2C, [368](#)
- ~I2CTouchController
  - touchgfx::I2CTouchController, [370](#)
- ~ImageProgress
  - touchgfx::ImageProgress, [380](#)
- ~Keyboard
  - touchgfx::Keyboard, [392](#)
- ~LCD
  - touchgfx::LCD, [400](#)
- ~LineProgress
  - touchgfx::LineProgress, [458](#)
- ~ListLayout
  - touchgfx::ListLayout, [463](#)
- ~MCUInstrumentation
  - touchgfx::MCUInstrumentation, [474](#)
- ~MVPApplication
  - touchgfx::MVPApplication, [485](#)
- ~MVPHeap
  - touchgfx::MVPHeap, [486](#)
- ~ModalWindow
  - touchgfx::ModalWindow, [477](#)
- ~MoveAnimator
  - touchgfx::MoveAnimator, [481](#)
- ~NoTouchController
  - touchgfx::NoTouchController, [489](#)
- ~NoTransition
  - touchgfx::NoTransition, [490](#)
- ~Outline
  - touchgfx::Outline, [494](#)
- ~Presenter
  - touchgfx::Presenter, [538](#)
- ~RadioButton



- touchgfx::RadioButton, 553
- ~RadioButtonGroup
  - touchgfx::RadioButtonGroup, 559
- ~RenderingBuffer
  - touchgfx::RenderingBuffer, 575
- ~ScalableImage
  - touchgfx::ScalableImage, 583
- ~Scanline
  - touchgfx::Scanline, 588
- ~Screen
  - touchgfx::Screen, 591
- ~ScrollBase
  - touchgfx::ScrollBase, 610
- ~ScrollList
  - touchgfx::ScrollList, 624
- ~ScrollWheel
  - touchgfx::ScrollWheel, 628
- ~ScrollWheelBase
  - touchgfx::ScrollWheelBase, 630
- ~ScrollWheelWithSelectionStyle
  - touchgfx::ScrollWheelWithSelectionStyle, 635
- ~ScrollableContainer
  - touchgfx::ScrollableContainer, 599
- ~Shape
  - touchgfx::Shape, 644
- ~SlideTransition
  - touchgfx::SlideTransition, 668
- ~Snapper
  - touchgfx::Snapper, 671
- ~SnapshotWidget
  - touchgfx::SnapshotWidget, 673
- ~TextProgress
  - touchgfx::TextProgress, 700
- ~TextureMapper
  - touchgfx::TextureMapper, 711
- ~TouchController
  - touchgfx::TouchController, 739
- ~Transition
  - touchgfx::Transition, 741
- ~UIEventListener
  - touchgfx::UIEventListener, 750
- ~Widget
  - touchgfx::Widget, 772
- ~ZoomAnimationImage
  - touchgfx::ZoomAnimationImage, 780
- abs
  - touchgfx, 50
- AbstractButton, 63
  - touchgfx::AbstractButton, 64
- AbstractButtonContainer, 65
- AbstractClock, 67
  - touchgfx::AbstractClock, 68
- AbstractDirectionProgress, 70
  - touchgfx::AbstractDirectionProgress, 70
- AbstractPainter, 71
  - touchgfx::AbstractPainter, 72
- AbstractPainterBW, 74
- AbstractPainterGRAY2, 76
- AbstractPainterGRAY4, 78
- AbstractPainterRGB565, 81
- AbstractPainterRGB888, 84
- AbstractPartition, 86
  - touchgfx::AbstractPartition, 88
- AbstractProgressIndicator, 93
  - touchgfx::AbstractProgressIndicator, 94
- AbstractShape, 98
  - touchgfx::AbstractShape, 100
- AbstractShape::ShapePoint< T >, 647
- activate
  - touchgfx::Presenter, 538
- add
  - touchgfx::Container, 226
  - touchgfx::ListLayout, 464
  - touchgfx::ModalWindow, 477
  - touchgfx::RadioButtonGroup, 560
  - touchgfx::Screen, 592
  - touchgfx::ScrollableContainer, 599
  - touchgfx::SlideMenu, 650
  - touchgfx::SwipeContainer, 677
  - touchgfx::Vector, 766
- addCell
  - touchgfx::Scanline, 588
- addCover
  - touchgfx::Cell, 191
- addSpan
  - touchgfx::Scanline, 589
- addToQueue
  - touchgfx::DMA\_Interface, 249
- advance
  - touchgfx::GlyphNode, 332
- afterTransition
  - touchgfx::Screen, 592
- Alignment
  - touchgfx, 47
- allocate
  - touchgfx::AbstractPartition, 88
- allocateAt
  - touchgfx::AbstractPartition, 89
- allowDMATransfers
  - touchgfx::HAL, 341
- allowHorizontalDrag
  - touchgfx::ScrollBase, 611
- allowVerticalDrag
  - touchgfx::ScrollBase, 611
- AnalogClock, 108
- angle
  - touchgfx::CWRUtil, 234, 235
- animateToItem
  - touchgfx::ScrollBase, 611
- animateToPosition
  - touchgfx::ScrollBase, 612
  - touchgfx::ScrollWheelBase, 630
- animateToState
  - touchgfx::SlideMenu, 650
- AnimatedImage, 115
  - touchgfx::AnimatedImage, 117

- AnimatedImageButtonStyle< T >, 120
- animationEnabled
  - touchgfx::AnalogClock, 110
- animationEndedHandler
  - touchgfx::SlideMenu, 650
- AnimationParameter
  - touchgfx::AnimationTextureMapper, 124
- AnimationState
  - touchgfx::ScrollBase, 610
- AnimationTextureMapper, 122
  - touchgfx::AnimationTextureMapper, 124
- AnimationTextureMapper::AnimationSetting, 122
- appSwitchScreen
  - touchgfx::Application, 128
- Application, 126
  - touchgfx::Application, 128
- applyTransformation
  - touchgfx::TextureMapper, 711
- arcsine
  - touchgfx::CWRUtil, 235
- area
  - touchgfx::Rect, 568
- at
  - touchgfx::AbstractPartition, 89, 90
- atoi
  - touchgfx::Unicode, 753
- attach
  - touchgfx::RenderingBuffer, 575
- backEaseIn
  - touchgfx::EasingEquations, 288
- backEaseInOut
  - touchgfx::EasingEquations, 288
- backEaseOut
  - touchgfx::EasingEquations, 289
- backPorchExited
  - touchgfx::HAL, 342
- beginFrame
  - touchgfx::HAL, 342
- bind
  - touchgfx::View, 771
- bindTransition
  - touchgfx::Screen, 592
- bitDepth
  - touchgfx::LCD16bpp, 412
  - touchgfx::LCD1bpp, 419
  - touchgfx::LCD24bpp, 426
  - touchgfx::LCD2bpp, 432
  - touchgfx::LCD4bpp, 439
  - touchgfx::LCD, 400
- Bitmap, 134
  - touchgfx::Bitmap, 136
- Bitmap::BitmapData, 144
- Bitmap::CacheTableEntry, 171
- Bitmap::DynamicBitmapData, 286
- BitmapFormat
  - touchgfx::Bitmap, 135
- blitCopy
  - touchgfx::HAL, 342
  - touchgfx::LCD16bpp, 412, 413
  - touchgfx::LCD1bpp, 419, 420
  - touchgfx::LCD24bpp, 426
  - touchgfx::LCD2bpp, 432, 433
  - touchgfx::LCD4bpp, 439
  - touchgfx::LCD, 400, 401
- blitCopyARGB8888
  - touchgfx::HAL, 342
  - touchgfx::LCD16bpp, 414
  - touchgfx::LCD24bpp, 427
- blitCopyAlphaPerPixel
  - touchgfx::LCD16bpp, 413
  - touchgfx::LCD2bpp, 433
  - touchgfx::LCD4bpp, 440
- blitCopyGlyph
  - touchgfx::HAL, 343
- blitCopyRLE
  - touchgfx::LCD1bpp, 420
- blitFill
  - touchgfx::HAL, 344
- BlitOp, 144
- BlitOperations
  - touchgfx, 47
- blitSetTransparencyKey
  - touchgfx::HALSDL2, 361
  - touchgfx::HAL, 344
- blockCopy
  - touchgfx::HALSDL2, 361
  - touchgfx::HAL, 344
- bottom
  - touchgfx::Rect, 569
- bounceEaseIn
  - touchgfx::EasingEquations, 289
- bounceEaseInOut
  - touchgfx::EasingEquations, 290
- bounceEaseOut
  - touchgfx::EasingEquations, 290
- Box, 145
  - touchgfx::Box, 146
- BoxProgress, 149
  - touchgfx::BoxProgress, 150
- BoxWithBorder, 152
  - touchgfx::BoxWithBorder, 153
- BoxWithBorderButtonStyle< T >, 156
- Button, 159
  - touchgfx::Button, 159
- ButtonController, 162
- ButtonWithIcon, 163
- ButtonWithLabel, 167
  - touchgfx::ButtonWithLabel, 168
- Buttons, 163
- bw
  - touchgfx::PainterBW, 496
- CWRUtil, 233
- CWRUtil::Q10, 539
- CWRUtil::Q15, 541
- CWRUtil::Q5, 543
- cache

- touchgfx::Bitmap, 136
- cacheAll
  - touchgfx::Bitmap, 137
- cacheDrawOperations
  - touchgfx::Application, 129
- cacheGetAddress
  - touchgfx::Bitmap, 137
- cachelsCached
  - touchgfx::Bitmap, 137
- cacheRemoveBitmap
  - touchgfx::Bitmap, 138
- cacheReplaceBitmap
  - touchgfx::Bitmap, 138
- cacheTextString
  - touchgfx::HAL, 345
- calculateAlpha
  - touchgfx::Rasterizer, 565
- calculateTextHeight
  - touchgfx::TextAreaWithWildcardBase, 695
- Callback
  - touchgfx::Callback, 172
  - touchgfx::Callback< dest\_type, T1, T2, void >, 174
  - touchgfx::Callback< dest\_type, T1, void, void >, 176
  - touchgfx::Callback< dest\_type, void, void, void >, 177, 178
- Callback< dest\_type, T1, T2, T3 >, 171
- Callback< dest\_type, T1, T2, void >, 173
- Callback< dest\_type, T1, void, void >, 175
- Callback< dest\_type, void, void, void >, 177
- cancelAnimationTextureMapperAnimation
  - touchgfx::AnimationTextureMapper, 124
- cancelMoveAnimation
  - touchgfx::MoveAnimator, 482
- Canvas, 179
  - touchgfx::Canvas, 179
- CanvasWidget, 182
  - touchgfx::CanvasWidget, 182
- CanvasWidgetRenderer, 186
- capacity
  - touchgfx::AbstractPartition, 90
  - touchgfx::Partition, 530
- ceil28\_4
  - touchgfx, 51
- Cell, 190
- childGeometryChanged
  - touchgfx::Drawable, 262
  - touchgfx::ScrollableContainer, 600
- circEaseIn
  - touchgfx::EasingEquations, 291
- circEaseInOut
  - touchgfx::EasingEquations, 291
- circEaseOut
  - touchgfx::EasingEquations, 291
- Circle, 193
  - touchgfx::Circle, 194
- CircleProgress, 205
  - touchgfx::CircleProgress, 206
- clear
  - touchgfx::AbstractPartition, 90
  - touchgfx::GPIO, 334
  - touchgfx::Vector, 766
- clearAllTimerWidgets
  - touchgfx::Application, 129
- clearCache
  - touchgfx::Bitmap, 139
- clearFadeAnimationEndedAction
  - touchgfx::FadeAnimator, 308
- clearMoveAnimationEndedAction
  - touchgfx::MoveAnimator, 482
- ClickButtonTrigger, 211
- ClickEvent, 212
  - touchgfx::ClickEvent, 214
- ClickEventType
  - touchgfx::ClickEvent, 212
- ClickListener
  - touchgfx::ClickListener, 217
- ClickListener< T >, 216
- clz
  - touchgfx, 51
- Color, 218
- colortype, 220
  - touchgfx::colortype, 220
- concatenateXRotation
  - touchgfx::Matrix4x4, 469
- concatenateXScale
  - touchgfx::Matrix4x4, 469
- concatenateXTranslation
  - touchgfx::Matrix4x4, 469
- concatenateYRotation
  - touchgfx::Matrix4x4, 470
- concatenateYScale
  - touchgfx::Matrix4x4, 470
- concatenateYTranslation
  - touchgfx::Matrix4x4, 470
- concatenateZRotation
  - touchgfx::Matrix4x4, 471
- concatenateZScale
  - touchgfx::Matrix4x4, 471
- concatenateZTranslation
  - touchgfx::Matrix4x4, 471
- configureInterrupts
  - touchgfx::HALSDL2, 362
  - touchgfx::HAL, 345
- configureLCDInterrupt
  - touchgfx::HALSDL2, 362
- ConstFont, 221
  - touchgfx::ConstFont, 222
- Container, 224
  - touchgfx::Container, 226
- contains
  - touchgfx::Container, 226
  - touchgfx::Vector, 767
- convertHandValueToAngle
  - touchgfx::AnalogClock, 110
- copyFBRegionToMemory

- touchgfx::HAL, 345, 346
- copyFrameBufferRegionToMemory
  - touchgfx::LCD16bpp, 414
  - touchgfx::LCD1bpp, 421
  - touchgfx::LCD24bpp, 427
  - touchgfx::LCD2bpp, 434
  - touchgfx::LCD4bpp, 440
  - touchgfx::LCD, 401
- copyScreenshotToClipboard
  - touchgfx::HALSDL2, 362
- cosine
  - touchgfx::CWRUtil, 236
- CoverTransition
  - touchgfx::CoverTransition, 231
- CoverTransition< templateDirection >, 230
- CoverTransition< templateDirection >::FullSolidRect, 319
- crossProduct
  - touchgfx::Vector4, 770
- cubicEaseln
  - touchgfx::EasingEquations, 292
- cubicEaselnOut
  - touchgfx::EasingEquations, 292
- cubicEaseOut
  - touchgfx::EasingEquations, 293
- DMA\_Interface, 247
  - touchgfx::DMA\_Interface, 248
- DMA\_Queue, 252
  - touchgfx::DMA\_Queue, 253
- deactivate
  - touchgfx::Presenter, 539
- dec
  - touchgfx::AbstractPartition, 91
- DigitalClock, 239
  - touchgfx::DigitalClock, 241
- Direction
  - touchgfx, 48
- DirectionType
  - touchgfx::AbstractDirectionProgress, 70
- disableAlpha
  - touchgfx::DMA\_Interface, 249
- disableInterrupts
  - touchgfx::HALSDL2, 362
  - touchgfx::HAL, 346
- displayLeadingZeroForHourIndicator
  - touchgfx::DigitalClock, 241
- DisplayMode
  - touchgfx::DigitalClock, 241
- DisplayOrientation
  - touchgfx, 48
- DisplayRotation
  - touchgfx, 48
- DisplayTransformation, 244
- dist\_sqr
  - touchgfx::Point, 534
- doSampleTouch
  - touchgfx::HALSDL2, 362
- doScroll
  - touchgfx::ScrollableContainer, 600
- DragEvent, 255
  - touchgfx::DragEvent, 256
- Draggable
  - touchgfx::Draggable, 258
- Draggable< T >, 258
- draw
  - touchgfx::Application, 129
  - touchgfx::Box, 147
  - touchgfx::BoxWithBorder, 154
  - touchgfx::Button, 160
  - touchgfx::ButtonWithIcon, 164
  - touchgfx::ButtonWithLabel, 168
  - touchgfx::CanvasWidget, 183
  - touchgfx::Container, 226
  - touchgfx::CoverTransition::FullSolidRect, 320
  - touchgfx::Drawable, 263
  - touchgfx::Image, 375
  - touchgfx::JSMOCHelper, 387
  - touchgfx::Keyboard, 392
  - touchgfx::PixelDataWidget, 532
  - touchgfx::PreRenderable, 537
  - touchgfx::RadioButton, 553
  - touchgfx::ScalableImage, 584
  - touchgfx::Screen, 592, 593
  - touchgfx::SnapshotWidget, 673
  - touchgfx::TextArea, 683
  - touchgfx::TextAreaWithOneWildcard, 690
  - touchgfx::TextAreaWithTwoWildcards, 693
  - touchgfx::TextureMapper, 711
  - touchgfx::TiledImage, 725
  - touchgfx::TouchArea, 736
- drawBorder
  - touchgfx::LCD, 402
- drawCanvasWidget
  - touchgfx::AbstractShape, 100
  - touchgfx::CanvasWidget, 183
  - touchgfx::Circle, 194
  - touchgfx::Line, 448
- drawGlyph
  - touchgfx::LCD16bpp, 415
  - touchgfx::LCD1bpp, 421
  - touchgfx::LCD24bpp, 427
  - touchgfx::LCD2bpp, 434
  - touchgfx::LCD4bpp, 441
  - touchgfx::LCD, 402, 403
- drawHorizontalLine
  - touchgfx::LCD, 404
- drawPartialBitmap
  - touchgfx::LCD16bpp, 415
  - touchgfx::LCD1bpp, 422
  - touchgfx::LCD24bpp, 428
  - touchgfx::LCD2bpp, 435
  - touchgfx::LCD4bpp, 441
  - touchgfx::LCD, 404
- drawRect
  - touchgfx::LCD, 405
- drawString

- touchgfx::LCD, 405
- drawStringLTR
  - touchgfx::LCD, 405
- drawStringRTL
  - touchgfx::LCD, 406
- drawTextureMapScanLine
  - touchgfx::LCD16bpp, 416
  - touchgfx::LCD1bpp, 422
  - touchgfx::LCD24bpp, 429
  - touchgfx::LCD2bpp, 435
  - touchgfx::LCD4bpp, 442
  - touchgfx::LCD, 407
- drawTextureMapTriangle
  - touchgfx::LCD, 407
- drawTriangle
  - touchgfx::ScalableImage, 584
  - touchgfx::TextureMapper, 711
- drawVerticalLine
  - touchgfx::LCD, 408
- Drawable, 259
  - touchgfx::Drawable, 262
- DrawableList, 273
  - touchgfx::DrawableList, 275
- DrawableListItems
  - touchgfx::DrawableListItems, 283
- DrawableListItems< TYPE, SIZE >, 282
- DrawableListItemsInterface, 284
- DrawableType
  - touchgfx::Drawable, 262
- DrawingSurface, 285
- dynamicBitmapCreate
  - touchgfx::Bitmap, 139
- dynamicBitmapDelete
  - touchgfx::Bitmap, 139
- dynamicBitmapGetAddress
  - touchgfx::Bitmap, 140
- dynamicBitmapSetSolidRect
  - touchgfx::Bitmap, 140
- EasingEquation
  - touchgfx, 47
- EasingEquations, 286
- Edge, 303
  - touchgfx::Edge, 304
- elasticEaseIn
  - touchgfx::EasingEquations, 293
- elasticEaseInOut
  - touchgfx::EasingEquations, 294
- elasticEaseOut
  - touchgfx::EasingEquations, 294
- element
  - touchgfx::AbstractPartition, 91
  - touchgfx::Partition, 530
- element\_size
  - touchgfx::AbstractPartition, 91
  - touchgfx::Partition, 531
- enableAlpha
  - touchgfx::DMA\_Interface, 249
- enableCopyWithTransparentPixels
  - touchgfx::DMA\_Interface, 249
- enableHorizontalScroll
  - touchgfx::ScrollableContainer, 600
- enableInterrupts
  - touchgfx::HALSDL2, 363
  - touchgfx::HAL, 346
- enableLCDControllerInterrupt
  - touchgfx::HALSDL2, 363
  - touchgfx::HAL, 346
- enableMCULoadCalculation
  - touchgfx::HAL, 347
- enableVerticalScroll
  - touchgfx::ScrollableContainer, 601
- endFrame
  - touchgfx::HAL, 347
- evaluatePendingScreenTransition
  - touchgfx::MVPApplication, 485
- Event, 305
- EventType
  - touchgfx::Event, 305
- execute
  - touchgfx::Callback, 172
  - touchgfx::Callback< dest\_type, T1, T2, void >, 174
  - touchgfx::Callback< dest\_type, T1, void, void >, 176
  - touchgfx::Callback< dest\_type, void, void, void >, 178
  - touchgfx::DMA\_Interface, 249
  - touchgfx::GenericCallback, 321
  - touchgfx::GenericCallback< T1, T2, void >, 323
  - touchgfx::GenericCallback< T1, void, void >, 325
  - touchgfx::GenericCallback< void >, 326
- executeCompleted
  - touchgfx::DMA\_Interface, 250
- expandToFit
  - touchgfx::Rect, 569
- expoEaseIn
  - touchgfx::EasingEquations, 294
- expoEaseInOut
  - touchgfx::EasingEquations, 295
- expoEaseOut
  - touchgfx::EasingEquations, 295
- FadeAnimator
  - touchgfx::FadeAnimator, 307
- FadeAnimator< T >, 306
- fillMemory
  - touchgfx::LCD1bpp, 423
- fillRect
  - touchgfx::LCD16bpp, 416
  - touchgfx::LCD1bpp, 423
  - touchgfx::LCD24bpp, 429
  - touchgfx::LCD2bpp, 436
  - touchgfx::LCD4bpp, 442
  - touchgfx::LCD, 408
- FillingRule
  - touchgfx::Rasterizer, 564
- finalizeTransition
  - touchgfx, 51

- find
  - touchgfx::AbstractPartition, 92
  - touchgfx::ConstFont, 223
- first
  - touchgfx::DMA\_Queue, 253
  - touchgfx::LockFreeDMA\_Queue, 467
- fixed28\_4Mul
  - touchgfx, 52
- fixed28\_4ToFloat
  - touchgfx, 52
- floatToFixed16\_16
  - touchgfx, 52
- floatToFixed28\_4
  - touchgfx, 52
- floorDivMod
  - touchgfx, 53
- flush
  - touchgfx::DMA\_Interface, 250
  - touchgfx::NoDMA, 487
- flushDMA
  - touchgfx::HAL, 347
- flushFrameBuffer
  - touchgfx::HALSDL2, 363
  - touchgfx::HAL, 347
- Font, 310
  - touchgfx::Font, 311
- FontManager, 318
- FontProvider, 319
- forEachChild
  - touchgfx::Container, 227
- forceReportAsSolid
  - touchgfx::Box, 147
- forceState
  - touchgfx::ToggleButton, 732
  - touchgfx::ToggleButtonTrigger, 734
- FrameBuffer
  - touchgfx, 49
- FrameRefreshStrategy
  - touchgfx::HAL, 341
- fromUTF8
  - touchgfx::Unicode, 753
- frontPorchEntered
  - touchgfx::HAL, 348
- GPIO\_ID
  - touchgfx::GPIO, 333
- GPIO, 333
- gcd
  - touchgfx, 53
- GenericCallback< T1, T2, T3 >, 320
- GenericCallback< T1, T2, void >, 322
- GenericCallback< T1, void, void >, 324
- GenericCallback< void >, 325
- GestureEvent, 327
  - touchgfx::GestureEvent, 328
- GestureType
  - touchgfx::GestureEvent, 327
- Gestures, 329
  - touchgfx::Gestures, 330
- get
  - touchgfx::GPIO, 334
  - touchgfx::LED, 444
- getAbsoluteRect
  - touchgfx::Drawable, 263
- getAlignment
  - touchgfx::TypedText, 747
- getAllocationCount
  - touchgfx::AbstractPartition, 92
- getAllowed
  - touchgfx::DMA\_Interface, 250
- getAlpha
  - touchgfx::AbstractButtonContainer, 66
  - touchgfx::Box, 147
  - touchgfx::BoxProgress, 150
  - touchgfx::BoxWithBorder, 154
  - touchgfx::Button, 160
  - touchgfx::CanvasWidget, 184
  - touchgfx::CircleProgress, 206
  - touchgfx::DigitalClock, 241
  - touchgfx::Image, 376
  - touchgfx::ImageProgress, 380
  - touchgfx::LineProgress, 458
  - touchgfx::PainterGRAY2, 502
  - touchgfx::PainterGRAY2Bitmap, 505
  - touchgfx::PainterGRAY4, 508
  - touchgfx::PainterGRAY4Bitmap, 511
  - touchgfx::PainterRGB565, 515
  - touchgfx::PainterRGB565Bitmap, 518
  - touchgfx::PainterRGB888, 521
  - touchgfx::PainterRGB888Bitmap, 525
  - touchgfx::PixelDataWidget, 532
  - touchgfx::RadioButton, 554
  - touchgfx::ScalableImage, 584
  - touchgfx::SnapshotWidget, 674
  - touchgfx::TextArea, 683
  - touchgfx::TextProgress, 700
  - touchgfx::TextureMapper, 712
  - touchgfx::ZoomAnimationImage, 781
- getAlphaData
  - touchgfx::Bitmap, 140
- getAnchorAtZero
  - touchgfx::ImageProgress, 380
- getAngle
  - touchgfx::AbstractShape, 101
- getAnimationDelay
  - touchgfx::ZoomAnimationImage, 781
- getAnimationDuration
  - touchgfx::AnalogClock, 110
  - touchgfx::SlideMenu, 651
- getAnimationEasingEquation
  - touchgfx::SlideMenu, 651
- getAnimationStep
  - touchgfx::AnimationTextureMapper, 124
- getAnimationSteps
  - touchgfx::ScrollBase, 612
- getAnimationStorage
  - touchgfx::HAL, 348



- getArc
  - touchgfx::Circle, [195](#)
- getArcEnd
  - touchgfx::Circle, [195](#), [196](#)
- getArcStart
  - touchgfx::Circle, [196](#)
- getArgv
  - touchgfx::HALSDL2, [363](#)
- getBackgroundHeight
  - touchgfx::ModalWindow, [477](#)
- getBackgroundWidth
  - touchgfx::ModalWindow, [477](#)
- getBackgroundX
  - touchgfx::SlideMenu, [651](#)
- getBackgroundY
  - touchgfx::SlideMenu, [651](#)
- getBitmap
  - touchgfx::Image, [376](#)
  - touchgfx::ImageProgress, [381](#)
  - touchgfx::ScalableImage, [585](#)
  - touchgfx::TextureMapper, [712](#)
- getBitmapPositionX
  - touchgfx::TextureMapper, [712](#)
- getBitmapPositionY
  - touchgfx::TextureMapper, [712](#)
- getBitsPerPixel
  - touchgfx::Font, [312](#)
- getBlitCaps
  - touchgfx::DMA\_Interface, [250](#)
  - touchgfx::HAL, [348](#)
  - touchgfx::NoDMA, [487](#)
- getBlueColor
  - touchgfx::Color, [218](#)
- getBorderColor
  - touchgfx::BoxWithBorder, [154](#)
- getBorderSize
  - touchgfx::BoxWithBorder, [154](#)
- getBoundingRect
  - touchgfx::TextureMapper, [713](#)
- getBuffer
  - touchgfx::Keyboard, [392](#)
- getBufferPosition
  - touchgfx::Keyboard, [392](#)
- getButtonController
  - touchgfx::HAL, [348](#)
- getCCConsumed
  - touchgfx::MCUInstrumentation, [474](#)
- getCPUCycles
  - touchgfx::HAL, [348](#)
  - touchgfx::MCUInstrumentation, [474](#)
- getCachedAbsX
  - touchgfx::Drawable, [263](#)
  - touchgfx::JSMOCHelper, [387](#)
- getCachedAbsY
  - touchgfx::Drawable, [263](#)
  - touchgfx::JSMOCHelper, [387](#)
- getCachedVisibleRect
  - touchgfx::Drawable, [264](#)
  - touchgfx::JSMOCHelper, [388](#)
- getCacheX
  - touchgfx::AbstractShape, [101](#)
  - touchgfx::Shape, [644](#)
- getCacheY
  - touchgfx::AbstractShape, [101](#)
  - touchgfx::Shape, [645](#)
- getCallbackAreaForCoordinates
  - touchgfx::Keyboard, [393](#)
- getCameraDistance
  - touchgfx::TextureMapper, [713](#)
- getCameraX
  - touchgfx::TextureMapper, [713](#)
- getCameraY
  - touchgfx::TextureMapper, [713](#)
- getCapPrecision
  - touchgfx::Circle, [197](#)
  - touchgfx::CircleProgress, [206](#)
- getCells
  - touchgfx::Outline, [494](#)
- getCenter
  - touchgfx::Circle, [197](#)
  - touchgfx::CircleProgress, [207](#)
- getCharForKey
  - touchgfx::Keyboard, [393](#)
- getCharWidth
  - touchgfx::Font, [312](#)
- getCircular
  - touchgfx::DrawableList, [275](#)
  - touchgfx::ScrollBase, [612](#)
- getClientFrameBuffer
  - touchgfx::HAL, [348](#)
- getCollapsedXCoordinate
  - touchgfx::SlideMenu, [651](#)
- getCollapsedYCoordinate
  - touchgfx::SlideMenu, [652](#)
- getColor
  - touchgfx::Box, [148](#)
  - touchgfx::BoxProgress, [150](#)
  - touchgfx::BoxWithBorder, [154](#)
  - touchgfx::PainterBW, [497](#)
  - touchgfx::PainterGRAY2, [502](#)
  - touchgfx::PainterGRAY4, [508](#)
  - touchgfx::PainterRGB565, [515](#)
  - touchgfx::PainterRGB888, [522](#)
  - touchgfx::TextArea, [684](#)
  - touchgfx::TextProgress, [700](#)
- getColor32
  - touchgfx::colortype, [221](#)
- getColorFrom24BitRGB
  - touchgfx::Color, [218](#)
- getContainedArea
  - touchgfx::Container, [227](#)
  - touchgfx::ScrollableContainer, [601](#)
- getCornerX
  - touchgfx::AbstractShape, [101](#)
  - touchgfx::Shape, [645](#)
- getCornerY

- touchgfx::AbstractShape, 102
- touchgfx::Shape, 645
- getCovers
  - touchgfx::Scanline::iterator, 386
- getCurrentHour
  - touchgfx::AbstractClock, 68
- getCurrentMinute
  - touchgfx::AbstractClock, 68
- getCurrentScreen
  - touchgfx::Application, 130
- getCurrentSecond
  - touchgfx::AbstractClock, 68
- getCurrentlyDisplayedBitmap
  - touchgfx::Button, 160
  - touchgfx::ImageButtonStyle, 378
  - touchgfx::RadioButton, 554
- getCurrentlyDisplayedIcon
  - touchgfx::ButtonWithIcon, 165
  - touchgfx::IconButtonStyle, 373
- getData
  - touchgfx::Bitmap, 141
- getDelay
  - touchgfx::RepeatButton, 578
  - touchgfx::RepeatButtonTrigger, 581
- getDeltaX
  - touchgfx::DragEvent, 256
- getDeltaY
  - touchgfx::DragEvent, 256
- getDeselectionEnabled
  - touchgfx::RadioButton, 554
  - touchgfx::RadioButtongroup, 560
- getDirection
  - touchgfx::AbstractDirectionProgress, 71
  - touchgfx::ListLayout, 464
- getDisplayHeight
  - touchgfx::HAL, 349
- getDisplayMode
  - touchgfx::DigitalClock, 242
- getDisplayOrientation
  - touchgfx::HAL, 349
- getDisplayWidth
  - touchgfx::HAL, 349
- getDragAcceleration
  - touchgfx::ScrollBase, 612
- getDrawable
  - touchgfx::DrawableListItems, 283
  - touchgfx::DrawableListItemsInterface, 285
- getDrawableIndex
  - touchgfx::DrawableList, 275
- getDrawableIndices
  - touchgfx::DrawableList, 276
- getDrawableMargin
  - touchgfx::DrawableList, 276
  - touchgfx::ScrollBase, 613
- getDrawableSize
  - touchgfx::DrawableList, 276
  - touchgfx::ScrollBase, 613
- getElapsedUS
  - touchgfx::MCUInstrumentation, 474
- getElement
  - touchgfx::Matrix4x4, 472
  - touchgfx::Quadruple, 549
- getEllipsisChar
  - touchgfx::Font, 312
- getEnd
  - touchgfx::Line, 448
  - touchgfx::LineProgress, 459
- getEndAngle
  - touchgfx::CircleProgress, 207
- getEventType
  - touchgfx::ClickEvent, 214
  - touchgfx::DragEvent, 257
  - touchgfx::Event, 306
  - touchgfx::GestureEvent, 328
- getExpandDirection
  - touchgfx::SlideMenu, 652
- getExpandedStateTimeout
  - touchgfx::SlideMenu, 652
- getExpandedStateTimer
  - touchgfx::SlideMenu, 652
- getExpandedXCoordinate
  - touchgfx::SlideMenu, 652
- getExpandedYCoordinate
  - touchgfx::SlideMenu, 652
- getFadeAnimationDelay
  - touchgfx::FadeAnimator, 308
- getFallbackChar
  - touchgfx::Font, 313
- getFingerSize
  - touchgfx::HAL, 349
- getFirstChild
  - touchgfx::Container, 227
- getFont
  - touchgfx::FontManager, 318
  - touchgfx::FontProvider, 319
  - touchgfx::TypedText, 748
- getFontHeight
  - touchgfx::Font, 313
- getForce
  - touchgfx::ClickEvent, 214
- getFormat
  - touchgfx::Bitmap, 141
- getFrameRefreshStrategy
  - touchgfx::HAL, 349
- getGlyph
  - touchgfx::ConstFont, 223
  - touchgfx::Font, 313
- getGreenColor
  - touchgfx::Color, 219
- getHeight
  - touchgfx::Bitmap, 141
  - touchgfx::Drawable, 264
  - touchgfx::JSMOCHelper, 388
  - touchgfx::RenderingBuffer, 576
- getHiddenPixelsWhenExpanded
  - touchgfx::SlideMenu, 653



- getHorizontal
  - touchgfx::DrawableList, 277
  - touchgfx::ScrollBase, 613
- getHourHandMinuteCorrection
  - touchgfx::AnalogClock, 111
- getIconX
  - touchgfx::ButtonWithIcon, 165
  - touchgfx::IconButtonStyle, 373
- getIconY
  - touchgfx::ButtonWithIcon, 165
  - touchgfx::IconButtonStyle, 373
- getId
  - touchgfx::Bitmap, 141
  - touchgfx::TypedText, 748
- getIndentation
  - touchgfx::TextArea, 684
- getIndicatorMax
  - touchgfx::Slider, 660
- getIndicatorMin
  - touchgfx::Slider, 660
- getIndicatorPositionRangeSize
  - touchgfx::Slider, 660
- getIndicatorRadius
  - touchgfx::Slider, 660
- getInstance
  - touchgfx::Application, 130
  - touchgfx::HAL, 350
- getInterval
  - touchgfx::RepeatButton, 579
  - touchgfx::RepeatButtonTrigger, 581
- getItem
  - touchgfx::ScrollList, 624
- getItemIndex
  - touchgfx::DrawableList, 277
- getItemSize
  - touchgfx::DrawableList, 277
- getKerning
  - touchgfx::ConstFont, 223
  - touchgfx::Font, 314
  - touchgfx::InternalFlashFont, 384
- getKeyForCoordinates
  - touchgfx::Keyboard, 393
- getKeyMappingList
  - touchgfx::Keyboard, 394
- getLCDRefreshCount
  - touchgfx::HAL, 350
- getLabelRotation
  - touchgfx::ButtonWithLabel, 168
- getLabelText
  - touchgfx::ButtonWithLabel, 169
- getLanguage
  - touchgfx::Texts, 706
- getLargeBitmap
  - touchgfx::ZoomAnimationImage, 781
- getLastChild
  - touchgfx::Container, 227
  - touchgfx::Drawable, 264
  - touchgfx::ScrollableContainer, 601
  - touchgfx::Widget, 773
- getLayout
  - touchgfx::Keyboard, 394
- getLineEndingStyle
  - touchgfx::Line, 448
  - touchgfx::LineProgress, 459
- getLineWidth
  - touchgfx::Circle, 197
  - touchgfx::CircleProgress, 207
  - touchgfx::Line, 449
  - touchgfx::LineProgress, 459
- getLinespacing
  - touchgfx::TextArea, 684
- getMCULoadPct
  - touchgfx::HAL, 350
- getMaxPixelsLeft
  - touchgfx::Font, 314
- getMaxPixelsRight
  - touchgfx::Font, 314
- getMaxSwipeItems
  - touchgfx::ScrollBase, 613
- getMaxTextHeight
  - touchgfx::Font, 315
- getMaxValue
  - touchgfx::Slider, 660
- getMinValue
  - touchgfx::Slider, 661
- getMinimalRect
  - touchgfx::AbstractShape, 102
  - touchgfx::CanvasWidget, 184
  - touchgfx::Circle, 198
  - touchgfx::Line, 449
- getMinimumTextHeight
  - touchgfx::Font, 315
- getMinuteHandSecondCorrection
  - touchgfx::AnalogClock, 111
- getMissingBufferSize
  - touchgfx::CanvasWidgetRenderer, 187
- getMoveAnimationDelay
  - touchgfx::MoveAnimator, 482
- getNearestAlignedOffset
  - touchgfx::ScrollBase, 614
  - touchgfx::ScrollList, 624
- getNewX
  - touchgfx::DragEvent, 257
- getNewY
  - touchgfx::DragEvent, 257
- getNextChar
  - touchgfx::TextProvider, 704
- getNextLigature
  - touchgfx::TextProvider, 704
- getNextSibling
  - touchgfx::Drawable, 265
- getNormalizedOffset
  - touchgfx::ScrollBase, 614
- getNumCells
  - touchgfx::Outline, 494
- getNumLines

- touchgfx::LCD, 409
- getNumPix
  - touchgfx::Scanline::iterator, 386
- getNumPoints
  - touchgfx::AbstractShape, 102
  - touchgfx::Shape, 646
- getNumSpans
  - touchgfx::Scanline, 589
- getNumberOfDecimals
  - touchgfx::TextProgress, 701
- getNumberOfDrawables
  - touchgfx::DrawableList, 278
  - touchgfx::DrawableListItems, 284
  - touchgfx::DrawableListItemsInterface, 285
- getNumberOfItems
  - touchgfx::DrawableList, 278
  - touchgfx::ScrollBase, 614
- getNumberOfLines
  - touchgfx::Font, 315
- getNumberOfPages
  - touchgfx::SwipeContainer, 678
- getNumberOfRegisteredTimerWidgets
  - touchgfx::Application, 130
- getOffset
  - touchgfx::DrawableList, 278
  - touchgfx::ScrollBase, 615
  - touchgfx::TiledImage, 725
- getOldX
  - touchgfx::DragEvent, 257
- getOldY
  - touchgfx::DragEvent, 257
- getOrigin
  - touchgfx::AbstractShape, 103
- getOrigoX
  - touchgfx::TextureMapper, 713
- getOrigoY
  - touchgfx::TextureMapper, 714
- getOrigoZ
  - touchgfx::TextureMapper, 714
- getOutlineBuffer
  - touchgfx::CanvasWidgetRenderer, 187
- getOutlineBufferSize
  - touchgfx::CanvasWidgetRenderer, 187
- getPaddingAfter
  - touchgfx::ScrollList, 624
- getPaddingBefore
  - touchgfx::ScrollList, 625
- getPainter
  - touchgfx::CanvasWidget, 184
- getParent
  - touchgfx::Drawable, 265
- getPixelData
  - touchgfx::ConstFont, 224
  - touchgfx::InternalFlashFont, 384
- getPositionForItem
  - touchgfx::ScrollBase, 615
  - touchgfx::ScrollList, 625
  - touchgfx::ScrollWheelBase, 630
- getPrecision
  - touchgfx::Circle, 199
- getPressed
  - touchgfx::AbstractButtonContainer, 66
- getPressedState
  - touchgfx::AbstractButton, 64
- getProgress
  - touchgfx::AbstractProgressIndicator, 94
- getProgressIndicatorHeight
  - touchgfx::AbstractProgressIndicator, 95
- getProgressIndicatorWidth
  - touchgfx::AbstractProgressIndicator, 95
- getProgressIndicatorX
  - touchgfx::AbstractProgressIndicator, 95
- getProgressIndicatorY
  - touchgfx::AbstractProgressIndicator, 95
- getRadioButton
  - touchgfx::RadioButtonGroup, 560
- getRadius
  - touchgfx::Circle, 199
  - touchgfx::CircleProgress, 207
- getRange
  - touchgfx::AbstractProgressIndicator, 96
- getRect
  - touchgfx::Bitmap, 141
  - touchgfx::Drawable, 265
- getRedColor
  - touchgfx::Color, 219
- getRenderingAlgorithm
  - touchgfx::TextureMapper, 714
- getRenderingBuffer
  - touchgfx::Renderer, 573
- getRequiredNumberOfDrawables
  - touchgfx::DrawableList, 278
- getRootContainer
  - touchgfx::Screen, 593
- getRotation
  - touchgfx::TextArea, 684
- getScale
  - touchgfx::AbstractShape, 103
  - touchgfx::TextureMapper, 714
- getScalingAlgorithm
  - touchgfx::ScalableImage, 585
- getScalingMode
  - touchgfx::ZoomAnimationImage, 781
- getScanlineCounts
  - touchgfx::CanvasWidgetRenderer, 187
- getScanlineCovers
  - touchgfx::CanvasWidgetRenderer, 187
- getScanlineStartIndices
  - touchgfx::CanvasWidgetRenderer, 188
- getScanlineWidth
  - touchgfx::CanvasWidgetRenderer, 188
- getScrolledX
  - touchgfx::ScrollableContainer, 601
- getScrolledY
  - touchgfx::ScrollableContainer, 602
- getSelected

- touchgfx::RadioButton, 554
- getSelectedItem
  - touchgfx::ScrollWheelBase, 631
- getSelectedItemExtraSizeAfter
  - touchgfx::ScrollWheelWithSelectionStyle, 635
- getSelectedItemExtraSizeBefore
  - touchgfx::ScrollWheelWithSelectionStyle, 635
- getSelectedItemMarginAfter
  - touchgfx::ScrollWheelWithSelectionStyle, 636
- getSelectedItemMarginBefore
  - touchgfx::ScrollWheelWithSelectionStyle, 636
- getSelectedItemOffset
  - touchgfx::ScrollWheelBase, 631
- getSelectedRadioButton
  - touchgfx::RadioButtonsGroup, 560
- getSelectedRadioButtonIndex
  - touchgfx::RadioButtonsGroup, 561
- getShadeAlpha
  - touchgfx::ModalWindow, 477
- getShadeColor
  - touchgfx::ModalWindow, 478
- getSmallBitmap
  - touchgfx::ZoomAnimationImage, 781
- getSnapping
  - touchgfx::ScrollList, 625
- getSolidRect
  - touchgfx::Bitmap, 142
  - touchgfx::Box, 148
  - touchgfx::BoxWithBorder, 154
  - touchgfx::Button, 160
  - touchgfx::ButtonWithLabel, 169
  - touchgfx::CanvasWidget, 184
  - touchgfx::Container, 228
  - touchgfx::CoverTransition::FullSolidRect, 320
  - touchgfx::Drawable, 265
  - touchgfx::Image, 376
  - touchgfx::PixelDataWidget, 533
  - touchgfx::RadioButton, 555
  - touchgfx::ScalableImage, 585
  - touchgfx::SnapshotWidget, 674
  - touchgfx::TextArea, 684
  - touchgfx::TextureMapper, 714
  - touchgfx::TiledImage, 725
  - touchgfx::TouchArea, 736
- getSolidRectAbsolute
  - touchgfx::Drawable, 265
- getSpacingAbove
  - touchgfx::Font, 315
- getStart
  - touchgfx::Line, 450
  - touchgfx::LineProgress, 459
- getStartAngle
  - touchgfx::CircleProgress, 208
- getState
  - touchgfx::SlideMenu, 653
  - touchgfx::ToggleButton, 732
- getStateChangeButtonX
  - touchgfx::SlideMenu, 653
- getStateChangeButtonY
  - touchgfx::SlideMenu, 653
- getStringWidth
  - touchgfx::Font, 316
- getStringWidthLTR
  - touchgfx::Font, 317
- getStringWidthRTL
  - touchgfx::Font, 317
- getSwipeAcceleration
  - touchgfx::ScrollBase, 615
- getTFTCurrentLine
  - touchgfx::HAL, 350
- getTFTFrameBuffer
  - touchgfx::HALSDL2, 364
  - touchgfx::HAL, 351
- getText
  - touchgfx::Texts, 706
  - touchgfx::TypedText, 748
- getTextDirection
  - touchgfx::TypedText, 748
- getTextHeight
  - touchgfx::TextArea, 685
  - touchgfx::TextAreaWithOneWildcard, 691
  - touchgfx::TextAreaWithTwoWildcards, 693
- getTextWidth
  - touchgfx::DigitalClock, 242
  - touchgfx::TextArea, 685
  - touchgfx::TextAreaWithOneWildcard, 691
  - touchgfx::TextAreaWithTwoWildcards, 693
- getTimerWidgetCountForDrawable
  - touchgfx::Application, 130
- getToggleCanceled
  - touchgfx::ToggleButtonTrigger, 734
- getTouchSampleRate
  - touchgfx::HAL, 351
- getType
  - touchgfx::AbstractButton, 64
  - touchgfx::AnimatedImage, 117
  - touchgfx::Box, 148
  - touchgfx::BoxWithBorder, 155
  - touchgfx::Button, 161
  - touchgfx::ButtonWithIcon, 165
  - touchgfx::ButtonWithLabel, 169
  - touchgfx::ClickEvent, 215
  - touchgfx::Container, 228
  - touchgfx::DragEvent, 258
  - touchgfx::Drawable, 266
  - touchgfx::GestureEvent, 328
  - touchgfx::Image, 376
  - touchgfx::Keyboard, 394
  - touchgfx::ListLayout, 464
  - touchgfx::RadioButton, 555
  - touchgfx::ScalableImage, 585
  - touchgfx::ScrollableContainer, 602
  - touchgfx::Slider, 661
  - touchgfx::SnapshotWidget, 674
  - touchgfx::TextArea, 685
  - touchgfx::TextAreaWithOneWildcard, 691

- touchgfx::TextAreaWithTwoWildcards, 694
- touchgfx::TextureMapper, 715
- touchgfx::TiledImage, 726
- touchgfx::ToggleButton, 732
- touchgfx::TouchArea, 736
- touchgfx::Widget, 773
- touchgfx::ZoomAnimationImage, 782
- getTypedText
  - touchgfx::TextArea, 685
  - touchgfx::TextProgress, 701
- getUsedBufferSize
  - touchgfx::CanvasWidgetRenderer, 188
- getValue
  - touchgfx::AbstractProgressIndicator, 97
  - touchgfx::Slider, 661
- getValueRangeSize
  - touchgfx::Slider, 661
- getVelocity
  - touchgfx::GestureEvent, 328
- getVisiblePixelsWhenCollapsed
  - touchgfx::SlideMenu, 653
- getVisibleRect
  - touchgfx::Drawable, 266
- getWideTextAction
  - touchgfx::TextArea, 686
- getWidget
  - touchgfx::JSMOCHelper, 388
- getWidth
  - touchgfx::Bitmap, 142
  - touchgfx::Drawable, 266
  - touchgfx::JSMOCHelper, 388
  - touchgfx::RenderingBuffer, 576
- getWildcard
  - touchgfx::TextAreaWithOneWildcard, 691
- getWildcard1
  - touchgfx::TextAreaWithTwoWildcards, 694
- getWildcard2
  - touchgfx::TextAreaWithTwoWildcards, 694
- getWindowTitle
  - touchgfx::HALSDL2, 364
- getWriteMemoryUsageReport
  - touchgfx::CanvasWidgetRenderer, 188
- getX0
  - touchgfx::TextureMapper, 715
- getX1
  - touchgfx::TextureMapper, 715
- getX2
  - touchgfx::TextureMapper, 715
- getX3
  - touchgfx::TextureMapper, 716
- getXAdjust
  - touchgfx::RenderingBuffer, 576
- getXAngle
  - touchgfx::TextureMapper, 716
- getXBorder
  - touchgfx::ScrollableContainer, 602
- getXOffset
  - touchgfx::TiledImage, 726
- getXScrollbar
  - touchgfx::ScrollableContainer, 602
- getY0
  - touchgfx::TextureMapper, 716
- getY1
  - touchgfx::TextureMapper, 716
- getY2
  - touchgfx::TextureMapper, 716
- getY3
  - touchgfx::TextureMapper, 717
- getYAngle
  - touchgfx::TextureMapper, 717
- getYBorder
  - touchgfx::ScrollableContainer, 603
- getYOffset
  - touchgfx::TiledImage, 726
- getYScrollbar
  - touchgfx::ScrollableContainer, 603
- getZ0
  - touchgfx::TextureMapper, 717
- getZ1
  - touchgfx::TextureMapper, 717
- getZ2
  - touchgfx::TextureMapper, 717
- getZ3
  - touchgfx::TextureMapper, 718
- getZAngle
  - touchgfx::TextureMapper, 718
- getW
  - touchgfx::Quadruple, 550
- getX
  - touchgfx::ClickEvent, 215
  - touchgfx::Drawable, 267
  - touchgfx::GestureEvent, 329
  - touchgfx::Quadruple, 550
- getY
  - touchgfx::ClickEvent, 215
  - touchgfx::Drawable, 267
  - touchgfx::GestureEvent, 329
  - touchgfx::Quadruple, 550
  - touchgfx::Scanline, 589
- getZ
  - touchgfx::Quadruple, 550
- giveFrameBufferSemaphore
  - touchgfx::OSWrappers, 491
- giveFrameBufferSemaphoreFromISR
  - touchgfx::OSWrappers, 491
- GlyphFlags
  - touchgfx, 49
- GlyphNode, 331
- Gradient
  - touchgfx, 49
- Gradients, 335
  - touchgfx::Gradients, 336
- HALSDL2, 359
  - touchgfx::HALSDL2, 361
- HAL, 336
  - touchgfx::HAL, 341

- handleClickEvent
  - touchgfx::AbstractButton, 64
  - touchgfx::Application, 131
  - touchgfx::ClickButtonTrigger, 211
  - touchgfx::ClickListener, 217
  - touchgfx::Drawable, 267
  - touchgfx::Keyboard, 394
  - touchgfx::RadioButton, 555
  - touchgfx::RepeatButton, 579
  - touchgfx::RepeatButtonTrigger, 581
  - touchgfx::Screen, 593
  - touchgfx::ScrollList, 625
  - touchgfx::ScrollWheelBase, 631
  - touchgfx::ScrollableContainer, 603
  - touchgfx::Slider, 662
  - touchgfx::Snapper, 671
  - touchgfx::SwipeContainer, 678
  - touchgfx::ToggleButton, 732
  - touchgfx::ToggleButtonTrigger, 734
  - touchgfx::TouchArea, 737
  - touchgfx::TouchButtonTrigger, 738
  - touchgfx::UIEventListener, 750
- handleDragEvent
  - touchgfx::Application, 131
  - touchgfx::Draggable, 259
  - touchgfx::Drawable, 267
  - touchgfx::Keyboard, 395
  - touchgfx::Screen, 593
  - touchgfx::ScrollBase, 616
  - touchgfx::ScrollWheelBase, 632
  - touchgfx::ScrollableContainer, 603
  - touchgfx::Slider, 662
  - touchgfx::Snapper, 671
  - touchgfx::SwipeContainer, 678
  - touchgfx::TouchArea, 737
  - touchgfx::UIEventListener, 751
- handleGestureEvent
  - touchgfx::Application, 131
  - touchgfx::Drawable, 268
  - touchgfx::Screen, 594
  - touchgfx::ScrollBase, 616
  - touchgfx::ScrollWheelBase, 632
  - touchgfx::ScrollableContainer, 604
  - touchgfx::SwipeContainer, 679
  - touchgfx::UIEventListener, 751
- handleKeyEvent
  - touchgfx::Application, 131
  - touchgfx::Screen, 594
  - touchgfx::UIEventListener, 751
- handlePendingScreenTransition
  - touchgfx::Application, 132
  - touchgfx::MVPApplication, 485
  - touchgfx::UIEventListener, 751
- handleTickEvent
  - touchgfx::AnimatedImage, 118
  - touchgfx::AnimationTextureMapper, 125
  - touchgfx::Application, 132
  - touchgfx::CoverTransition, 232
  - touchgfx::Drawable, 268
  - touchgfx::FadeAnimator, 308
  - touchgfx::MoveAnimator, 482
  - touchgfx::NoTransition, 490
  - touchgfx::RepeatButton, 579
  - touchgfx::Screen, 594
  - touchgfx::ScrollBase, 616
  - touchgfx::ScrollableContainer, 604
  - touchgfx::SlideTransition, 668
  - touchgfx::SwipeContainer, 679
  - touchgfx::Transition, 741
  - touchgfx::UIEventListener, 751
  - touchgfx::ZoomAnimationImage, 782
- hasBuffer
  - touchgfx::CanvasWidgetRenderer, 188
- hasTransparentPixels
  - touchgfx::Bitmap, 142
- hasValidId
  - touchgfx::TypedText, 748
- height
  - touchgfx::GlyphNode, 332
- hide
  - touchgfx::ModalWindow, 478
- hw\_init
  - touchgfx, 53
- I2CTouchController, 370
  - touchgfx::I2CTouchController, 370
- I2C, 368
  - touchgfx::I2C, 368
- IconButtonStyle< T >, 371
- identity
  - touchgfx::Matrix4x4, 472
- Image, 374
  - touchgfx::Image, 375
- ImageButtonStyle< T >, 377
- ImageProgress, 379
  - touchgfx::ImageProgress, 380
- inbox
  - touchgfx::RenderingBuffer, 576
- includes
  - touchgfx::Rect, 569
- indexOf
  - touchgfx::AbstractPartition, 92
- init
  - touchgfx::ButtonController, 162
  - touchgfx::Buttons, 163
  - touchgfx::CoverTransition, 232
  - touchgfx::GPIO, 334
  - touchgfx::I2CTouchController, 371
  - touchgfx::I2C, 369
  - touchgfx::LCD16bpp, 417
  - touchgfx::LCD24bpp, 430
  - touchgfx::LCD2bpp, 436
  - touchgfx::LCD4bpp, 443
  - touchgfx::LCD, 409
  - touchgfx::LED, 444
  - touchgfx::MCUInstrumentation, 475
  - touchgfx::NoTouchController, 489

- touchgfx::SDL2TouchController, 641
- touchgfx::SDLTouchController, 642
- touchgfx::SlideTransition, 669
- touchgfx::TouchController, 740
- touchgfx::Transition, 742
- initMoveDrawable
  - touchgfx::CoverTransition, 232
  - touchgfx::SlideTransition, 669
- initialize
  - touchgfx::DMA\_Interface, 250
  - touchgfx::HAL, 351
  - touchgfx::OSWrappers, 491
  - touchgfx::ScrollBase, 616
  - touchgfx::ScrollWheelWithSelectionStyle, 636
  - touchgfx::TextProvider, 705
- initializeTime12Hour
  - touchgfx::AnalogClock, 111
- initializeTime24Hour
  - touchgfx::AnalogClock, 112
- insert
  - touchgfx::Container, 228
  - touchgfx::ListLayout, 464
- instance
  - touchgfx::Application, 133
- InternalFlashFont, 383
  - touchgfx::InternalFlashFont, 383
- intersect
  - touchgfx::Rect, 569, 570
- invalidate
  - touchgfx::CanvasWidget, 185
  - touchgfx::Drawable, 268
- invalidateArea
  - touchgfx::Application, 132
- invalidateRect
  - touchgfx::Drawable, 268
- invalidateScrollbars
  - touchgfx::ScrollableContainer, 604
- isAlphaPerPixel
  - touchgfx::Bitmap, 142
- isAnimatedImageRunning
  - touchgfx::AnimatedImage, 118
- isAnimating
  - touchgfx::ScrollBase, 616
- isDMARunning
  - touchgfx::DMA\_Interface, 251
- isDmaQueueEmpty
  - touchgfx::DMA\_Interface, 250
- isDmaQueueFull
  - touchgfx::DMA\_Interface, 251
- isDone
  - touchgfx::Transition, 742
- isEmpty
  - touchgfx::DMA\_Queue, 254
  - touchgfx::LockFreeDMA\_Queue, 467
  - touchgfx::Rect, 570
  - touchgfx::Vector, 767
- isFadeAnimationRunning
  - touchgfx::FadeAnimator, 308
- isFull
  - touchgfx::DMA\_Queue, 254
  - touchgfx::LockFreeDMA\_Queue, 467
- isMoveAnimationRunning
  - touchgfx::MoveAnimator, 482
- isPreRendered
  - touchgfx::PreRenderable, 537
- isReady
  - touchgfx::Scanline, 589
- isReverse
  - touchgfx::AnimatedImage, 118
- isRunning
  - touchgfx::AnimatedImage, 118
  - touchgfx::FadeAnimator, 308
  - touchgfx::MoveAnimator, 482
  - touchgfx::ZoomAnimationImage, 782
- isScrollableXY
  - touchgfx::ScrollableContainer, 604
- isShowing
  - touchgfx::ModalWindow, 478
- isTextureMapperAnimationRunning
  - touchgfx::AnimationTextureMapper, 125
- isTouchable
  - touchgfx::Drawable, 269
- isValid
  - touchgfx::Callback, 173
  - touchgfx::Callback< dest\_type, T1, T2, void >, 175
  - touchgfx::Callback< dest\_type, T1, void, void >, 176
  - touchgfx::Callback< dest\_type, void, void, void >, 178
  - touchgfx::GenericCallback, 322
  - touchgfx::GenericCallback< T1, T2, void >, 324
  - touchgfx::GenericCallback< T1, void, void >, 325
  - touchgfx::GenericCallback< void >, 326
- isVisible
  - touchgfx::Drawable, 269
- isZoomAnimationRunning
  - touchgfx::ZoomAnimationImage, 782
- itemChanged
  - touchgfx::DrawableList, 279
  - touchgfx::ScrollBase, 617
  - touchgfx::ScrollWheelWithSelectionStyle, 636
- iterator
  - touchgfx::Scanline::iterator, 385
- itoa
  - touchgfx::Unicode, 754
- JSMOCHelper, 386
  - touchgfx::JSMOCHelper, 387
- JSMOC
  - touchgfx::Screen, 594
- keepOffsetInsideLimits
  - touchgfx::ScrollBase, 617
  - touchgfx::ScrollList, 626
  - touchgfx::ScrollWheelBase, 632
- KerningNode, 389
- Keyboard, 389



- touchgfx::Keyboard, 392
- Keyboard::CallbackArea, 178
- Keyboard::Key, 389
- Keyboard::KeyMapping, 397
- Keyboard::KeyMappingList, 397
- Keyboard::Layout, 397
- LCD16bpp, 411
- LCD1bpp, 418
- LCD24bpp, 424
- LCD2bpp, 431
- LCD2getPixel
  - touchgfx, 54
- LCD2setPixel
  - touchgfx, 54
- LCD2shiftVal
  - touchgfx, 55
- LCD4bpp, 437
- LCD4getPixel
  - touchgfx, 55
- LCD4setPixel
  - touchgfx, 56
- LCD::StringVisuals, 675
- LCD, 398
- LED, 444
- LINE\_ENDING\_STYLE
  - touchgfx::Line, 447
- lcd
  - touchgfx::HAL, 351
- Line, 445
  - touchgfx::Line, 447
- LineProgress, 457
  - touchgfx::LineProgress, 458
- lineTo
  - touchgfx::Canvas, 180
  - touchgfx::Outline, 494
  - touchgfx::Rasterizer, 565
- linearEaseIn
  - touchgfx::EasingEquations, 296
- linearEaseInOut
  - touchgfx::EasingEquations, 296
- linearEaseNone
  - touchgfx::EasingEquations, 297
- linearEaseOut
  - touchgfx::EasingEquations, 297
- ListLayout, 462
  - touchgfx::ListLayout, 463
- loadSkin
  - touchgfx::HALSDL2, 364
- lockDMAToFrontPorch
  - touchgfx::HAL, 351
- lockFrameBuffer
  - touchgfx::HAL, 352
- LockFreeDMA\_Queue, 465
  - touchgfx::LockFreeDMA\_Queue, 466
- lookupBilinearRenderVariant
  - touchgfx, 56
- lookupNearestNeighborRenderVariant
  - touchgfx, 57
- lookupRenderVariant
  - touchgfx::ScalableImage, 585
  - touchgfx::TextureMapper, 718
- MAX\_TIMER\_WIDGETS
  - touchgfx::Application, 134
- MCUInstrumentation, 473
  - touchgfx::MCUInstrumentation, 473
- MVPApplication, 484
  - touchgfx::MVPApplication, 485
- MVPHeap, 485
  - touchgfx::MVPHeap, 486
- makeSnapshot
  - touchgfx::SnapshotWidget, 674
- makeTransition
  - touchgfx, 57
- Matrix4x4, 468
  - touchgfx::Matrix4x4, 469
- maxCapacity
  - touchgfx::Vector, 767
- memset
  - touchgfx, 58
- mixColors
  - touchgfx::AbstractPainterRGB565, 82
- ModalWindow, 475
  - touchgfx::ModalWindow, 476
- MoveAnimator
  - touchgfx::MoveAnimator, 481
- MoveAnimator< T >, 480
- moveChildrenRelative
  - touchgfx::Container, 229
  - touchgfx::ScrollableContainer, 604
- moveOrigin
  - touchgfx::AbstractShape, 103
- moveRelative
  - touchgfx::Drawable, 269
- moveTo
  - touchgfx::Canvas, 181
  - touchgfx::Drawable, 270
  - touchgfx::Outline, 495
  - touchgfx::Rasterizer, 565
- mulQ5
  - touchgfx::CWRUtil, 237
- muldiv
  - touchgfx, 58
- muldivQ5
  - touchgfx::CWRUtil, 237
- next
  - touchgfx::Scanline::iterator, 386
- nextFadeAnimationStep
  - touchgfx::FadeAnimator, 308
- nextLine
  - touchgfx::LCD16bpp, 417
  - touchgfx::LCD1bpp, 423
  - touchgfx::LCD24bpp, 430
  - touchgfx::LCD2bpp, 436
  - touchgfx::LCD4bpp, 443
- nextMoveAnimationStep

- touchgfx::MoveAnimator, 483
- nextPixel
  - touchgfx::LCD16bpp, 417
  - touchgfx::LCD1bpp, 424
  - touchgfx::LCD24bpp, 430
  - touchgfx::LCD2bpp, 437
  - touchgfx::LCD4bpp, 443
- NoDMA, 486
  - touchgfx::NoDMA, 487
- noTouch
  - touchgfx::HAL, 352
- NoTouchController, 488
- NoTransition, 490
  - touchgfx::NoTransition, 490
- numCellsMissing
  - touchgfx::CanvasWidgetRenderer, 189
- numCellsUsed
  - touchgfx::CanvasWidgetRenderer, 189
- OSWrappers, 491
- off
  - touchgfx::LED, 445
- on
  - touchgfx::LED, 445
- operation
  - touchgfx::BlitOp, 145
- operator int
  - touchgfx::CWRUtil::Q10, 540
  - touchgfx::CWRUtil::Q15, 542
  - touchgfx::CWRUtil::Q5, 545
- operator uint16\_t
  - touchgfx::colortype, 221
- operator!=
  - touchgfx::Bitmap, 142
  - touchgfx::Rect, 570
- operator\*
  - touchgfx, 58, 59
  - touchgfx::CWRUtil::Q10, 540
  - touchgfx::CWRUtil::Q5, 545, 546
- operator+
  - touchgfx::CWRUtil::Q10, 540
  - touchgfx::CWRUtil::Q15, 542
  - touchgfx::CWRUtil::Q5, 546
- operator-
  - touchgfx::CWRUtil::Q10, 541
  - touchgfx::CWRUtil::Q15, 543
  - touchgfx::CWRUtil::Q5, 546, 547
- operator/
  - touchgfx::CWRUtil::Q10, 541
  - touchgfx::CWRUtil::Q15, 543
  - touchgfx::CWRUtil::Q5, 547
- operator==
  - touchgfx::Bitmap, 143
  - touchgfx::Rect, 571
- operator&
  - touchgfx::Rect, 571
- operator&=
  - touchgfx::Rect, 571
- operator[]
  - touchgfx::DrawableListItems, 284
  - touchgfx::Vector, 767, 768
- Outline, 493
  - touchgfx::Outline, 494
- OutlineFlags\_t
  - touchgfx::Outline, 494
- packedCoord
  - touchgfx::Cell, 191
- PainterBWBitmap, 498
  - touchgfx::PainterBWBitmap, 499
- PainterBW, 496
- PainterGRAY2, 501
  - touchgfx::PainterGRAY2, 501
- PainterGRAY2Bitmap, 504
  - touchgfx::PainterGRAY2Bitmap, 505
- PainterGRAY4, 507
  - touchgfx::PainterGRAY4, 508
- PainterGRAY4Bitmap, 510
  - touchgfx::PainterGRAY4Bitmap, 511
- PainterRGB565, 513
  - touchgfx::PainterRGB565, 514
- PainterRGB565Bitmap, 517
  - touchgfx::PainterRGB565Bitmap, 518
- PainterRGB888, 520
  - touchgfx::PainterRGB888, 521
- PainterRGB888Bitmap, 523
  - touchgfx::PainterRGB888Bitmap, 524
- Pair
  - touchgfx::Pair, 527, 528
- Pair< T1, T2 >, 527
- Partition
  - touchgfx::Partition, 530
- Partition< ListOfTypes, NUMBER\_OF\_ELEMENTS >, 528
- pauseAnimation
  - touchgfx::AnimatedImage, 118
- performDisplayOrientationChange
  - touchgfx::HALSDL2, 365
  - touchgfx::HAL, 352
- PixelDataWidget, 531
  - touchgfx::PixelDataWidget, 532
- Point, 534
- Point3D, 534
- Point4, 535
  - touchgfx::Point4, 535
- pop
  - touchgfx::DMA\_Queue, 254
  - touchgfx::LockFreeDMA\_Queue, 467
- positionToValue
  - touchgfx::Slider, 662
- preRender
  - touchgfx::PreRenderable, 537
- PreRenderable
  - touchgfx::PreRenderable, 537
- PreRenderable< T >, 536
- prepareTransition
  - touchgfx, 59
- Presenter, 538



- touchgfx::Presenter, 538
- pushCopyOf
  - touchgfx::DMA\_Queue, 254
  - touchgfx::LockFreeDMA\_Queue, 467
- Q10
  - touchgfx::CWRUtil::Q10, 539, 540
- Q15
  - touchgfx::CWRUtil::Q15, 542
- Q5
  - touchgfx::CWRUtil::Q5, 544
- quadEaseIn
  - touchgfx::EasingEquations, 297
- quadEaseInOut
  - touchgfx::EasingEquations, 298
- quadEaseOut
  - touchgfx::EasingEquations, 298
- Quadruple, 548
  - touchgfx::Quadruple, 549
- quartEaseIn
  - touchgfx::EasingEquations, 299
- quartEaseInOut
  - touchgfx::EasingEquations, 299
- quartEaseOut
  - touchgfx::EasingEquations, 300
- quickRemoveAt
  - touchgfx::Vector, 768
- quintEaseIn
  - touchgfx::EasingEquations, 300
- quintEaseInOut
  - touchgfx::EasingEquations, 300
- quintEaseOut
  - touchgfx::EasingEquations, 301
- RadioButton, 552
  - touchgfx::RadioButton, 553
- radioButtonClickedHandler
  - touchgfx::RadioButtonGroup, 561
- radioButtonDeselectedHandler
  - touchgfx::RadioButtonGroup, 561
- RadioButtonGroup
  - touchgfx::RadioButtonGroup, 559
- RadioButtonGroup< CAPACITY >, 558
- Rasterizer, 563
  - touchgfx::Rasterizer, 565
- readRegister
  - touchgfx::I2C, 369
- realX
  - touchgfx::LCD, 409
- realY
  - touchgfx::LCD, 410
- Rect, 567
  - touchgfx::Rect, 568
- refreshDrawableListsLayout
  - touchgfx::ScrollWheelWithSelectionMode, 637
- refreshDrawables
  - touchgfx::DrawableList, 279
- registerBitmapDatabase
  - touchgfx::Bitmap, 143
- registerClickEvent
  - touchgfx::Gestures, 330
- registerDragEvent
  - touchgfx::Gestures, 330
- registerEventListener
  - touchgfx::Gestures, 330
  - touchgfx::HAL, 352
- registerTaskDelayFunction
  - touchgfx::HAL, 353
- registerTextCache
  - touchgfx::HAL, 353
- registerTexts
  - touchgfx::TypedText, 749
- registerTimerWidget
  - touchgfx::Application, 132
- registerTypedTextDatabase
  - touchgfx::TypedText, 749
- remove
  - touchgfx::Container, 229
  - touchgfx::ListLayout, 465
  - touchgfx::ModalWindow, 478
  - touchgfx::Screen, 594
  - touchgfx::SlideMenu, 653
  - touchgfx::SwipeContainer, 679
  - touchgfx::Vector, 768
- removeAll
  - touchgfx::Container, 229
  - touchgfx::ListLayout, 465
- removeAt
  - touchgfx::Vector, 768
- render
  - touchgfx::AbstractPainter, 72
  - touchgfx::AbstractPainterBW, 75
  - touchgfx::AbstractPainterGRAY2, 77
  - touchgfx::AbstractPainterGRAY4, 79
  - touchgfx::AbstractPainterRGB565, 82
  - touchgfx::AbstractPainterRGB888, 85
  - touchgfx::Canvas, 181
  - touchgfx::PainterBWBitmap, 499
  - touchgfx::PainterBW, 497
  - touchgfx::PainterGRAY2, 502
  - touchgfx::PainterGRAY2Bitmap, 505
  - touchgfx::PainterGRAY4, 509
  - touchgfx::PainterGRAY4Bitmap, 512
  - touchgfx::PainterRGB565, 515
  - touchgfx::PainterRGB565Bitmap, 518
  - touchgfx::PainterRGB888, 522
  - touchgfx::PainterRGB888Bitmap, 525
  - touchgfx::Rasterizer, 566
  - touchgfx::Renderer, 573
- renderInit
  - touchgfx::AbstractPainterBW, 76
  - touchgfx::AbstractPainterGRAY2, 77
  - touchgfx::AbstractPainterGRAY4, 80
  - touchgfx::AbstractPainterRGB565, 83
  - touchgfx::AbstractPainterRGB888, 85
  - touchgfx::PainterBWBitmap, 500
  - touchgfx::PainterGRAY2Bitmap, 506

- touchgfx::PainterGRAY4Bitmap, 512
- touchgfx::PainterRGB565Bitmap, 519
- touchgfx::PainterRGB888Bitmap, 525
- renderLCD\_FrameBufferToMemory
  - touchgfx::HALSDL2, 365
- renderNext
  - touchgfx::AbstractPainterBW, 76
  - touchgfx::AbstractPainterGRAY2, 78
  - touchgfx::AbstractPainterGRAY4, 80
  - touchgfx::AbstractPainterRGB565, 83
  - touchgfx::AbstractPainterRGB888, 85
  - touchgfx::PainterBWBitmap, 500
  - touchgfx::PainterBW, 498
  - touchgfx::PainterGRAY2, 503
  - touchgfx::PainterGRAY2Bitmap, 506
  - touchgfx::PainterGRAY4, 509
  - touchgfx::PainterGRAY4Bitmap, 512
  - touchgfx::PainterRGB565, 516
  - touchgfx::PainterRGB565Bitmap, 519
  - touchgfx::PainterRGB888, 522
  - touchgfx::PainterRGB888Bitmap, 526
- renderPixel
  - touchgfx::AbstractPainterGRAY2, 78
  - touchgfx::AbstractPainterGRAY4, 80
  - touchgfx::AbstractPainterRGB565, 84
  - touchgfx::AbstractPainterRGB888, 86
- Renderer, 572
  - touchgfx::Renderer, 572
- RenderingAlgorithm
  - touchgfx::TextureMapper, 710
- RenderingBuffer, 573
  - touchgfx::RenderingBuffer, 575
- RenderingVariant
  - touchgfx, 47
- RepeatButton, 577
  - touchgfx::RepeatButton, 578
- RepeatButtonTrigger, 580
- reportAsSolid
  - touchgfx::Box, 149
- reset
  - touchgfx::ButtonController, 162
  - touchgfx::Outline, 495
  - touchgfx::Rasterizer, 566
  - touchgfx::Scanline, 590
  - touchgfx::ScrollableContainer, 605
- resetDrawChainCache
  - touchgfx::Drawable, 270
- resetExpandedStateTimer
  - touchgfx::SlideMenu, 654
- resetSpans
  - touchgfx::Scanline, 590
- resizeHeightToCurrentText
  - touchgfx::TextArea, 686
- resizeToCurrentText
  - touchgfx::TextArea, 686
- reverse
  - touchgfx::Vector, 769
- right
  - touchgfx::Rect, 571
- rotate0
  - touchgfx, 49
- rotate90
  - touchgfx, 49
- rotateRect
  - touchgfx::LCD, 410
- row
  - touchgfx::RenderingBuffer, 577
- SDL2TouchController, 641
- SDLTouchController, 642
- sample
  - touchgfx::ButtonController, 162
  - touchgfx::Buttons, 163
- sampleKey
  - touchgfx::HALSDL2, 365
  - touchgfx::HAL, 353
- sampleTouch
  - touchgfx::I2CTouchController, 371
  - touchgfx::NoTouchController, 489
  - touchgfx::SDL2TouchController, 642
  - touchgfx::SDLTouchController, 643
  - touchgfx::TouchController, 740
- saveNextScreenshots
  - touchgfx::HALSDL2, 365
- saveScreenshot
  - touchgfx::HALSDL2, 366
- ScalableImage, 582
  - touchgfx::ScalableImage, 583
- ScalingAlgorithm
  - touchgfx::ScalableImage, 583
- Scanline, 586
  - touchgfx::Scanline, 588
- Scanline::iterator, 385
- Screen, 590
  - touchgfx::Screen, 591
- ScrollBase, 607
  - touchgfx::ScrollBase, 610
- ScrollList, 622
  - touchgfx::ScrollList, 623
- ScrollWheel, 628
  - touchgfx::ScrollWheel, 628
- ScrollWheelBase, 629
  - touchgfx::ScrollWheelBase, 630
- ScrollWheelWithSelectionStyle, 633
  - touchgfx::ScrollWheelWithSelectionStyle, 635
- ScrollableContainer, 596
  - touchgfx::ScrollableContainer, 599
- sdl\_init
  - touchgfx::HALSDL2, 366
- seedExecution
  - touchgfx::DMA\_Interface, 251
- set
  - touchgfx::Cell, 192
  - touchgfx::GPIO, 334
- setAction
  - touchgfx::AbstractButton, 65
  - touchgfx::AbstractButtonContainer, 66

- setAllowed
  - touchgfx::DMA\_Interface, 251
- setAlpha
  - touchgfx::AbstractButtonContainer, 67
  - touchgfx::Box, 148
  - touchgfx::BoxProgress, 151
  - touchgfx::BoxWithBorder, 155
  - touchgfx::Button, 161
  - touchgfx::CanvasWidget, 185
  - touchgfx::CircleProgress, 208
  - touchgfx::DigitalClock, 242
  - touchgfx::Image, 377
  - touchgfx::ImageProgress, 381
  - touchgfx::LineProgress, 460
  - touchgfx::PainterGRAY2, 503
  - touchgfx::PainterGRAY2Bitmap, 506
  - touchgfx::PainterGRAY4, 510
  - touchgfx::PainterGRAY4Bitmap, 513
  - touchgfx::PainterRGB565, 516
  - touchgfx::PainterRGB565Bitmap, 520
  - touchgfx::PainterRGB888, 523
  - touchgfx::PainterRGB888Bitmap, 526
  - touchgfx::PixelDataWidget, 533
  - touchgfx::RadioButton, 555
  - touchgfx::ScalableImage, 586
  - touchgfx::SnapshotWidget, 675
  - touchgfx::TextArea, 686
  - touchgfx::TextProgress, 701
  - touchgfx::TextureMapper, 718
  - touchgfx::ZoomAnimationImage, 782
- setAnchorAtZero
  - touchgfx::ImageProgress, 381
- setAngle
  - touchgfx::AbstractShape, 104
- setAnimateToCallback
  - touchgfx::ScrollWheelBase, 632
- setAnimation
  - touchgfx::AnalogClock, 112
- setAnimationDelay
  - touchgfx::ZoomAnimationImage, 783
- setAnimationDuration
  - touchgfx::SlideMenu, 654
- setAnimationEasingEquation
  - touchgfx::SlideMenu, 654
- setAnimationEndedCallback
  - touchgfx::ScrollBase, 617
  - touchgfx::ZoomAnimationImage, 783
- setAnimationSteps
  - touchgfx::ScrollBase, 618
- setArc
  - touchgfx::Circle, 199, 200
- setBackground
  - touchgfx::AbstractProgressIndicator, 97
  - touchgfx::AnalogClock, 112
  - touchgfx::ModalWindow, 478, 479
- setBaselineY
  - touchgfx::DigitalClock, 242
  - touchgfx::TextArea, 687
- setBitmap
  - touchgfx::AnimatedImage, 119
  - touchgfx::Image, 377
  - touchgfx::ImageProgress, 382
  - touchgfx::PainterBWBitmap, 500
  - touchgfx::PainterGRAY2Bitmap, 507
  - touchgfx::PainterGRAY4Bitmap, 513
  - touchgfx::PainterRGB565Bitmap, 520
  - touchgfx::PainterRGB888Bitmap, 526
  - touchgfx::ScalableImage, 586
  - touchgfx::TextureMapper, 719
  - touchgfx::TiledImage, 726
- setBitmapFormat
  - touchgfx::PixelDataWidget, 533
- setBitmapPosition
  - touchgfx::TextureMapper, 719
- setBitmapXY
  - touchgfx::AnimatedImageButtonStyle, 121
  - touchgfx::ImageButtonStyle, 379
- setBitmaps
  - touchgfx::AnimatedImage, 119
  - touchgfx::AnimatedImageButtonStyle, 121
  - touchgfx::Button, 161
  - touchgfx::ButtonWithIcon, 165
  - touchgfx::ImageButtonStyle, 378
  - touchgfx::RadioButton, 556
  - touchgfx::Slider, 662, 663
  - touchgfx::ToggleButton, 733
  - touchgfx::ZoomAnimationImage, 783
- setBorderColor
  - touchgfx::BoxWithBorder, 155
- setBorderSize
  - touchgfx::BoxWithBorder, 155
  - touchgfx::BoxWithBorderButtonStyle, 157
- setBoxWithBorderColors
  - touchgfx::BoxWithBorderButtonStyle, 157
- setBoxWithBorderHeight
  - touchgfx::BoxWithBorderButtonStyle, 158
- setBoxWithBorderPosition
  - touchgfx::BoxWithBorderButtonStyle, 158
- setBoxWithBorderWidth
  - touchgfx::BoxWithBorderButtonStyle, 158
- setBuffer
  - touchgfx::Keyboard, 395
- setBufferPosition
  - touchgfx::Keyboard, 395
- setButtonController
  - touchgfx::HAL, 354
- setCCConsumed
  - touchgfx::MCUInstrumentation, 475
- setCache
  - touchgfx::AbstractShape, 104
  - touchgfx::Bitmap, 143
  - touchgfx::Shape, 646
- setCalibrationMatrix
  - touchgfx::TouchCalibration, 739
- setCamera
  - touchgfx::TextureMapper, 719

- setCameraDistance
  - touchgfx::TextureMapper, 720
- setCapPrecision
  - touchgfx::Circle, 200
  - touchgfx::CircleProgress, 208
  - touchgfx::Line, 450
- setCenter
  - touchgfx::Circle, 201
  - touchgfx::CircleProgress, 209
- setCircle
  - touchgfx::Circle, 201, 202
- setCircular
  - touchgfx::DrawableList, 279
  - touchgfx::ScrollBase, 618
  - touchgfx::ScrollWheelWithSelectionStyle, 637
- setClickAction
  - touchgfx::ClickListener, 217
- setColor
  - touchgfx::Box, 149
  - touchgfx::BoxProgress, 151
  - touchgfx::BoxWithBorder, 156
  - touchgfx::DigitalClock, 243
  - touchgfx::PainterBW, 498
  - touchgfx::PainterGRAY2, 503
  - touchgfx::PainterGRAY4, 510
  - touchgfx::PainterRGB565, 516
  - touchgfx::PainterRGB888, 523
  - touchgfx::TextArea, 687
  - touchgfx::TextProgress, 702
- setCoord
  - touchgfx::Cell, 192
- setCorner
  - touchgfx::AbstractShape, 105
  - touchgfx::Shape, 646
- setCover
  - touchgfx::Cell, 192
- setCurrentState
  - touchgfx::ZoomAnimationImage, 783
- setDelay
  - touchgfx::RepeatButton, 579
  - touchgfx::RepeatButtonTrigger, 581
- setDeselectedAction
  - touchgfx::RadioButton, 556
- setDeselectionEnabled
  - touchgfx::RadioButton, 556
  - touchgfx::RadioButtonGroup, 561
- setDimension
  - touchgfx::ZoomAnimationImage, 784
- setDirection
  - touchgfx::AbstractDirectionProgress, 71
  - touchgfx::ListLayout, 465
- setDisplayMode
  - touchgfx::DigitalClock, 243
- setDisplayOrientation
  - touchgfx::HAL, 354
- setDoneAction
  - touchgfx::AnimatedImage, 119
- setDragAcceleration
  - touchgfx::ScrollBase, 618
- setDragAction
  - touchgfx::Snapper, 671
- setDragThreshold
  - touchgfx::Gestures, 331
  - touchgfx::HAL, 354
- setDrawableSize
  - touchgfx::DrawableList, 280
  - touchgfx::ScrollBase, 619
  - touchgfx::ScrollWheelWithSelectionStyle, 638
- setDrawables
  - touchgfx::DrawableList, 280
  - touchgfx::ScrollList, 626
  - touchgfx::ScrollWheel, 628
  - touchgfx::ScrollWheelWithSelectionStyle, 637
- setEasingEquation
  - touchgfx::ScrollBase, 619
- setElement
  - touchgfx::Matrix4x4, 472
  - touchgfx::Quadruple, 550
- setEnd
  - touchgfx::Line, 451
  - touchgfx::LineProgress, 460
- setEndSwipeElasticWidth
  - touchgfx::SwipeContainer, 679
- setExpandDirection
  - touchgfx::SlideMenu, 654
- setExpandedStateTimeout
  - touchgfx::SlideMenu, 654
- setFadeAnimationDelay
  - touchgfx::FadeAnimator, 309
- setFadeAnimationEndedAction
  - touchgfx::FadeAnimator, 309
- setFillingRule
  - touchgfx::Rasterizer, 566
- setFingerSize
  - touchgfx::HAL, 355
- setFontProvider
  - touchgfx::FontManager, 318
- setFrameBufferStartAddress
  - touchgfx::HAL, 355
- setFrameBufferStartAddresses
  - touchgfx::HAL, 355
- setFrameRateCompensation
  - touchgfx::HAL, 356
- setFrameRefreshStrategy
  - touchgfx::HAL, 356
- setHeight
  - touchgfx::DigitalClock, 243
  - touchgfx::Drawable, 270
  - touchgfx::DrawableList, 280
  - touchgfx::ScrollBase, 619
  - touchgfx::ScrollWheelWithSelectionStyle, 638
  - touchgfx::TiledImageButton, 730
  - touchgfx::ZoomAnimationImage, 784
- setHiddenPixelsWhenExpanded
  - touchgfx::SlideMenu, 655
- setHorizontal

- touchgfx::DrawableList, 281
- touchgfx::ScrollBase, 620
- touchgfx::ScrollWheelWithSelectionStyle, 638
- setHourHandMinuteCorrection
  - touchgfx::AnalogClock, 113
- setIconBitmaps
  - touchgfx::IconButtonStyle, 373
- setIconXY
  - touchgfx::ButtonWithIcon, 166
  - touchgfx::IconButtonStyle, 374
- setIconX
  - touchgfx::ButtonWithIcon, 166
  - touchgfx::IconButtonStyle, 373
- setIconY
  - touchgfx::ButtonWithIcon, 167
  - touchgfx::IconButtonStyle, 374
- setIndentation
  - touchgfx::TextArea, 687
- setInterval
  - touchgfx::RepeatButton, 580
  - touchgfx::RepeatButtonTrigger, 581
- setItemPressedCallback
  - touchgfx::ScrollBase, 620
- setItemSelectedCallback
  - touchgfx::ScrollBase, 620
- setKeyListener
  - touchgfx::Keyboard, 395
- setKeymappingList
  - touchgfx::Keyboard, 396
- setLabelColor
  - touchgfx::ButtonWithLabel, 169
- setLabelColorPressed
  - touchgfx::ButtonWithLabel, 170
- setLabelRotation
  - touchgfx::ButtonWithLabel, 170
- setLabelText
  - touchgfx::ButtonWithLabel, 170
- setLanguage
  - touchgfx::Texts, 706
- setLayout
  - touchgfx::Keyboard, 396
- setLine
  - touchgfx::Line, 451
- setLineEndingStyle
  - touchgfx::Line, 452
  - touchgfx::LineProgress, 460
- setLineWidth
  - touchgfx::Circle, 202
  - touchgfx::CircleProgress, 209
  - touchgfx::Line, 452, 453
  - touchgfx::LineProgress, 461
- setLinespacing
  - touchgfx::TextArea, 688
- setMCUActive
  - touchgfx::HAL, 357
  - touchgfx::MCUInstrumentation, 475
- setMCUInstrumentation
  - touchgfx::HAL, 357
- setMaxRenderY
  - touchgfx::Outline, 495
  - touchgfx::Rasterizer, 566
- setMaxSwipeItems
  - touchgfx::ScrollBase, 620
- setMaxVelocity
  - touchgfx::ScrollableContainer, 605
- setMinuteHandSecondCorrection
  - touchgfx::AnalogClock, 113
- setMoveAnimationDelay
  - touchgfx::MoveAnimator, 483
- setMoveAnimationEndedAction
  - touchgfx::MoveAnimator, 483
- setNewValueCallback
  - touchgfx::Slider, 663
- setNumberOfDecimals
  - touchgfx::TextProgress, 702
- setNumberOfItems
  - touchgfx::DrawableList, 281
  - touchgfx::ScrollBase, 621
  - touchgfx::ScrollWheelWithSelectionStyle, 639
- setOffset
  - touchgfx::AbstractPainter, 74
  - touchgfx::DrawableList, 281
  - touchgfx::ScrollBase, 621
  - touchgfx::ScrollWheelWithSelectionStyle, 639
  - touchgfx::TiledImage, 727
- setOrigin
  - touchgfx::AbstractShape, 105
- setOrigo
  - touchgfx::TextureMapper, 720
- setPadding
  - touchgfx::ScrollList, 626
- setPageIndicatorBitmaps
  - touchgfx::SwipeContainer, 680
- setPageIndicatorXYWithCenteredX
  - touchgfx::SwipeContainer, 680
- setPageIndicatorXY
  - touchgfx::SwipeContainer, 680
- setPainter
  - touchgfx::CanvasWidget, 185
  - touchgfx::CircleProgress, 209
  - touchgfx::LineProgress, 461
- setPixelData
  - touchgfx::PixelDataWidget, 533
- setPosition
  - touchgfx::Drawable, 270
  - touchgfx::ZoomAnimationImage, 784
- setPrecision
  - touchgfx::Circle, 203
- setPressed
  - touchgfx::AbstractButtonContainer, 67
- setPressedAction
  - touchgfx::TouchArea, 737
- setProgressIndicatorPosition
  - touchgfx::AbstractProgressIndicator, 97
  - touchgfx::BoxProgress, 151
  - touchgfx::CircleProgress, 209

- touchgfx::ImageProgress, 382
- touchgfx::LineProgress, 461
- touchgfx::TextProgress, 702
- setRadioButtonDeselectedHandler
  - touchgfx::RadioButtonsGroup, 562
- setRadioButtonSelectedHandler
  - touchgfx::RadioButtonsGroup, 562
- setRadius
  - touchgfx::Circle, 203
  - touchgfx::CircleProgress, 210
- setRange
  - touchgfx::AbstractProgressIndicator, 97
- setRenderingAlgorithm
  - touchgfx::TextureMapper, 720
- setRenderingBuffer
  - touchgfx::Renderer, 573
- setRotation
  - touchgfx::TextArea, 688
- setRotationCenter
  - touchgfx::AnalogClock, 113
- setScale
  - touchgfx::AbstractShape, 106
  - touchgfx::TextureMapper, 721
- setScalingAlgorithm
  - touchgfx::ScalableImage, 586
- setScalingMode
  - touchgfx::ZoomAnimationImage, 785
- setScanlineWidth
  - touchgfx::CanvasWidgetRenderer, 189
- setScreenContainer
  - touchgfx::Transition, 742
- setScrollThreshold
  - touchgfx::ScrollableContainer, 606
- setScrollbarPadding
  - touchgfx::ScrollableContainer, 605
- setScrollbarWidth
  - touchgfx::ScrollableContainer, 606
- setScrollbarsAlpha
  - touchgfx::ScrollableContainer, 605
- setScrollbarsColor
  - touchgfx::ScrollableContainer, 606
- setScrollbarsPermanentlyVisible
  - touchgfx::ScrollableContainer, 606
- setScrollbarsVisible
  - touchgfx::ScrollableContainer, 606
- setSelectItemOffset
  - touchgfx::ScrollWheelWithSelectionStyle, 641
- setSelected
  - touchgfx::RadioButton, 558
  - touchgfx::RadioButtonsGroup, 562
- setSelectedItemExtraSize
  - touchgfx::ScrollWheelWithSelectionStyle, 639
- setSelectedItemMargin
  - touchgfx::ScrollWheelWithSelectionStyle, 640
- setSelectedItemOffset
  - touchgfx::ScrollWheelBase, 633
- setSelectedItemPosition
  - touchgfx::ScrollWheelWithSelectionStyle, 640
- setSelectedPage
  - touchgfx::SwipeContainer, 680
- setShadeAlpha
  - touchgfx::ModalWindow, 479
- setShadeColor
  - touchgfx::ModalWindow, 479
- setShape
  - touchgfx::AbstractShape, 107
- setSnapPosition
  - touchgfx::Snapper, 672
- setSnappedAction
  - touchgfx::Snapper, 672
- setSnapping
  - touchgfx::ScrollList, 627
- setStart
  - touchgfx::Line, 453, 454
  - touchgfx::LineProgress, 462
- setStartEndAngle
  - touchgfx::CircleProgress, 210
- setStartValueCallback
  - touchgfx::Slider, 664
- setState
  - touchgfx::SlideMenu, 655
- setStateChangedAnimationEndedCallback
  - touchgfx::SlideMenu, 655
- setStateChangedCallback
  - touchgfx::SlideMenu, 655
- setStopValueCallback
  - touchgfx::Slider, 664
- setSwipeAcceleration
  - touchgfx::ScrollBase, 621
- setSwipeCutoff
  - touchgfx::SwipeContainer, 681
- setTFTFrameBuffer
  - touchgfx::HALSDL2, 366
  - touchgfx::HAL, 357
- setText
  - touchgfx::TextButtonStyle, 697
- setTextColors
  - touchgfx::TextButtonStyle, 697
- setTextIndentation
  - touchgfx::Keyboard, 396
- setTextPosition
  - touchgfx::TextButtonStyle, 698
- setTextRotation
  - touchgfx::TextButtonStyle, 698
- setTextXY
  - touchgfx::TextButtonStyle, 698
- setTextureMapperAnimationEndedAction
  - touchgfx::AnimationTextureMapper, 125
- setTextureMapperAnimationStepAction
  - touchgfx::AnimationTextureMapper, 125
- setTextX
  - touchgfx::TextButtonStyle, 698
- setTextY
  - touchgfx::TextButtonStyle, 699
- setTileBitmaps
  - touchgfx::TiledImageButtonStyle, 730



- setTileOffset
  - touchgfx::TiledImageButtonStyle, 730
- setTime12Hour
  - touchgfx::AbstractClock, 69
- setTime24Hour
  - touchgfx::AbstractClock, 69
- setToggleCanceled
  - touchgfx::ToggleButtonTrigger, 735
- setTouchSampleRate
  - touchgfx::HAL, 357
- setTouchable
  - touchgfx::Drawable, 271
- setTwoWildcardText
  - touchgfx::TwoWildcardTextButtonStyle, 744
- setTwoWildcardTextColors
  - touchgfx::TwoWildcardTextButtonStyle, 744
- setTwoWildcardTextPosition
  - touchgfx::TwoWildcardTextButtonStyle, 744
- setTwoWildcardTextRotation
  - touchgfx::TwoWildcardTextButtonStyle, 745
- setTwoWildcardTextXY
  - touchgfx::TwoWildcardTextButtonStyle, 745
- setTwoWildcardTextX
  - touchgfx::TwoWildcardTextButtonStyle, 745
- setTwoWildcardTextY
  - touchgfx::TwoWildcardTextButtonStyle, 745
- setType
  - touchgfx::ClickEvent, 215
- setTypedText
  - touchgfx::DigitalClock, 243
  - touchgfx::TextArea, 688
  - touchgfx::TextProgress, 703
- setUpdateTicksInterval
  - touchgfx::AnimatedImage, 119
  - touchgfx::AnimatedImageButtonStyle, 122
- setValue
  - touchgfx::AbstractProgressIndicator, 98
  - touchgfx::BoxProgress, 152
  - touchgfx::CircleProgress, 210
  - touchgfx::ImageProgress, 382
  - touchgfx::LineProgress, 462
  - touchgfx::Slider, 665
  - touchgfx::TextProgress, 703
- setValueRange
  - touchgfx::Slider, 666
- setViewDistance
  - touchgfx::Matrix4x4, 472
- setVisible
  - touchgfx::Drawable, 272
- setVisiblePixelsWhenCollapsed
  - touchgfx::SlideMenu, 657
- setVsyncInterval
  - touchgfx::HALSDL2, 367
- setWidthTextAction
  - touchgfx::TextArea, 688
- setWidthWidget
  - touchgfx::JSMOCHelper, 388
- setWidthWidgetAlpha
  - touchgfx::AbstractPainter, 74
- setWidth
  - touchgfx::DigitalClock, 244
  - touchgfx::Drawable, 272
  - touchgfx::DrawableList, 282
  - touchgfx::ScrollBase, 622
  - touchgfx::ScrollWheelWithSelectionStyle, 641
  - touchgfx::TiledImageButtonStyle, 731
  - touchgfx::ZoomAnimationImage, 785
- setWildcard
  - touchgfx::TextAreaWithOneWildcard, 691
- setWildcard1
  - touchgfx::TextAreaWithTwoWildcards, 694
- setWildcard2
  - touchgfx::TextAreaWithTwoWildcards, 694
- setWildcardText
  - touchgfx::WildcardTextButtonStyle, 775
- setWildcardTextBuffer
  - touchgfx::WildcardTextButtonStyle, 775
- setWildcardTextBuffer1
  - touchgfx::TwoWildcardTextButtonStyle, 746
- setWildcardTextBuffer2
  - touchgfx::TwoWildcardTextButtonStyle, 746
- setWildcardTextColors
  - touchgfx::WildcardTextButtonStyle, 775
- setWildcardTextPosition
  - touchgfx::WildcardTextButtonStyle, 775
- setWildcardTextRotation
  - touchgfx::WildcardTextButtonStyle, 776
- setWildcardTextXY
  - touchgfx::WildcardTextButtonStyle, 776
- setWildcardTextX
  - touchgfx::WildcardTextButtonStyle, 776
- setWildcardTextY
  - touchgfx::WildcardTextButtonStyle, 776
- setWindowSize
  - touchgfx::ScrollList, 627
- setWindowTitle
  - touchgfx::HALSDL2, 367
- setWriteMemoryUsageReport
  - touchgfx::CanvasWidgetRenderer, 190
- setXBaselineY
  - touchgfx::TextArea, 689
- setXOffset
  - touchgfx::TiledImage, 727
- setXY
  - touchgfx::Drawable, 273
- setYOffset
  - touchgfx::TiledImage, 727
- setup
  - touchgfx::SlideMenu, 656
- setupAnimation
  - touchgfx::AnimationTextureMapper, 126
- setupBuffer
  - touchgfx::CanvasWidgetRenderer, 189
- setupDataCopy
  - touchgfx::DMA\_Interface, 251
  - touchgfx::NoDMA, 488

- setupDataFill
  - touchgfx::DMA\_Interface, 252
  - touchgfx::NoDMA, 488
- setupDrawChain
  - touchgfx::Container, 230
  - touchgfx::Drawable, 271
  - touchgfx::Keyboard, 396
  - touchgfx::PreRenderable, 537
- setupHand
  - touchgfx::AnalogClock, 114
- setupHorizontalSlider
  - touchgfx::Slider, 664
- setupHourHand
  - touchgfx::AnalogClock, 114
- setupMinuteHand
  - touchgfx::AnalogClock, 114
- setupScreen
  - touchgfx::Screen, 595
- setupSecondHand
  - touchgfx::AnalogClock, 115
- setupVerticalSlider
  - touchgfx::Slider, 665
- setW
  - touchgfx::Quadruple, 551
- setX
  - touchgfx::ClickEvent, 216
  - touchgfx::Drawable, 272
  - touchgfx::Quadruple, 551
- setY
  - touchgfx::ClickEvent, 216
  - touchgfx::Drawable, 273
  - touchgfx::Quadruple, 551
- setZ
  - touchgfx::Quadruple, 551
- Shape< POINTS >, 643
- show
  - touchgfx::ModalWindow, 480
- signalDMAInterrupt
  - touchgfx::DMA\_Interface, 252
  - touchgfx::HAL, 358
  - touchgfx::NoDMA, 488
- signalVSync
  - touchgfx::OSWrappers, 491
- sine
  - touchgfx::CWRUtil, 237, 238
- sineEaseIn
  - touchgfx::EasingEquations, 301
- sineEaseInOut
  - touchgfx::EasingEquations, 302
- sineEaseOut
  - touchgfx::EasingEquations, 302
- size
  - touchgfx::Vector, 769
- SlideMenu, 647
- SlideTransition
  - touchgfx::SlideTransition, 668
- SlideTransition< templateDirection >, 667
- Slider, 657
  - touchgfx::Slider, 659
- Snapper
  - touchgfx::Snapper, 670
- Snapper< T >, 670
- SnapshotWidget, 672
  - touchgfx::SnapshotWidget, 673
- snprintf
  - touchgfx::Unicode, 754, 755
- snprintfFloat
  - touchgfx::Unicode, 756, 757
- snprintfFloats
  - touchgfx::Unicode, 758, 760
- sqrtQ10
  - touchgfx::CWRUtil, 238
- start
  - touchgfx::DMA\_Interface, 252
- startAnimation
  - touchgfx::AnimatedImage, 120
  - touchgfx::AnimationTextureMapper, 126
- startFadeAnimation
  - touchgfx::FadeAnimator, 309
- startMoveAnimation
  - touchgfx::MoveAnimator, 483
- startSMOC
  - touchgfx::Screen, 595
- startTimerAndSetParameters
  - touchgfx::ZoomAnimationImage, 785
- startZoomAndMoveAnimation
  - touchgfx::ZoomAnimationImage, 786
- startZoomAnimation
  - touchgfx::ZoomAnimationImage, 787
- stateChangeButtonClickedHandler
  - touchgfx::SlideMenu, 657
- States
  - touchgfx::ZoomAnimationImage, 780
- step
  - touchgfx::Edge, 304
- stopAnimation
  - touchgfx::AnimatedImage, 120
  - touchgfx::ScrollBase, 622
- StringVisuals
  - touchgfx::LCD::StringVisuals, 676
- stringWidth
  - touchgfx::LCD, 410
- strlen
  - touchgfx::Unicode, 761
- strncmp
  - touchgfx::Unicode, 761
- strncmp\_ignore\_white\_spaces
  - touchgfx::Unicode, 762
- strncpy
  - touchgfx::Unicode, 762, 763
- SupportedTypesList
  - touchgfx::Partition, 529
- swapFrameBuffers
  - touchgfx::HAL, 358
- SwipeContainer, 677
- switchScreen



- touchgfx::Application, 133
- TYPED\_TEXT\_INVALID
  - touchgfx, 60
- takeFrameBufferSemaphore
  - touchgfx::OSWrappers, 492
- taskDelay
  - touchgfx::HAL, 358
  - touchgfx::OSWrappers, 492
- taskEntry
  - touchgfx::HALSDL2, 367
  - touchgfx::HAL, 358
- tearDown
  - touchgfx::CoverTransition, 232
  - touchgfx::SlideTransition, 669
  - touchgfx::Transition, 742
- tearDownScreen
  - touchgfx::Screen, 595
- TextArea, 681
  - touchgfx::TextArea, 683
- TextAreaWithOneWildcard, 689
  - touchgfx::TextAreaWithOneWildcard, 690
- TextAreaWithTwoWildcards, 692
  - touchgfx::TextAreaWithTwoWildcards, 693
- TextAreaWithWildcardBase, 695
  - touchgfx::TextAreaWithWildcardBase, 695
- TextButtonStyle< T >, 696
- TextDirection
  - touchgfx, 47
- TextProgress, 699
  - touchgfx::TextProgress, 700
- TextProvider, 703
  - touchgfx::TextProvider, 704
- TextRotation
  - touchgfx, 50
- Texts, 705
- TextureMapper, 706
- TextureSurface, 722
- tick
  - touchgfx::Gestures, 331
  - touchgfx::HAL, 358
- tickMoveDrawable
  - touchgfx::CoverTransition, 232
  - touchgfx::SlideTransition, 669
- TiledImage, 722
  - touchgfx::TiledImage, 723
- TiledImageButtonStyle< T >, 729
- to
  - touchgfx::CWRUtil::Q5, 548
- toQ10
  - touchgfx::CWRUtil, 238
- toQ5
  - touchgfx::CWRUtil, 239
- toUTF8
  - touchgfx::Unicode, 763
- toggle
  - touchgfx::GPIO, 335
  - touchgfx::LED, 445
- ToggleButton, 731
  - touchgfx::ToggleButton, 732
- ToggleButtonTrigger, 733
- top
  - touchgfx::GlyphNode, 332
- touch
  - touchgfx::HAL, 358
- TouchArea, 735
  - touchgfx::TouchArea, 736
- TouchButtonTrigger, 737
- TouchCalibration, 738
- TouchController, 739
- touchgfx, 35
  - abs, 50
  - Alignment, 47
  - BlitOperations, 47
  - ceil28\_4, 51
  - clz, 51
  - Direction, 48
  - DisplayOrientation, 48
  - DisplayRotation, 48
  - EasingEquation, 47
  - finalizeTransition, 51
  - fixed28\_4Mul, 52
  - fixed28\_4ToFloat, 52
  - floatToFixed16\_16, 52
  - floatToFixed28\_4, 52
  - floorDivMod, 53
  - FrameBuffer, 49
  - gcd, 53
  - GlyphFlags, 49
  - Gradient, 49
  - hw\_init, 53
  - LCD2getPixel, 54
  - LCD2setPixel, 54
  - LCD2shiftVal, 55
  - LCD4getPixel, 55
  - LCD4setPixel, 56
  - lookupBilinearRenderVariant, 56
  - lookupNearestNeighborRenderVariant, 57
  - makeTransition, 57
  - memset, 58
  - muldiv, 58
  - operator\*, 58, 59
  - prepareTransition, 59
  - RenderingVariant, 47
  - rotate0, 49
  - rotate90, 49
  - TYPED\_TEXT\_INVALID, 60
  - TextDirection, 47
  - TextRotation, 50
  - touchgfx\_generic\_init, 59
  - touchgfx\_init, 60
  - WideTextAction, 50
- touchgfx::AbstractButton
  - AbstractButton, 64
  - getPressedState, 64
  - getType, 64
  - handleClickEvent, 64

- setAction, 65
- touchgfx::AbstractButtonContainer
  - getAlpha, 66
  - getPressed, 66
  - setAction, 66
  - setAlpha, 67
  - setPressed, 67
- touchgfx::AbstractClock
  - ~AbstractClock, 68
  - AbstractClock, 68
  - getCurrentHour, 68
  - getCurrentMinute, 68
  - getCurrentSecond, 68
  - setTime12Hour, 69
  - setTime24Hour, 69
  - updateClock, 69
- touchgfx::AbstractDirectionProgress
  - ~AbstractDirectionProgress, 71
  - AbstractDirectionProgress, 70
  - DirectionType, 70
  - getDirection, 71
  - setDirection, 71
- touchgfx::AbstractPainter
  - ~AbstractPainter, 72
  - AbstractPainter, 72
  - render, 72
  - setOffset, 74
  - setWidgetAlpha, 74
- touchgfx::AbstractPainterBW
  - render, 75
  - renderInit, 76
  - renderNext, 76
- touchgfx::AbstractPainterGRAY2
  - render, 77
  - renderInit, 77
  - renderNext, 78
  - renderPixel, 78
- touchgfx::AbstractPainterGRAY4
  - render, 79
  - renderInit, 80
  - renderNext, 80
  - renderPixel, 80
- touchgfx::AbstractPainterRGB565
  - mixColors, 82
  - render, 82
  - renderInit, 83
  - renderNext, 83
  - renderPixel, 84
- touchgfx::AbstractPainterRGB888
  - render, 85
  - renderInit, 85
  - renderNext, 85
  - renderPixel, 86
- touchgfx::AbstractPartition
  - ~AbstractPartition, 87
  - AbstractPartition, 88
  - allocate, 88
  - allocateAt, 89
  - at, 89, 90
  - capacity, 90
  - clear, 90
  - dec, 91
  - element, 91
  - element\_size, 91
  - find, 92
  - getAllocationCount, 92
  - indexOf, 92
- touchgfx::AbstractProgressIndicator
  - ~AbstractProgressIndicator, 94
  - AbstractProgressIndicator, 94
  - getProgress, 94
  - getProgressIndicatorHeight, 95
  - getProgressIndicatorWidth, 95
  - getProgressIndicatorX, 95
  - getProgressIndicatorY, 95
  - getRange, 96
  - getValue, 97
  - setBackground, 97
  - setProgressIndicatorPosition, 97
  - setRange, 97
  - setValue, 98
- touchgfx::AbstractShape
  - ~AbstractShape, 100
  - AbstractShape, 100
  - drawCanvasWidget, 100
  - getAngle, 101
  - getCacheX, 101
  - getCacheY, 101
  - getCornerX, 101
  - getCornerY, 102
  - getMinimalRect, 102
  - getNumPoints, 102
  - getOrigin, 103
  - getScale, 103
  - moveOrigin, 103
  - setAngle, 104
  - setCache, 104
  - setCorner, 105
  - setOrigin, 105
  - setScale, 106
  - setShape, 107
  - updateAbstractShapeCache, 107
  - updateAngle, 107
- touchgfx::AnalogClock
  - animationEnabled, 110
  - convertHandValueToAngle, 110
  - getAnimationDuration, 110
  - getHourHandMinuteCorrection, 111
  - getMinuteHandSecondCorrection, 111
  - initializeTime12Hour, 111
  - initializeTime24Hour, 112
  - setAnimation, 112
  - setBackground, 112
  - setHourHandMinuteCorrection, 113
  - setMinuteHandSecondCorrection, 113
  - setRotationCenter, 113

- setupHand, 114
  - setupHourHand, 114
  - setupMinuteHand, 114
  - setupSecondHand, 115
  - updateClock, 115
- touchgfx::AnimatedImage
  - AnimatedImage, 117
  - getType, 117
  - handleTickEvent, 118
  - isAnimatedImageRunning, 118
  - isReverse, 118
  - isRunning, 118
  - pauseAnimation, 118
  - setBitmap, 119
  - setBitmaps, 119
  - setDoneAction, 119
  - setUpdateTicksInterval, 119
  - startAnimation, 120
  - stopAnimation, 120
- touchgfx::AnimatedImageButtonStyle
  - setBitmapXY, 121
  - setBitmaps, 121
  - setUpdateTicksInterval, 122
- touchgfx::AnimationTextureMapper
  - ~AnimationTextureMapper, 124
  - AnimationParameter, 124
  - AnimationTextureMapper, 124
  - cancelAnimationTextureMapperAnimation, 124
  - getAnimationStep, 124
  - handleTickEvent, 125
  - isTextureMapperAnimationRunning, 125
  - setTextureMapperAnimationEndedAction, 125
  - setTextureMapperAnimationStepAction, 125
  - setupAnimation, 126
  - startAnimation, 126
- touchgfx::Application
  - appSwitchScreen, 128
  - Application, 128
  - cacheDrawOperations, 129
  - clearAllTimerWidgets, 129
  - draw, 129
  - getCurrentScreen, 130
  - getInstance, 130
  - getNumberOfRegisteredTimerWidgets, 130
  - getTimerWidgetCountForDrawable, 130
  - handleClickEvent, 131
  - handleDragEvent, 131
  - handleGestureEvent, 131
  - handleKeyEvent, 131
  - handlePendingScreenTransition, 132
  - handleTickEvent, 132
  - instance, 133
  - invalidateArea, 132
  - MAX\_TIMER\_WIDGETS, 134
  - registerTimerWidget, 132
  - switchScreen, 133
  - unregisterTimerWidget, 133
- touchgfx::Bitmap
  - Bitmap, 136
  - BitmapFormat, 135
  - cache, 136
  - cacheAll, 137
  - cacheGetAddress, 137
  - cacheIsCached, 137
  - cacheRemoveBitmap, 138
  - cacheReplaceBitmap, 138
  - clearCache, 139
  - dynamicBitmapCreate, 139
  - dynamicBitmapDelete, 139
  - dynamicBitmapGetAddress, 140
  - dynamicBitmapSetSolidRect, 140
  - getAlphaData, 140
  - getData, 141
  - getFormat, 141
  - getHeight, 141
  - getId, 141
  - getRect, 141
  - getSolidRect, 142
  - getWidth, 142
  - hasTransparentPixels, 142
  - isAlphaPerPixel, 142
  - operator!=, 142
  - operator==, 143
  - registerBitmapDatabase, 143
  - setCache, 143
- touchgfx::BlitOp
  - operation, 145
- touchgfx::Box
  - ~Box, 147
  - Box, 146
  - draw, 147
  - forceReportAsSolid, 147
  - getAlpha, 147
  - getColor, 148
  - getSolidRect, 148
  - getType, 148
  - reportAsSolid, 149
  - setAlpha, 148
  - setColor, 149
- touchgfx::BoxProgress
  - ~BoxProgress, 150
  - BoxProgress, 150
  - getAlpha, 150
  - getColor, 150
  - setAlpha, 151
  - setColor, 151
  - setProgressIndicatorPosition, 151
  - setValue, 152
- touchgfx::BoxWithBorder
  - BoxWithBorder, 153
  - draw, 154
  - getAlpha, 154
  - getBorderColor, 154
  - getBorderSize, 154
  - getColor, 154
  - getSolidRect, 154

- getType, 155
  - setAlpha, 155
  - setBorderColor, 155
  - setBorderSize, 155
  - setColor, 156
- touchgfx::BoxWithBorderButtonStyle
  - setBorderSize, 157
  - setBoxWithBorderColors, 157
  - setBoxWithBorderHeight, 158
  - setBoxWithBorderPosition, 158
  - setBoxWithBorderWidth, 158
- touchgfx::Button
  - ~Button, 160
  - Button, 159
  - draw, 160
  - getAlpha, 160
  - getCurrentlyDisplayedBitmap, 160
  - getSolidRect, 160
  - getType, 161
  - setAlpha, 161
  - setBitmaps, 161
- touchgfx::ButtonController
  - ~ButtonController, 162
  - init, 162
  - reset, 162
  - sample, 162
- touchgfx::ButtonWithIcon
  - draw, 164
  - getCurrentlyDisplayedIcon, 165
  - getIconX, 165
  - getIconY, 165
  - getType, 165
  - setBitmaps, 165
  - setIconXY, 166
  - setIconX, 166
  - setIconY, 167
- touchgfx::ButtonWithLabel
  - ButtonWithLabel, 168
  - draw, 168
  - getLabelRotation, 168
  - getLabelText, 169
  - getSolidRect, 169
  - getType, 169
  - setLabelColor, 169
  - setLabelColorPressed, 170
  - setLabelRotation, 170
  - setLabelText, 170
  - updateTextPosition, 170
- touchgfx::Buttons
  - init, 163
  - sample, 163
- touchgfx::CWRUtil
  - angle, 234, 235
  - arcsine, 235
  - cosine, 236
  - mulQ5, 237
  - muldivQ5, 237
  - sine, 237, 238
  - sqrtQ10, 238
  - toQ10, 238
  - toQ5, 239
- touchgfx::CWRUtil::Q10
  - operator int, 540
  - operator\*, 540
  - operator+, 540
  - operator-, 541
  - operator/, 541
  - Q10, 539, 540
- touchgfx::CWRUtil::Q15
  - operator int, 542
  - operator+, 542
  - operator-, 543
  - operator/, 543
  - Q15, 542
- touchgfx::CWRUtil::Q5
  - operator int, 545
  - operator\*, 545, 546
  - operator+, 546
  - operator-, 546, 547
  - operator/, 547
  - Q5, 544
  - to, 548
- touchgfx::Callback
  - Callback, 172
  - execute, 172
  - isValid, 173
- touchgfx::Callback< dest\_type, T1, T2, void >
  - Callback, 174
  - execute, 174
  - isValid, 175
- touchgfx::Callback< dest\_type, T1, void, void >
  - Callback, 176
  - execute, 176
  - isValid, 176
- touchgfx::Callback< dest\_type, void, void, void >
  - Callback, 177, 178
  - execute, 178
  - isValid, 178
- touchgfx::Canvas
  - ~Canvas, 180
  - Canvas, 179
  - lineTo, 180
  - moveTo, 181
  - render, 181
- touchgfx::CanvasWidget
  - ~CanvasWidget, 183
  - CanvasWidget, 182
  - draw, 183
  - drawCanvasWidget, 183
  - getAlpha, 184
  - getMinimalRect, 184
  - getPainter, 184
  - getSolidRect, 184
  - invalidate, 185
  - setAlpha, 185
  - setPainter, 185

- touchgfx::CanvasWidgetRenderer
  - getMissingBufferSize, 187
  - getOutlineBuffer, 187
  - getOutlineBufferSize, 187
  - getScanlineCounts, 187
  - getScanlineCovers, 187
  - getScanlineStartIndices, 188
  - getScanlineWidth, 188
  - getUsedBufferSize, 188
  - getWriteMemoryUsageReport, 188
  - hasBuffer, 188
  - numCellsMissing, 189
  - numCellsUsed, 189
  - setScanlineWidth, 189
  - setWriteMemoryUsageReport, 190
  - setupBuffer, 189
- touchgfx::Cell
  - addCover, 191
  - packedCoord, 191
  - set, 192
  - setCoord, 192
  - setCover, 192
- touchgfx::Circle
  - Circle, 194
  - drawCanvasWidget, 194
  - getArc, 195
  - getArcEnd, 195, 196
  - getArcStart, 196
  - getCapPrecision, 197
  - getCenter, 197
  - getLineWidth, 197
  - getMinimalRect, 198
  - getPrecision, 199
  - getRadius, 199
  - setArc, 199, 200
  - setCapPrecision, 200
  - setCenter, 201
  - setCircle, 201, 202
  - setLineWidth, 202
  - setPrecision, 203
  - setRadius, 203
  - updateArcEnd, 204
  - updateArcStart, 204
- touchgfx::CircleProgress
  - ~CircleProgress, 206
  - CircleProgress, 206
  - getAlpha, 206
  - getCapPrecision, 206
  - getCenter, 207
  - getEndAngle, 207
  - getLineWidth, 207
  - getRadius, 207
  - getStartAngle, 208
  - setAlpha, 208
  - setCapPrecision, 208
  - setCenter, 209
  - setLineWidth, 209
  - setPainter, 209
  - setProgressIndicatorPosition, 209
  - setRadius, 210
  - setStartEndAngle, 210
  - setValue, 210
- touchgfx::ClickButtonTrigger
  - handleClickEvent, 211
- touchgfx::ClickEvent
  - ~ClickEvent, 214
  - ClickEvent, 214
  - ClickEventType, 212
  - getEventType, 214
  - getForce, 214
  - getType, 215
  - getX, 215
  - getY, 215
  - setType, 215
  - setX, 216
  - setY, 216
- touchgfx::ClickListener
  - ClickListener, 217
  - handleClickEvent, 217
  - setClickAction, 217
- touchgfx::Color
  - getBlueColor, 218
  - getColorFrom24BitRGB, 218
  - getGreenColor, 219
  - getRedColor, 219
- touchgfx::ConstFont
  - ConstFont, 222
  - find, 223
  - getGlyph, 223
  - getKerning, 223
  - getPixelData, 224
- touchgfx::Container
  - ~Container, 226
  - add, 226
  - Container, 226
  - contains, 226
  - draw, 226
  - forEachChild, 227
  - getContainedArea, 227
  - getFirstChild, 227
  - getLastChild, 227
  - getSolidRect, 228
  - getType, 228
  - insert, 228
  - moveChildrenRelative, 229
  - remove, 229
  - removeAll, 229
  - setupDrawChain, 230
  - unlink, 230
- touchgfx::CoverTransition
  - ~CoverTransition, 231
  - CoverTransition, 231
  - handleTickEvent, 232
  - init, 232
  - initMoveDrawable, 232
  - tearDown, 232

- tickMoveDrawable, 232
- touchgfx::CoverTransition::FullSolidRect
  - draw, 320
  - getSolidRect, 320
- touchgfx::DMA\_Interface
  - ~DMA\_Interface, 248
  - addToQueue, 249
  - DMA\_Interface, 248
  - disableAlpha, 249
  - enableAlpha, 249
  - enableCopyWithTransparentPixels, 249
  - execute, 249
  - executeCompleted, 250
  - flush, 250
  - getAllowed, 250
  - getBlitCaps, 250
  - initialize, 250
  - isDMARunning, 251
  - isDmaQueueEmpty, 250
  - isDmaQueueFull, 251
  - seedExecution, 251
  - setAllowed, 251
  - setupDataCopy, 251
  - setupDataFill, 252
  - signalDMAInterrupt, 252
  - start, 252
  - waitForFrameBufferSemaphore, 252
- touchgfx::DMA\_Queue
  - ~DMA\_Queue, 253
  - DMA\_Queue, 253
  - first, 253
  - isEmpty, 254
  - isFull, 254
  - pop, 254
  - pushCopyOf, 254
- touchgfx::DigitalClock
  - ~DigitalClock, 241
  - DigitalClock, 241
  - displayLeadingZeroForHourIndicator, 241
  - DisplayMode, 241
  - getAlpha, 241
  - getDisplayMode, 242
  - getTextWidth, 242
  - setAlpha, 242
  - setBaselineY, 242
  - setColor, 243
  - setDisplayMode, 243
  - setHeight, 243
  - setTypedText, 243
  - setWidth, 244
  - updateClock, 244
- touchgfx::DisplayTransformation
  - transformDisplayToFrameBuffer, 245, 246
  - transformFrameBufferToDisplay, 246
- touchgfx::DragEvent
  - ~DragEvent, 256
  - DragEvent, 256
  - getDeltaX, 256
  - getDeltaY, 256
  - getEventType, 257
  - getNewX, 257
  - getNewY, 257
  - getOldX, 257
  - getOldY, 257
  - getType, 258
- touchgfx::Draggable
  - ~Draggable, 259
  - Draggable, 258
  - handleDragEvent, 259
- touchgfx::Drawable
  - ~Drawable, 262
  - childGeometryChanged, 262
  - draw, 263
  - Drawable, 262
  - DrawableType, 262
  - getAbsoluteRect, 263
  - getCachedAbsX, 263
  - getCachedAbsY, 263
  - getCachedVisibleRect, 264
  - getHeight, 264
  - getLastChild, 264
  - getNextSibling, 265
  - getParent, 265
  - getRect, 265
  - getSolidRect, 265
  - getSolidRectAbsolute, 265
  - getType, 266
  - getVisibleRect, 266
  - getWidth, 266
  - getX, 267
  - getY, 267
  - handleClickEvent, 267
  - handleDragEvent, 267
  - handleGestureEvent, 268
  - handleTickEvent, 268
  - invalidate, 268
  - invalidateRect, 268
  - isTouchable, 269
  - isVisible, 269
  - moveRelative, 269
  - moveTo, 270
  - resetDrawChainCache, 270
  - setHeight, 270
  - setPosition, 270
  - setTouchable, 271
  - setVisible, 272
  - setWidth, 272
  - setXY, 273
  - setupDrawChain, 271
  - setX, 272
  - setY, 273
  - translateRectToAbsolute, 273
- touchgfx::DrawableList
  - ~DrawableList, 275
  - DrawableList, 275
  - getCircular, 275

- getDrawableIndex, 275
- getDrawableIndices, 276
- getDrawableMargin, 276
- getDrawableSize, 276
- getHorizontal, 277
- getItemIndex, 277
- getItemSize, 277
- getNumberOfDrawables, 278
- getNumberOfItems, 278
- getOffset, 278
- getRequiredNumberOfDrawables, 278
- itemChanged, 279
- refreshDrawables, 279
- setCircular, 279
- setDrawableSize, 280
- setDrawables, 280
- setHeight, 280
- setHorizontal, 281
- setNumberOfItems, 281
- setOffset, 281
- setWidth, 282
- touchgfx::DrawableListItems
  - ~DrawableListItems, 283
  - DrawableListItems, 283
  - getDrawable, 283
  - getNumberOfDrawables, 284
  - operator[], 284
- touchgfx::DrawableListItemsInterface
  - ~DrawableListItemsInterface, 285
  - getDrawable, 285
  - getNumberOfDrawables, 285
- touchgfx::EasingEquations
  - backEaseIn, 288
  - backEaseInOut, 288
  - backEaseOut, 289
  - bounceEaseIn, 289
  - bounceEaseInOut, 290
  - bounceEaseOut, 290
  - circEaseIn, 291
  - circEaseInOut, 291
  - circEaseOut, 291
  - cubicEaseIn, 292
  - cubicEaseInOut, 292
  - cubicEaseOut, 293
  - elasticEaseIn, 293
  - elasticEaseInOut, 294
  - elasticEaseOut, 294
  - expoEaseIn, 294
  - expoEaseInOut, 295
  - expoEaseOut, 295
  - linearEaseIn, 296
  - linearEaseInOut, 296
  - linearEaseNone, 297
  - linearEaseOut, 297
  - quadEaseIn, 297
  - quadEaseInOut, 298
  - quadEaseOut, 298
  - quartEaseIn, 299
  - quartEaseInOut, 299
  - quartEaseOut, 300
  - quintEaseIn, 300
  - quintEaseInOut, 300
  - quintEaseOut, 301
  - sineEaseIn, 301
  - sineEaseInOut, 302
  - sineEaseOut, 302
- touchgfx::Edge
  - Edge, 304
  - step, 304
- touchgfx::Event
  - ~Event, 305
  - EventType, 305
  - getEventType, 306
- touchgfx::FadeAnimator
  - ~FadeAnimator, 307
  - clearFadeAnimationEndedAction, 308
  - FadeAnimator, 307
  - getFadeAnimationDelay, 308
  - handleTickEvent, 308
  - isFadeAnimationRunning, 308
  - isRunning, 308
  - nextFadeAnimationStep, 308
  - setFadeAnimationDelay, 309
  - setFadeAnimationEndedAction, 309
  - startFadeAnimation, 309
- touchgfx::Font
  - Font, 311
  - getBitsPerPixel, 312
  - getCharWidth, 312
  - getEllipsisChar, 312
  - getFallbackChar, 313
  - getFontHeight, 313
  - getGlyph, 313
  - getKerning, 314
  - getMaxPixelsLeft, 314
  - getMaxPixelsRight, 314
  - getMaxTextHeight, 315
  - getMinimumTextHeight, 315
  - getNumberOfLines, 315
  - getSpacingAbove, 315
  - getStringWidth, 316
  - getStringWidthLTR, 317
  - getStringWidthRTL, 317
- touchgfx::FontManager
  - getFont, 318
  - setFontProvider, 318
- touchgfx::FontProvider
  - ~FontProvider, 319
  - getFont, 319
- touchgfx::GPIO
  - clear, 334
  - GPIO\_ID, 333
  - get, 334
  - init, 334
  - set, 334
  - toggle, 335



- touchgfx::GenericCallback
  - ~GenericCallback, [321](#)
  - execute, [321](#)
  - isValid, [322](#)
- touchgfx::GenericCallback< T1, T2, void >
  - ~GenericCallback, [323](#)
  - execute, [323](#)
  - isValid, [324](#)
- touchgfx::GenericCallback< T1, void, void >
  - ~GenericCallback, [325](#)
  - execute, [325](#)
  - isValid, [325](#)
- touchgfx::GenericCallback< void >
  - ~GenericCallback, [326](#)
  - execute, [326](#)
  - isValid, [326](#)
- touchgfx::GestureEvent
  - GestureEvent, [328](#)
  - GestureType, [327](#)
  - getEventType, [328](#)
  - getType, [328](#)
  - getVelocity, [328](#)
  - getX, [329](#)
  - getY, [329](#)
- touchgfx::Gestures
  - Gestures, [330](#)
  - registerClickEvent, [330](#)
  - registerDragEvent, [330](#)
  - registerEventListener, [330](#)
  - setDragThreshold, [331](#)
  - tick, [331](#)
- touchgfx::GlyphNode
  - advance, [332](#)
  - height, [332](#)
  - top, [332](#)
  - width, [333](#)
- touchgfx::Gradients
  - Gradients, [336](#)
- touchgfx::HALSDL2
  - blitSetTransparencyKey, [361](#)
  - blockCopy, [361](#)
  - configureInterrupts, [362](#)
  - configureLCDInterrupt, [362](#)
  - copyScreenshotToClipboard, [362](#)
  - disableInterrupts, [362](#)
  - doSampleTouch, [362](#)
  - enableInterrupts, [363](#)
  - enableLCDControllerInterrupt, [363](#)
  - flushFrameBuffer, [363](#)
  - getArgv, [363](#)
  - getTFTFrameBuffer, [364](#)
  - getWindowTitle, [364](#)
  - HALSDL2, [361](#)
  - loadSkin, [364](#)
  - performDisplayOrientationChange, [365](#)
  - renderLCD\_FrameBufferToMemory, [365](#)
  - sampleKey, [365](#)
  - saveNextScreenshots, [365](#)
  - saveScreenshot, [366](#)
  - sdl\_init, [366](#)
  - setTFTFrameBuffer, [366](#)
  - setVsyncInterval, [367](#)
  - setWindowTitle, [367](#)
  - taskEntry, [367](#)
- touchgfx::HAL
  - ~HAL, [341](#)
  - allowDMATransfers, [341](#)
  - backPorchExited, [342](#)
  - beginFrame, [342](#)
  - blitCopy, [342](#)
  - blitCopyARGB8888, [342](#)
  - blitCopyGlyph, [343](#)
  - blitFill, [344](#)
  - blitSetTransparencyKey, [344](#)
  - blockCopy, [344](#)
  - cacheTextString, [345](#)
  - configureInterrupts, [345](#)
  - copyFBRegionToMemory, [345](#), [346](#)
  - disableInterrupts, [346](#)
  - enableInterrupts, [346](#)
  - enableLCDControllerInterrupt, [346](#)
  - enableMCULoadCalculation, [347](#)
  - endFrame, [347](#)
  - flushDMA, [347](#)
  - flushFrameBuffer, [347](#)
  - FrameRefreshStrategy, [341](#)
  - frontPorchEntered, [348](#)
  - getAnimationStorage, [348](#)
  - getBlitCaps, [348](#)
  - getButtonController, [348](#)
  - getCPUCycles, [348](#)
  - getClientFrameBuffer, [348](#)
  - getDisplayHeight, [349](#)
  - getDisplayOrientation, [349](#)
  - getDisplayWidth, [349](#)
  - getFingerSize, [349](#)
  - getFrameRefreshStrategy, [349](#)
  - getInstance, [350](#)
  - getLCDRefreshCount, [350](#)
  - getMCULoadPct, [350](#)
  - getTFTCurrentLine, [350](#)
  - getTFTFrameBuffer, [351](#)
  - getTouchSampleRate, [351](#)
  - HAL, [341](#)
  - initialize, [351](#)
  - lcd, [351](#)
  - lockDMAToFrontPorch, [351](#)
  - lockFrameBuffer, [352](#)
  - noTouch, [352](#)
  - performDisplayOrientationChange, [352](#)
  - registerEventListener, [352](#)
  - registerTaskDelayFunction, [353](#)
  - registerTextCache, [353](#)
  - sampleKey, [353](#)
  - setButtonController, [354](#)
  - setDisplayOrientation, [354](#)



- setDragThreshold, 354
- setFingerSize, 355
- setFrameBufferStartAddress, 355
- setFrameBufferStartAddresses, 355
- setFrameRateCompensation, 356
- setFrameRefreshStrategy, 356
- setMCUActive, 357
- setMCUInstrumentation, 357
- setTFTFrameBuffer, 357
- setTouchSampleRate, 357
- signalDMAInterrupt, 358
- swapFrameBuffers, 358
- taskDelay, 358
- taskEntry, 358
- tick, 358
- touch, 358
- unlockFrameBuffer, 359
- vSync, 359
- touchgfx::I2CTouchController
  - ~I2CTouchController, 370
  - I2CTouchController, 370
  - init, 371
  - sampleTouch, 371
- touchgfx::I2C
  - ~I2C, 368
  - I2C, 368
  - init, 369
  - readRegister, 369
  - writeRegister, 369
- touchgfx::IconButtonStyle
  - getCurrentlyDisplayedIcon, 373
  - getIconX, 373
  - getIconY, 373
  - setIconBitmaps, 373
  - setIconXY, 374
  - setIconX, 373
  - setIconY, 374
- touchgfx::Image
  - draw, 375
  - getAlpha, 376
  - getBitmap, 376
  - getSolidRect, 376
  - getType, 376
  - Image, 375
  - setAlpha, 377
  - setBitmap, 377
- touchgfx::ImageButtonStyle
  - getCurrentlyDisplayedBitmap, 378
  - setBitmapXY, 379
  - setBitmaps, 378
- touchgfx::ImageProgress
  - ~ImageProgress, 380
  - getAlpha, 380
  - getAnchorAtZero, 380
  - getBitmap, 381
  - ImageProgress, 380
  - setAlpha, 381
  - setAnchorAtZero, 381
  - setBitmap, 382
  - setProgressIndicatorPosition, 382
  - setValue, 382
- touchgfx::InternalFlashFont
  - getKerning, 384
  - getPixelData, 384
  - InternalFlashFont, 383
- touchgfx::JSMOCHelper
  - draw, 387
  - getCachedAbsX, 387
  - getCachedAbsY, 387
  - getCachedVisibleRect, 388
  - getHeight, 388
  - getWidth, 388
  - JSMOCHelper, 387
  - setWidth, 388
- touchgfx::Keyboard
  - ~Keyboard, 392
  - draw, 392
  - getBuffer, 392
  - getBufferPosition, 392
  - getCallbackAreaForCoordinates, 393
  - getCharForKey, 393
  - getKeyForCoordinates, 393
  - getKeyMappingList, 394
  - getLayout, 394
  - getType, 394
  - handleClickEvent, 394
  - handleDragEvent, 395
  - Keyboard, 392
  - setBuffer, 395
  - setBufferPosition, 395
  - setKeyListener, 395
  - setKeymappingList, 396
  - setLayout, 396
  - setTextIndentation, 396
  - setupDrawChain, 396
- touchgfx::LCD16bpp
  - bitDepth, 412
  - blitCopy, 412, 413
  - blitCopyARGB8888, 414
  - blitCopyAlphaPerPixel, 413
  - copyFrameBufferRegionToMemory, 414
  - drawGlyph, 415
  - drawPartialBitmap, 415
  - drawTextureMapScanLine, 416
  - fillRect, 416
  - init, 417
  - nextLine, 417
  - nextPixel, 417
- touchgfx::LCD1bpp
  - bitDepth, 419
  - blitCopy, 419, 420
  - blitCopyRLE, 420
  - copyFrameBufferRegionToMemory, 421
  - drawGlyph, 421
  - drawPartialBitmap, 422

- drawTextureMapScanLine, 422
- fillMemory, 423
- fillRect, 423
- nextLine, 423
- nextPixel, 424
- touchgfx::LCD24bpp
  - bitDepth, 426
  - blitCopy, 426
  - blitCopyARGB8888, 427
  - copyFrameBufferRegionToMemory, 427
  - drawGlyph, 427
  - drawPartialBitmap, 428
  - drawTextureMapScanLine, 429
  - fillRect, 429
  - init, 430
  - nextLine, 430
  - nextPixel, 430
- touchgfx::LCD2bpp
  - bitDepth, 432
  - blitCopy, 432, 433
  - blitCopyAlphaPerPixel, 433
  - copyFrameBufferRegionToMemory, 434
  - drawGlyph, 434
  - drawPartialBitmap, 435
  - drawTextureMapScanLine, 435
  - fillRect, 436
  - init, 436
  - nextLine, 436
  - nextPixel, 437
- touchgfx::LCD4bpp
  - bitDepth, 439
  - blitCopy, 439
  - blitCopyAlphaPerPixel, 440
  - copyFrameBufferRegionToMemory, 440
  - drawGlyph, 441
  - drawPartialBitmap, 441
  - drawTextureMapScanLine, 442
  - fillRect, 442
  - init, 443
  - nextLine, 443
  - nextPixel, 443
- touchgfx::LCD::StringVisuals
  - StringVisuals, 676
- touchgfx::LCD
  - ~LCD, 400
  - bitDepth, 400
  - blitCopy, 400, 401
  - copyFrameBufferRegionToMemory, 401
  - drawBorder, 402
  - drawGlyph, 402, 403
  - drawHorizontalLine, 404
  - drawPartialBitmap, 404
  - drawRect, 405
  - drawString, 405
  - drawStringLTR, 405
  - drawStringRTL, 406
  - drawTextureMapScanLine, 407
  - drawTextureMapTriangle, 407
  - drawVerticalLine, 408
  - fillRect, 408
  - getNumLines, 409
  - init, 409
  - realX, 409
  - realY, 410
  - rotateRect, 410
  - stringWidth, 410
- touchgfx::LED
  - get, 444
  - init, 444
  - off, 445
  - on, 445
  - toggle, 445
- touchgfx::Line
  - drawCanvasWidget, 448
  - getEnd, 448
  - getLineEndingStyle, 448
  - getLineWidth, 449
  - getMinimalRect, 449
  - getStart, 450
  - LINE\_ENDING\_STYLE, 447
  - Line, 447
  - setCapPrecision, 450
  - setEnd, 451
  - setLine, 451
  - setLineEndingStyle, 452
  - setLineWidth, 452, 453
  - setStart, 453, 454
  - updateEnd, 454, 455
  - updateLineWidth, 455, 456
  - updateStart, 456
- touchgfx::LineProgress
  - ~LineProgress, 458
  - getAlpha, 458
  - getEnd, 459
  - getLineEndingStyle, 459
  - getLineWidth, 459
  - getStart, 459
  - LineProgress, 458
  - setAlpha, 460
  - setEnd, 460
  - setLineEndingStyle, 460
  - setLineWidth, 461
  - setPainter, 461
  - setProgressIndicatorPosition, 461
  - setStart, 462
  - setValue, 462
- touchgfx::ListLayout
  - ~ListLayout, 463
  - add, 464
  - getDirection, 464
  - getType, 464
  - insert, 464
  - ListLayout, 463
  - remove, 465
  - removeAll, 465
  - setDirection, 465

- touchgfx::LockFreeDMA\_Queue
  - first, [467](#)
  - isEmpty, [467](#)
  - isFull, [467](#)
  - LockFreeDMA\_Queue, [466](#)
  - pop, [467](#)
  - pushCopyOf, [467](#)
- touchgfx::MCUInstrumentation
  - ~MCUInstrumentation, [474](#)
  - getCCConsumed, [474](#)
  - getCPUCycles, [474](#)
  - getElapsedUS, [474](#)
  - init, [475](#)
  - MCUInstrumentation, [473](#)
  - setCCConsumed, [475](#)
  - setMCUActive, [475](#)
- touchgfx::MVPApplication
  - ~MVPApplication, [485](#)
  - evaluatePendingScreenTransition, [485](#)
  - handlePendingScreenTransition, [485](#)
  - MVPApplication, [485](#)
- touchgfx::MVPHeap
  - ~MVPHeap, [486](#)
  - MVPHeap, [486](#)
- touchgfx::Matrix4x4
  - concatenateXRotation, [469](#)
  - concatenateXScale, [469](#)
  - concatenateXTranslation, [469](#)
  - concatenateYRotation, [470](#)
  - concatenateYScale, [470](#)
  - concatenateYTranslation, [470](#)
  - concatenateZRotation, [471](#)
  - concatenateZScale, [471](#)
  - concatenateZTranslation, [471](#)
  - getElement, [472](#)
  - identity, [472](#)
  - Matrix4x4, [469](#)
  - setElement, [472](#)
  - setViewDistance, [472](#)
- touchgfx::ModalWindow
  - ~ModalWindow, [477](#)
  - add, [477](#)
  - getBackgroundHeight, [477](#)
  - getBackgroundWidth, [477](#)
  - getShadeAlpha, [477](#)
  - getShadeColor, [478](#)
  - hide, [478](#)
  - isShowing, [478](#)
  - ModalWindow, [476](#)
  - remove, [478](#)
  - setBackground, [478](#), [479](#)
  - setShadeAlpha, [479](#)
  - setShadeColor, [479](#)
  - show, [480](#)
- touchgfx::MoveAnimator
  - ~MoveAnimator, [481](#)
  - cancelMoveAnimation, [482](#)
  - clearMoveAnimationEndedAction, [482](#)
  - getMoveAnimationDelay, [482](#)
  - handleTickEvent, [482](#)
  - isMoveAnimationRunning, [482](#)
  - isRunning, [482](#)
  - MoveAnimator, [481](#)
  - nextMoveAnimationStep, [483](#)
  - setMoveAnimationDelay, [483](#)
  - setMoveAnimationEndedAction, [483](#)
  - startMoveAnimation, [483](#)
- touchgfx::NoDMA
  - flush, [487](#)
  - getBlitCaps, [487](#)
  - NoDMA, [487](#)
  - setupDataCopy, [488](#)
  - setupDataFill, [488](#)
  - signalDMAInterrupt, [488](#)
- touchgfx::NoTouchController
  - ~NoTouchController, [489](#)
  - init, [489](#)
  - sampleTouch, [489](#)
- touchgfx::NoTransition
  - ~NoTransition, [490](#)
  - handleTickEvent, [490](#)
  - NoTransition, [490](#)
- touchgfx::OSWrappers
  - giveFrameBufferSemaphore, [491](#)
  - giveFrameBufferSemaphoreFromISR, [491](#)
  - initialize, [491](#)
  - signalVSync, [491](#)
  - takeFrameBufferSemaphore, [492](#)
  - taskDelay, [492](#)
  - tryTakeFrameBufferSemaphore, [492](#)
  - waitForVSync, [492](#)
- touchgfx::Outline
  - ~Outline, [494](#)
  - getCells, [494](#)
  - getNumCells, [494](#)
  - lineTo, [494](#)
  - moveTo, [495](#)
  - Outline, [494](#)
  - OutlineFlags\_t, [494](#)
  - reset, [495](#)
  - setMaxRenderY, [495](#)
  - wasOutlineTooComplex, [495](#)
- touchgfx::PainterBWBitmap
  - PainterBWBitmap, [499](#)
  - render, [499](#)
  - renderInit, [500](#)
  - renderNext, [500](#)
  - setBitmap, [500](#)
- touchgfx::PainterBW
  - bw, [496](#)
  - getColor, [497](#)
  - render, [497](#)
  - renderNext, [498](#)
  - setColor, [498](#)
- touchgfx::PainterGRAY2
  - getAlpha, [502](#)

- getColor, [502](#)
- PainterGRAY2, [501](#)
- render, [502](#)
- renderNext, [503](#)
- setAlpha, [503](#)
- setColor, [503](#)
- touchgfx::PainterGRAY2Bitmap
  - getAlpha, [505](#)
  - PainterGRAY2Bitmap, [505](#)
  - render, [505](#)
  - renderInit, [506](#)
  - renderNext, [506](#)
  - setAlpha, [506](#)
  - setBitmap, [507](#)
- touchgfx::PainterGRAY4
  - getAlpha, [508](#)
  - getColor, [508](#)
  - PainterGRAY4, [508](#)
  - render, [509](#)
  - renderNext, [509](#)
  - setAlpha, [510](#)
  - setColor, [510](#)
- touchgfx::PainterGRAY4Bitmap
  - getAlpha, [511](#)
  - PainterGRAY4Bitmap, [511](#)
  - render, [512](#)
  - renderInit, [512](#)
  - renderNext, [512](#)
  - setAlpha, [513](#)
  - setBitmap, [513](#)
- touchgfx::PainterRGB565
  - getAlpha, [515](#)
  - getColor, [515](#)
  - PainterRGB565, [514](#)
  - render, [515](#)
  - renderNext, [516](#)
  - setAlpha, [516](#)
  - setColor, [516](#)
- touchgfx::PainterRGB565Bitmap
  - getAlpha, [518](#)
  - PainterRGB565Bitmap, [518](#)
  - render, [518](#)
  - renderInit, [519](#)
  - renderNext, [519](#)
  - setAlpha, [520](#)
  - setBitmap, [520](#)
- touchgfx::PainterRGB888
  - getAlpha, [521](#)
  - getColor, [522](#)
  - PainterRGB888, [521](#)
  - render, [522](#)
  - renderNext, [522](#)
  - setAlpha, [523](#)
  - setColor, [523](#)
- touchgfx::PainterRGB888Bitmap
  - getAlpha, [525](#)
  - PainterRGB888Bitmap, [524](#)
  - render, [525](#)
  - renderInit, [525](#)
  - renderNext, [526](#)
  - setAlpha, [526](#)
  - setBitmap, [526](#)
- touchgfx::Pair
  - Pair, [527](#), [528](#)
- touchgfx::Partition
  - capacity, [530](#)
  - element, [530](#)
  - element\_size, [531](#)
  - Partition, [530](#)
  - SupportedTypesList, [529](#)
- touchgfx::PixelDataWidget
  - draw, [532](#)
  - getAlpha, [532](#)
  - getSolidRect, [533](#)
  - PixelDataWidget, [532](#)
  - setAlpha, [533](#)
  - setBitmapFormat, [533](#)
  - setPixelData, [533](#)
- touchgfx::Point
  - dist\_sqr, [534](#)
- touchgfx::Point4
  - Point4, [535](#)
- touchgfx::PreRenderable
  - draw, [537](#)
  - isPreRendered, [537](#)
  - preRender, [537](#)
  - PreRenderable, [537](#)
  - setupDrawChain, [537](#)
- touchgfx::Presenter
  - ~Presenter, [538](#)
  - activate, [538](#)
  - deactivate, [539](#)
  - Presenter, [538](#)
- touchgfx::Quadruple
  - getElement, [549](#)
  - getW, [550](#)
  - getX, [550](#)
  - getY, [550](#)
  - getZ, [550](#)
  - Quadruple, [549](#)
  - setElement, [550](#)
  - setW, [551](#)
  - setX, [551](#)
  - setY, [551](#)
  - setZ, [551](#)
- touchgfx::RadioButton
  - ~RadioButton, [553](#)
  - draw, [553](#)
  - getAlpha, [554](#)
  - getCurrentlyDisplayedBitmap, [554](#)
  - getDeselectionEnabled, [554](#)
  - getSelected, [554](#)
  - getSolidRect, [555](#)
  - getType, [555](#)
  - handleClickEvent, [555](#)
  - RadioButton, [553](#)

- setAlpha, 555
- setBitmaps, 556
- setDeselectedAction, 556
- setDeselectionEnabled, 556
- setSelected, 558
- touchgfx::RadioButtonGroup
  - ~RadioButtonGroup, 559
  - add, 560
  - getDeselectionEnabled, 560
  - getRadioButton, 560
  - getSelectedRadioButton, 560
  - getSelectedRadioButtonIndex, 561
  - radioButtonClickedHandler, 561
  - radioButtonDeselectedHandler, 561
  - RadioButtonGroup, 559
  - setDeselectionEnabled, 561
  - setRadioButtonDeselectedHandler, 562
  - setRadioButtonSelectedHandler, 562
  - setSelected, 562
- touchgfx::Rasterizer
  - calculateAlpha, 565
  - FillingRule, 564
  - lineTo, 565
  - moveTo, 565
  - Rasterizer, 565
  - render, 566
  - reset, 566
  - setFillingRule, 566
  - setMaxRenderY, 566
  - wasOutlineTooComplex, 567
- touchgfx::Rect
  - area, 568
  - bottom, 569
  - expandToFit, 569
  - includes, 569
  - intersect, 569, 570
  - isEmpty, 570
  - operator!=, 570
  - operator==, 571
  - operator&, 571
  - operator&=, 571
  - Rect, 568
  - right, 571
- touchgfx::Renderer
  - getRenderingBuffer, 573
  - render, 573
  - Renderer, 572
  - setRenderingBuffer, 573
- touchgfx::RenderingBuffer
  - ~RenderingBuffer, 575
  - attach, 575
  - getHeight, 576
  - getWidth, 576
  - getXAdjust, 576
  - inbox, 576
  - RenderingBuffer, 575
  - row, 577
- touchgfx::RepeatButton
  - getDelay, 578
  - getInterval, 579
  - handleClickEvent, 579
  - handleTickEvent, 579
  - RepeatButton, 578
  - setDelay, 579
  - setInterval, 580
- touchgfx::RepeatButtonTrigger
  - getDelay, 581
  - getInterval, 581
  - handleClickEvent, 581
  - setDelay, 581
  - setInterval, 581
- touchgfx::SDL2TouchController
  - init, 641
  - sampleTouch, 642
- touchgfx::SDLTouchController
  - init, 642
  - sampleTouch, 643
- touchgfx::ScalableImage
  - ~ScalableImage, 583
  - draw, 584
  - drawTriangle, 584
  - getAlpha, 584
  - getBitmap, 585
  - getScalingAlgorithm, 585
  - getSolidRect, 585
  - getType, 585
  - lookupRenderVariant, 585
  - ScalableImage, 583
  - ScalingAlgorithm, 583
  - setAlpha, 586
  - setBitmap, 586
  - setScalingAlgorithm, 586
- touchgfx::Scanline
  - ~Scanline, 588
  - addCell, 588
  - addSpan, 589
  - getNumSpans, 589
  - getY, 589
  - isReady, 589
  - reset, 590
  - resetSpans, 590
  - Scanline, 588
- touchgfx::Scanline::iterator
  - getCovers, 386
  - getNumPix, 386
  - iterator, 385
  - next, 386
- touchgfx::Screen
  - ~Screen, 591
  - add, 592
  - afterTransition, 592
  - bindTransition, 592
  - draw, 592, 593
  - getRootContainer, 593
  - handleClickEvent, 593
  - handleDragEvent, 593

- handleGestureEvent, 594
- handleKeyEvent, 594
- handleTickEvent, 594
- JSMOC, 594
- remove, 594
- Screen, 591
- setupScreen, 595
- startSMOC, 595
- tearDownScreen, 595
- useSMOCDrawing, 595
- usingSMOC, 596
- touchgfx::ScrollBase
  - ~ScrollBase, 610
  - allowHorizontalDrag, 611
  - allowVerticalDrag, 611
  - animateToItem, 611
  - animateToPosition, 612
  - AnimationState, 610
  - getAnimationSteps, 612
  - getCircular, 612
  - getDragAcceleration, 612
  - getDrawableMargin, 613
  - getDrawableSize, 613
  - getHorizontal, 613
  - getMaxSwipeItems, 613
  - getNearestAlignedOffset, 614
  - getNormalizedOffset, 614
  - getNumberOfItems, 614
  - getOffset, 615
  - getPositionForItem, 615
  - getSwipeAcceleration, 615
  - handleDragEvent, 616
  - handleGestureEvent, 616
  - handleTickEvent, 616
  - initialize, 616
  - isAnimating, 616
  - itemChanged, 617
  - keepOffsetInsideLimits, 617
  - ScrollBase, 610
  - setAnimationEndedCallback, 617
  - setAnimationSteps, 618
  - setCircular, 618
  - setDragAcceleration, 618
  - setDrawableSize, 619
  - setEasingEquation, 619
  - setHeight, 619
  - setHorizontal, 620
  - setItemPressedCallback, 620
  - setItemSelectedCallback, 620
  - setMaxSwipeItems, 620
  - setNumberOfItems, 621
  - setOffset, 621
  - setSwipeAcceleration, 621
  - setWidth, 622
  - stopAnimation, 622
- touchgfx::ScrollList
  - ~ScrollList, 624
  - getItem, 624
  - getNearestAlignedOffset, 624
  - getPaddingAfter, 624
  - getPaddingBefore, 625
  - getPositionForItem, 625
  - getSnapping, 625
  - handleClickEvent, 625
  - keepOffsetInsideLimits, 626
  - ScrollList, 623
  - setDrawables, 626
  - setPadding, 626
  - setSnapping, 627
  - setWindowSize, 627
- touchgfx::ScrollWheel
  - ~ScrollWheel, 628
  - ScrollWheel, 628
  - setDrawables, 628
- touchgfx::ScrollWheelBase
  - ~ScrollWheelBase, 630
  - animateToPosition, 630
  - getPositionForItem, 630
  - getSelectedItem, 631
  - getSelectedItemOffset, 631
  - handleClickEvent, 631
  - handleDragEvent, 632
  - handleGestureEvent, 632
  - keepOffsetInsideLimits, 632
  - ScrollWheelBase, 630
  - setAnimateToCallback, 632
  - setSelectedItemOffset, 633
- touchgfx::ScrollWheelWithSelectionStyle
  - ~ScrollWheelWithSelectionStyle, 635
  - getSelectedItemExtraSizeAfter, 635
  - getSelectedItemExtraSizeBefore, 635
  - getSelectedItemMarginAfter, 636
  - getSelectedItemMarginBefore, 636
  - initialize, 636
  - itemChanged, 636
  - refreshDrawableListsLayout, 637
  - ScrollWheelWithSelectionStyle, 635
  - setCircular, 637
  - setDrawableSize, 638
  - setDrawables, 637
  - setHeight, 638
  - setHorizontal, 638
  - setNumberOfItems, 639
  - setOffset, 639
  - setSelectedItemOffset, 641
  - setSelectedItemExtraSize, 639
  - setSelectedItemMargin, 640
  - setSelectedItemPosition, 640
  - setWidth, 641
- touchgfx::ScrollableContainer
  - ~ScrollableContainer, 599
  - add, 599
  - childGeometryChanged, 600
  - doScroll, 600
  - enableHorizontalScroll, 600
  - enableVerticalScroll, 601

- getContainedArea, 601
- getLastChild, 601
- getScrolledX, 601
- getScrolledY, 602
- getType, 602
- getXBorder, 602
- getXScrollbar, 602
- getYBorder, 603
- getYScrollbar, 603
- handleClickEvent, 603
- handleDragEvent, 603
- handleGestureEvent, 604
- handleTickEvent, 604
- invalidateScrollbars, 604
- isScrollableXY, 604
- moveChildrenRelative, 604
- reset, 605
- ScrollableContainer, 599
- setMaxVelocity, 605
- setScrollThreshold, 606
- setScrollbarPadding, 605
- setScrollbarWidth, 606
- setScrollbarsAlpha, 605
- setScrollbarsColor, 606
- setScrollbarsPermanentlyVisible, 606
- setScrollbarsVisible, 606
- touchgfx::Shape
  - ~Shape, 644
  - getCacheX, 644
  - getCacheY, 645
  - getCornerX, 645
  - getCornerY, 645
  - getNumPoints, 646
  - setCache, 646
  - setCorner, 646
- touchgfx::SlideMenu
  - add, 650
  - animateToState, 650
  - animationEndedHandler, 650
  - getAnimationDuration, 651
  - getAnimationEasingEquation, 651
  - getBackgroundX, 651
  - getBackgroundY, 651
  - getCollapsedXCoordinate, 651
  - getCollapsedYCoordinate, 652
  - getExpandDirection, 652
  - getExpandedStateTimeout, 652
  - getExpandedStateTimer, 652
  - getExpandedXCoordinate, 652
  - getExpandedYCoordinate, 652
  - getHiddenPixelsWhenExpanded, 653
  - getState, 653
  - getStateChangeButtonX, 653
  - getStateChangeButtonY, 653
  - getVisiblePixelsWhenCollapsed, 653
  - remove, 653
  - resetExpandedStateTimer, 654
  - setAnimationDuration, 654
  - setAnimationEasingEquation, 654
  - setExpandDirection, 654
  - setExpandedStateTimeout, 654
  - setHiddenPixelsWhenExpanded, 655
  - setState, 655
  - setStateChangedAnimationEndedCallback, 655
  - setStateChangedCallback, 655
  - setVisiblePixelsWhenCollapsed, 657
  - setup, 656
  - stateChangeButtonClickedHandler, 657
- touchgfx::SlideTransition
  - ~SlideTransition, 668
  - handleTickEvent, 668
  - init, 669
  - initMoveDrawable, 669
  - SlideTransition, 668
  - tearDown, 669
  - tickMoveDrawable, 669
- touchgfx::Slider
  - getIndicatorMax, 660
  - getIndicatorMin, 660
  - getIndicatorPositionRangeSize, 660
  - getIndicatorRadius, 660
  - getMaxValue, 660
  - getMinValue, 661
  - getType, 661
  - getValue, 661
  - getValueRangeSize, 661
  - handleClickEvent, 662
  - handleDragEvent, 662
  - positionToValue, 662
  - setBitmaps, 662, 663
  - setNewValueCallback, 663
  - setStartValueCallback, 664
  - setStopValueCallback, 664
  - setValue, 665
  - setValueRange, 666
  - setupHorizontalSlider, 664
  - setupVerticalSlider, 665
  - Slider, 659
  - updateIndicatorPosition, 666
  - valueToPosition, 667
- touchgfx::Snapper
  - ~Snapper, 671
  - handleClickEvent, 671
  - handleDragEvent, 671
  - setDragAction, 671
  - setSnapPosition, 672
  - setSnappedAction, 672
  - Snapper, 670
- touchgfx::SnapshotWidget
  - ~SnapshotWidget, 673
  - draw, 673
  - getAlpha, 674
  - getSolidRect, 674
  - getType, 674
  - makeSnapshot, 674
  - setAlpha, 675



- SnapshotWidget, 673
- touchgfx::SwipeContainer
  - add, 677
  - getNumberOfPages, 678
  - handleClickEvent, 678
  - handleDragEvent, 678
  - handleGestureEvent, 679
  - handleTickEvent, 679
  - remove, 679
  - setEndSwipeElasticWidth, 679
  - setPageIndicatorBitmaps, 680
  - setPageIndicatorXYWithCenteredX, 680
  - setPageIndicatorXY, 680
  - setSelectedPage, 680
  - setSwipeCutoff, 681
- touchgfx::TextArea
  - draw, 683
  - getAlpha, 683
  - getColor, 684
  - getIndentation, 684
  - getLinespacing, 684
  - getRotation, 684
  - getSolidRect, 684
  - getTextHeight, 685
  - getTextWidth, 685
  - getType, 685
  - getTypedText, 685
  - getWideTextAction, 686
  - resizeHeightToCurrentText, 686
  - resizeToCurrentText, 686
  - setAlpha, 686
  - setBaselineY, 687
  - setColor, 687
  - setIndentation, 687
  - setLinespacing, 688
  - setRotation, 688
  - setTypedText, 688
  - setWideTextAction, 688
  - setXBaselineY, 689
  - TextArea, 683
- touchgfx::TextAreaWithOneWildcard
  - draw, 690
  - getTextHeight, 691
  - getTextWidth, 691
  - getType, 691
  - getWildcard, 691
  - setWildcard, 691
  - TextAreaWithOneWildcard, 690
- touchgfx::TextAreaWithTwoWildcards
  - draw, 693
  - getTextHeight, 693
  - getTextWidth, 693
  - getType, 694
  - getWildcard1, 694
  - getWildcard2, 694
  - setWildcard1, 694
  - setWildcard2, 694
  - TextAreaWithTwoWildcards, 693
- touchgfx::TextAreaWithWildcardBase
  - calculateTextHeight, 695
  - TextAreaWithWildcardBase, 695
- touchgfx::TextButtonStyle
  - setText, 697
  - setTextColors, 697
  - setTextPosition, 698
  - setTextRotation, 698
  - setTextXY, 698
  - setTextX, 698
  - setTextY, 699
- touchgfx::TextProgress
  - ~TextProgress, 700
  - getAlpha, 700
  - getColor, 700
  - getNumberOfDecimals, 701
  - getTypedText, 701
  - setAlpha, 701
  - setColor, 702
  - setNumberOfDecimals, 702
  - setProgressIndicatorPosition, 702
  - setTypedText, 703
  - setValue, 703
  - TextProgress, 700
- touchgfx::TextProvider
  - getNextChar, 704
  - getNextLigature, 704
  - initialize, 705
  - TextProvider, 704
- touchgfx::Texts
  - getLanguage, 706
  - getText, 706
  - setLanguage, 706
- touchgfx::TextureMapper
  - ~TextureMapper, 711
  - applyTransformation, 711
  - draw, 711
  - drawTriangle, 711
  - getAlpha, 712
  - getBitmap, 712
  - getBitmapPositionX, 712
  - getBitmapPositionY, 712
  - getBoundingRect, 713
  - getCameraDistance, 713
  - getCameraX, 713
  - getCameraY, 713
  - getOrigoX, 713
  - getOrigoY, 714
  - getOrigoZ, 714
  - getRenderingAlgorithm, 714
  - getScale, 714
  - getSolidRect, 714
  - getType, 715
  - getX0, 715
  - getX1, 715
  - getX2, 715
  - getX3, 716
  - getXAngle, 716



- getY0, [716](#)
- getY1, [716](#)
- getY2, [716](#)
- getY3, [717](#)
- getYAngle, [717](#)
- getZ0, [717](#)
- getZ1, [717](#)
- getZ2, [717](#)
- getZ3, [718](#)
- getZAngle, [718](#)
- lookupRenderVariant, [718](#)
- RenderingAlgorithm, [710](#)
- setAlpha, [718](#)
- setBitmap, [719](#)
- setBitmapPosition, [719](#)
- setCamera, [719](#)
- setCameraDistance, [720](#)
- setOrigo, [720](#)
- setRenderingAlgorithm, [720](#)
- setScale, [721](#)
- updateAngles, [721](#)
- updateXAngle, [721](#)
- updateYAngle, [722](#)
- updateZAngle, [722](#)
- touchgfx::TiledImage
  - draw, [725](#)
  - getOffset, [725](#)
  - getSolidRect, [725](#)
  - getType, [726](#)
  - getXOffset, [726](#)
  - getYOffset, [726](#)
  - setBitmap, [726](#)
  - setOffset, [727](#)
  - setXOffset, [727](#)
  - setYOffset, [727](#)
  - TiledImage, [723](#)
- touchgfx::TiledImageButtonStyle
  - setHeight, [730](#)
  - setTileBitmaps, [730](#)
  - setTileOffset, [730](#)
  - setWidth, [731](#)
- touchgfx::ToggleButton
  - forceState, [732](#)
  - getState, [732](#)
  - getType, [732](#)
  - handleClickEvent, [732](#)
  - setBitmaps, [733](#)
  - ToggleButton, [732](#)
- touchgfx::ToggleButtonTrigger
  - forceState, [734](#)
  - getToggleCanceled, [734](#)
  - handleClickEvent, [734](#)
  - setToggleCanceled, [735](#)
- touchgfx::TouchArea
  - draw, [736](#)
  - getSolidRect, [736](#)
  - getType, [736](#)
  - handleClickEvent, [737](#)
  - handleDragEvent, [737](#)
  - setPressedAction, [737](#)
  - TouchArea, [736](#)
- touchgfx::TouchButtonTrigger
  - handleClickEvent, [738](#)
- touchgfx::TouchCalibration
  - setCalibrationMatrix, [739](#)
  - translatePoint, [739](#)
- touchgfx::TouchController
  - ~TouchController, [739](#)
  - init, [740](#)
  - sampleTouch, [740](#)
- touchgfx::Transition
  - ~Transition, [741](#)
  - handleTickEvent, [741](#)
  - init, [742](#)
  - isDone, [742](#)
  - setScreenContainer, [742](#)
  - tearDown, [742](#)
  - Transition, [741](#)
- touchgfx::TwoWildcardTextButtonStyle
  - setTwoWildcardText, [744](#)
  - setTwoWildcardTextColors, [744](#)
  - setTwoWildcardTextPosition, [744](#)
  - setTwoWildcardTextRotation, [745](#)
  - setTwoWildcardTextXY, [745](#)
  - setTwoWildcardTextX, [745](#)
  - setTwoWildcardTextY, [745](#)
  - setWildcardTextBuffer1, [746](#)
  - setWildcardTextBuffer2, [746](#)
- touchgfx::TypedText
  - getAlignment, [747](#)
  - getFont, [748](#)
  - getId, [748](#)
  - getText, [748](#)
  - getTextDirection, [748](#)
  - hasValidId, [748](#)
  - registerTexts, [749](#)
  - registerTypedTextDatabase, [749](#)
  - TypedText, [747](#)
- touchgfx::UIEventListener
  - ~UIEventListener, [750](#)
  - handleClickEvent, [750](#)
  - handleDragEvent, [751](#)
  - handleGestureEvent, [751](#)
  - handleKeyEvent, [751](#)
  - handlePendingScreenTransition, [751](#)
  - handleTickEvent, [751](#)
- touchgfx::Unicode
  - atoi, [753](#)
  - fromUTF8, [753](#)
  - itoa, [754](#)
  - snprintf, [754](#), [755](#)
  - snprintfFloat, [756](#), [757](#)
  - snprintfFloats, [758](#), [760](#)
  - strlen, [761](#)
  - strncmp, [761](#)
  - strncmp\_ignore\_white\_spaces, [762](#)

- strncpy, 762, 763
- toUTF8, 763
- UnicodeChar, 753
- utoa, 764
- vsnprintf, 764, 765
- touchgfx::Vector
  - add, 766
  - clear, 766
  - contains, 767
  - isEmpty, 767
  - maxCapacity, 767
  - operator[], 767, 768
  - quickRemoveAt, 768
  - remove, 768
  - removeAt, 768
  - reverse, 769
  - size, 769
  - Vector, 766
- touchgfx::Vector4
  - crossProduct, 770
  - Vector4, 770
- touchgfx::View
  - bind, 771
  - View, 771
- touchgfx::Widget
  - ~Widget, 772
  - getLastChild, 773
  - getType, 773
  - Widget, 772
- touchgfx::WildcardTextButtonStyle
  - setWildcardText, 775
  - setWildcardTextBuffer, 775
  - setWildcardTextColors, 775
  - setWildcardTextPosition, 775
  - setWildcardTextRotation, 776
  - setWildcardTextXY, 776
  - setWildcardTextX, 776
  - setWildcardTextY, 776
- touchgfx::ZoomAnimationImage
  - ~ZoomAnimationImage, 780
  - getAlpha, 781
  - getAnimationDelay, 781
  - getLargeBitmap, 781
  - getScalingMode, 781
  - getSmallBitmap, 781
  - getType, 782
  - handleTickEvent, 782
  - isRunning, 782
  - isZoomAnimationRunning, 782
  - setAlpha, 782
  - setAnimationDelay, 783
  - setAnimationEndedCallback, 783
  - setBitmaps, 783
  - setCurrentState, 783
  - setDimension, 784
  - setHeight, 784
  - setPosition, 784
  - setScalingMode, 785
  - setWidth, 785
  - startTimerAndSetParameters, 785
  - startZoomAndMoveAnimation, 786
  - startZoomAnimation, 787
  - States, 780
  - updateRenderingMethod, 787
  - updateZoomAnimationDeltaXY, 787
  - ZoomAnimationImage, 780
- touchgfx::colortype
  - colortype, 220
  - getColor32, 221
  - operator uint16\_t, 221
- touchgfx\_generic\_init
  - touchgfx, 59
- touchgfx\_init
  - touchgfx, 60
- transformDisplayToFrameBuffer
  - touchgfx::DisplayTransformation, 245, 246
- transformFrameBufferToDisplay
  - touchgfx::DisplayTransformation, 246
- Transition, 740
  - touchgfx::Transition, 741
- translatePoint
  - touchgfx::TouchCalibration, 739
- translateRectToAbsolute
  - touchgfx::Drawable, 273
- tryTakeFrameBufferSemaphore
  - touchgfx::OSWrappers, 492
- TwoWildcardTextButtonStyle< T >, 742
- TypedText, 746
  - touchgfx::TypedText, 747
- TypedText::TypedTextData, 749
- UIEventListener, 750
- Unicode, 752
- UnicodeChar
  - touchgfx::Unicode, 753
- unlink
  - touchgfx::Container, 230
- unlockFrameBuffer
  - touchgfx::HAL, 359
- unregisterTimerWidget
  - touchgfx::Application, 133
- updateAbstractShapeCache
  - touchgfx::AbstractShape, 107
- updateAngle
  - touchgfx::AbstractShape, 107
- updateAngles
  - touchgfx::TextureMapper, 721
- updateArcEnd
  - touchgfx::Circle, 204
- updateArcStart
  - touchgfx::Circle, 204
- updateClock
  - touchgfx::AbstractClock, 69
  - touchgfx::AnalogClock, 115
  - touchgfx::DigitalClock, 244
- updateEnd
  - touchgfx::Line, 454, 455

- updateIndicatorPosition
  - touchgfx::Slider, [666](#)
- updateLineWidth
  - touchgfx::Line, [455](#), [456](#)
- updateRenderingMethod
  - touchgfx::ZoomAnimationImage, [787](#)
- updateStart
  - touchgfx::Line, [456](#)
- updateTextPosition
  - touchgfx::ButtonWithLabel, [170](#)
- updateXAngle
  - touchgfx::TextureMapper, [721](#)
- updateYAngle
  - touchgfx::TextureMapper, [722](#)
- updateZAngle
  - touchgfx::TextureMapper, [722](#)
- updateZoomAnimationDeltaXY
  - touchgfx::ZoomAnimationImage, [787](#)
- useSMOCDrawing
  - touchgfx::Screen, [595](#)
- usingSMOC
  - touchgfx::Screen, [596](#)
- utoa
  - touchgfx::Unicode, [764](#)
- vSync
  - touchgfx::HAL, [359](#)
- valueToPosition
  - touchgfx::Slider, [667](#)
- Vector
  - touchgfx::Vector, [766](#)
- Vector< T, capacity >, [765](#)
- Vector4, [769](#)
  - touchgfx::Vector4, [770](#)
- View
  - touchgfx::View, [771](#)
- View< T >, [770](#)
- vsprintf
  - touchgfx::Unicode, [764](#), [765](#)
- waitForFrameBufferSemaphore
  - touchgfx::DMA\_Interface, [252](#)
- waitForVSync
  - touchgfx::OSWrappers, [492](#)
- wasOutlineTooComplex
  - touchgfx::Outline, [495](#)
  - touchgfx::Rasterizer, [567](#)
- WideTextAction
  - touchgfx, [50](#)
- Widget, [772](#)
  - touchgfx::Widget, [772](#)
- width
  - touchgfx::GlyphNode, [333](#)
- WildcardTextButtonStyle< T >, [773](#)
- writeRegister
  - touchgfx::I2C, [369](#)
- ZoomAnimationImage, [777](#)
  - touchgfx::ZoomAnimationImage, [780](#)