TouchGFX v4.12.3 User's Manual

www.st.com
www.touchgfx.com

Contents

Chapter 1

Changelog

```
4.12.3
======

* Release date: September 25rd, 2019

* New TouchGFX Core Features (since 4.12.0):
  * Binary Fonts: Binary fonts can be used as an alternative to
    compiling and linking font information in to your application. The
    main advantages of this approach is to get a smaller application
    binary and get a flexibility in providing different sets of fonts
    with your device.
  * Font Caching: Support for caching binary fonts, suitable for
    loading only the required characters from a file system, when a
    string is displayed.
  * Binary Translations: Support for binary translations, suitable for
    loading translations from a file system as opposed to linking them
    into the application.
    Read more about these feature here:
    https://touchgfx.zendesk.com/hc/en-us/articles/360024979552
  * Support for non-memory-mapped flash storage for 16bpp displays,
    allows storage of images and fonts in e.g. inexpensive SPI
    flashes.
  * Recognition of Unicode sequences for Arabic ligatures Allah,
    Akbar, Mohammad, Salam, Rasoul, Alayhe, Wasallam and Rial Sign.

* Bugfixes in TouchGFX Core:
  * TextureMapper (bilinear) would fail to draw L8_RGB888 and RGB888
    bitmaps on 24bpp displays correctly.
  * Setting a text with a wildcard in a TextArea without wildcard
    support in combination with RTL could cause a crash.
  * If a CacheableContainer was smaller than the associated bitmap,
    the size of the container would not be correct.
  * Fixed SnapshotWidget on 8bpp LCDs.
  * Fixed rendering of some Arabic ligatures.
  * Fixed rendering of some Hindi ligatures.
  * Fixed bug when applying certain GSUB substitution rules.
  * Fixed bug that binary fonts contained extra rules.


4.12.2
======

* Release date: August 22nd, 2019

* New TouchGFX Core Features:
  * WordWrapping wide text using TextArea::setWideTextAction() now
    wraps at normal space as well as Unicode characters 0x200B (Zero
    Width Space).

* Bugfixes in TouchGFX Core:
  * Binary fonts: The fontConverter tool was not writing kerning data
    into binary font files when the "binary_fonts" option was
    specified in the application configuration. This caused texts to
```

```
    appear incorrect when using binary fonts.


4.12.1
======

* Release date: August 15th, 2019

* Updated "Third Party Components.pdf" to reflect updated components
  * libpng-1.6.36
  * zlib-1.2.11
  * freetype-2.9.1

* Bugfixes in TouchGFX Designer:
  * Fixed a bug where having a delay action together with a button
    clicked action would result in compilation errors.
  * Fixed a bug where Canvas Buffer for a Screen was not correctly
    updated when adding a Canvas Widget to a Custom Container Instance.
  * Fixed a bug where an error message in the Online Applications
    window would get stuck.
  * Fixed faulty rendering on the Canvas in the Designer when using
    the Alpha value of the different Progress Indicators.
  * Fixed a bug where creating a new project, closing the Designer
    without saving it, and reloading the project would cause the
    project to have no available typographies.
  * Updated error message when trying to import an already open UI
    into another project to be more clear.
  * Fixed a bug where the Text Manager could have multiple foci
    visually in a specific circumstance.
  * Fixed a bug where the Properties tab for a Widget would not
    display a red border correctly, when an error is present on the
    Widget.
  * Fixed a bug where using the Consolas font would render incorrectly
    on the Canvas in the Designer after reloading a project using that
    font.

* Bugfixes in TouchGFX Core:
  * TextureMapper bug if Display Rotation was in use.
  * Disregard kerning data for CachedFont.
  * CachedFont did not look in font cache for the fallback character.


4.12.0
======

* Release date: 06-07-2019

* Important upgrade information:
  * Public version of drawGlyph has been removed. Use drawString
    instead.
  * Using bitmap format ARGB8888 for opaque images will no longer
    dither to 565 but keep full 888 colors. Using ARGB8888 for
    non-opaque images will still dither to 565 when the opaque format
    is RGB565.
  * Images converted to BW_RLE will no longer fall back to BW if the
    BW_RLE format causes the converted image to be larger. Instead a
    warning will be generated by the image converter. Use the Designer
    (or the new configuration file) to specify BW or BW_RLE for each
    individual image.

* New TouchGFX Designer Features:
  * A custom container can now be nested within another custom
    container. This enables composing custom components into larger
    custom components indefinitely.
  * A custom container supports defining custom triggers and custom
    actions, a screen supports defining custom actions. These triggers
    and actions support the flow of information from one component to
    another component. Using such triggers and actions in interactions
    within the Designer enables doing more real world application
    behaviour without leaving the Designer. Check out the
    documentation for further introduction.
  * A Container can now be generated as a CacheableContainer.
  * A new "Images" tab has been added for setting up individual image
```

```
    configurations (Image Format, Dither Algorithm, Layout Rotation,
    etc.).
  * Application settings and other new settings have been relocated to
    the "Config" tab.

* New TouchGFX Core Features:
  * Upgraded 3rd party libraries used by framework tools. This results
    in much nicer looking texts.
  * Improved kerning through larger kerning table.
  * Thai fonts are now rendered better with tighter line spacing and
    better rendering of Sara Am in some cases.
  * Preliminary support for Hindi (Devanagari). The following GSUB
    tables are applied: nukt (Nukta Forms), akhn (Akhands), rkrf
    (Rakar Forms), cjct (Conjunct Forms), vatu (Vattu Variants), rphf
    (Reph Forms), pref (Pre-Base Forms), half (Half Forms), blwf
    (Below-base Forms), abvf (Above-base Forms), pstf (Post-base
    Forms), and cfar (Conjunct Form After Ro). The following are NOT
    currently supported: abvs (Above-base Substitutions), blws
    (Below-base Substitutions), and psts (Post-base Substitutions).
    Also, not all GSUB tables types are supported.
  * Added a new Line::updateLengthAndAngle() API.
  * Added support for partial framebuffers rendering and updates.
  * Added simple string printing for debugging.
  * Allow changing the BitmapCache after initialization.
  * New macros for setting sections names for flash programming.
  * Added Circle::updateArc() to update arc start and arc end with
    minimal invalidation areas.
  * Updated CircleProgress to use higher precision calculations for
    updates.
  * Added CacheableContainer for offscreen widgets rendering.
  * Added support for L8 graphics assets with 16bit, 24bit and 32bit
    palettes.
  * Added support for L8 hardware acceleration via DMA2D.
  * Added new LCD32bpp framebuffer renderer.

* Bugfixes in TouchGFX Designer:
  * ProgressIndicator is updated automatically after call to
    CircleProgress::setStartEndAngle(),
    ImageProgress::setAnchorAtZero() and
    TextProgress::setNumberOfDecimals().

* Bugfixes in TouchGFX Core:
  * Fixed redraw of circleProgressIndicator when setting new value.
  * Removed additional screen redraw after a screen transition is
    complete. This additional redraw caused performance issues on some
    platforms. Invalidating the entire screen in
    Screen::afterTransition(), if required, is now the responsibility
    of the application developer.


4.11.0
======

* Release date: March 1st, 2019

* Important upgrade information:
  * If your application includes LCD.hpp and expects to have access to
    HAL, this will no longer work since LCD.hpp no longer includes
    HAL.hpp. Make sure to include HAL.hpp in this case. Older versions
    of sample applications Demo1 and Demo2 had this issue and have been
    updated.

* New TouchGFX Designer Features:
  * Added Bring Forward/Send Backwards support for widgets, via UI
    Buttons and keyboard shortcuts Ctrl + F, Ctrl + B.
  * Added support for copy and paste of Screens and
    CustomContainerDefinitons.
  * Added support for reordering CustomContainerDefinitions.
  * Switching between Screen and CustomContainerDefinitions now
    remembers the previously selected Screen and
    CustomContainerDefinition.
  * The last used typography is now used when creating new texts and
    widgets that use text.
```

```
  * Added new tree icon for CustomContainerInstances.
  * Disabled continuous code generation and compiling.
  * Improved readability of the output in Detailed Log window.
  * Widget Wildcard Characters added to the Texts tab, which adds
    default wildcard characters when using some widgets
  * Improved performance when loading a project.
  * Improved performance when generating a project.
  * Improved performance of validation engine.
  * Added support for 6 bit color displays (8bpp).
  * Added support for setting RadioButtonGroup for RadioButtons.
  * Added support for Display Rotation (Landscape/Portrait).
  * Added support for setting Landscape/Portrait simulator skins in
    the Designer.
  * Added support for the following widgets: AnalogClock,
    DigitalClock, TextureMapper, AnimatedTextureMapper & Shape.
  * The Designer now generates the Makefile and Visual Studio files
    used for running the Simulator.

* New TouchGFX Core Features:
  * Added support for 6 bit color displays (RGBA2222, BGRA2222,
    ARGB2222 and ABGR2222 framebuffer formats).
  * Added support for Thai.
  * Improved rendering of Arabic text.
  * Added handling of negative line spacing.

* Bugfixes in TouchGFX Designer:
  * Fixed Ctrl + A (select all) not working for
    CustomContainerDefinitions.
  * Fixed reordering of Screens selecting the first screen in the list
    and deleting the undo/redo history for the Screen that was moved.
  * Fixed bug where the undo/redo history would become broken after
    selecting the Application node.
  * Fixed application names not being allowed to start with a number
    or contain "-" or "_".
  * Fixed loading an application while on the CustomContainer tab
    resulting in erroneous content on the canvas.
  * Fixed pressing undo after moving multiple elements into a
    container resulting in a crash.
  * Fixed font files being locked when loading a project.
  * Fixed error not showing up on components that use text, when
    removing their Resource Text.
  * Fixed bug where loading a faulty application by double-clicking a
    TouchGFX file would cause the splash screen to get stuck.
  * Fixed faulty position code generation for ModalWindow.
  * Fixed missing "Move widget" interaction support for
    ScrollableContainer, ScrollList & ScrollWheel.
  * Fixed the ordering of the Recent Applications list. Now correctly
    updates when opening an application.
  * Fixed bug where inserting a widget could add an empty undo item to
    the undo/redo history.
  * Fixed missing header text and description in the properties pane
    for CustomContainerDefinitions.
  * Fixed bug where idle CPU usage was higher than expected.
  * Fixed bug where setting an interaction on a FlexButton inside a
    CustomContainer would generate faulty code.
  * Fixed bug where setting a mixin on a widget was not undo-able.
  * Fixed missing undo/redo functionality for adding styles to
    FlexButton.
  * Fixed wrong order of initializations when using numerous slider
    callbacks in interactions.

* Bugfixes in TouchGFX Core:
  * Fixed precision in CWR Painters for 4bpp and 2bpp.
  * Fixed precision in alpha blending formulaes for 8bpp, 16bpp and
    24bpp.


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 functons 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" dissappearing 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 unitialized 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 squences of up to
```

```
    three character are recognized.
  * Added support for Microsoft Visual Studio 2017.
  * TextArea and TextAreaWithWildcard(s) now support
    setWideTextAction() 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
    setWideTextAction() 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::vsnprintf 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 speeedup 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 repetions 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 automitical 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 simulatenously. 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.
  * snprintf("%x") would generate uppper case hex. Now "%X" generates
    uppercase hex and "%x" generates lower case hex, just like the
    standard snprintf().
```

```
  * Fixed randomness for demos when running on Linux.
  * Fixed redrawing when using heavily italicized fonts.
  * Pointer to ModelListener in Model class for all TouchGFX
    applications was not properly initialized (NULL).
  * Fixed support for heavily italicized fonts in touchgfx::TextArea.
  * Subtle error in the Image Converter where column 0 could get
    slightly incorrect pixel colors. As a result the entire image
    could be slightly wrong, probably not noticeable.
  * Minor error in Slider where values were not distributed evenly.

* Deprecated
  * LCD::drawGlyph() has been deprecated. Use LCD::drawString instead.

* Update Procedure
  * Compatible with existing 4.x applications and HAL ports. Please
    refer to this article for details:
    https://touchgfx.zendesk.com/hc/en-us/articles/206819819


4.4.2
=====

* Release date: November 26th, 2015

* Bugfixes:
  * Corrected rare GUI task hangup on STM32F7 targets when compiling
    with IAR 7.x


4.4.1
=====

* Release date: October 27th, 2015

* Bugfixes:
  * Corrected occasional GUI task hangup on STM32F7 targets when
    compiling with Keil 5.x
  * Fixed occasional tearing on STM32 F469 EVAL/Discovery boards when
    using DSI in landscape orientation and single frame buffer mode.
  * Modified IAR flash loader settings for STM32 F469 boards to enable
    programming of internal flash (Note: QuadSPI flash must still be
    programmed from ST-Link Utility as there are no IAR loaders for
    this)
  * GPIO class for perf. measurement for STM32F746G-EVAL boards now
    properly uses the BSP_LED functions. Note that only two signals
    are active on this board per default because LED2 and LED4 use IO
    Expander, making them unsuited for measuring performance.
  * Removed annoying "Get Alternative File" dialog popups in IAR
    Workbench when debugging Cortex-M7 applications.


4.4.0
=====

* Release date: October 6th, 2015

* New features
  * Added support for the Cortex-M7 core.
  * Introduced concept of "finger size" for touch input. When used,
    TouchGFX will attempt to find touchable widgets in the area
    surrounding the reported x,y coordinates, so users no longer have
    to click precisely on a widget. This feature makes it substantially
    easier to hit small buttons.
    See HAL::setFingerSize().
  * Supports Visual Studio 2015
  * Visual Studio projects for Demos and Examples now include
    Application.props under Resources for quick access. As always a
    rebuild might be required when altering the contents of
    Application.props.
  * Support for Bitmap Fonts in BDF format. If the requested font size
    is not available in the font file, the font converter will write
    the supported font size(s) in the error message. See the example
    monochrome_example for usage.
```

```
  * Generating assets now issues better error messages when spaces are
    detected in paths and file names.
  * All ST boards can now be flashed from the command line provided
    that ST-Link Utility Release 3.7 has been installed. Simply use
    'make -f target/ST/<board>/Makefile flash' to build and flash
    your application to the connected board. If timeouts occur during
    flashing, go to Device Manager in Windows and disable "MBED
    microcontroller USB Device" under "Disk drives".
  * New touchgfx-env version 2.5 available with new gcc cross compiler
    version 4.9.3. The older version 4.8.4 could generate invalid code
    for Cortex-M7 cores in rare cases.

* Board support
  * Added support for the STM32F7xx processors
  * Added support for the STM32F746G-DISCO and STM32756G-EVAL boards
  * Added support for the STM32F469 processor with DSI displays
  * Added support for the STM32469I-EVAL and STM32469I-Discovery boards

* Bugfixes
  * TextureMapper and ScaleableImage now draws images correctly when
    using "rotate90".
  * Fixed potential initialization order bug in STM32F4DMA.cpp
  * Fixed bug that limited number of glyphs in a single font to 32768.
    Now supports 65536 glyphs per font as intended.
  * Fixed bug that caused hal.lockDMAToFrontPorch(false) to not have
    any effect in single frame buffer mode.
  * ButtonWithLabel correctly center texts vertically if text contains
    newlines


4.3.0
=====

* Release date: June 8th, 2015

* New features
  * TextureMapper widget added.
    The TextureMapper is a highly optimized image renderer that can be
    used for displaying an image that is scaled and/or rotated in two
    or three dimensions during run time. This can be used for doing
    advanced rotation animations of images. See manual or
    texture_mapper_example for more information. LCD has new methods
    for drawing triangles and corresponding scan lines,
    drawTextureMapTriangle and drawTextureMapScanLine
  * Alpha Channel Dithering
    Images with alpha channel can now get the alpha channel dithered
    for smoother alpha gradients, see examples or Application Development
    section in manual for details
  * Compression of 1BPP (monochrome) bitmaps
    Added image format option of BW_RLE, which will cause bitmaps to
    be automatically run-length encoded if that takes up less space than
    the regular per-pixel format. Yields substantially smaller bitmap
    footprint in many cases. See advanced chapter in manual for details.
  * Slider widget added.
    See manual or slider_example for more information.
  * Makefiles has been updated to work with make-4.1.
  * Added support for the LPC4088 processor and the Embedded Artists
    LPC4088 Display Module board.
  * Individual bitmaps can now be placed in internal flash instead
    of external by having the bitmap file name include the string ".int."
  * MoveAnimator, FadeAnimator and ZoomAnimationImage now have a
    cancelMoveAnimation/cancelFadeAnimation/cancelZoomAnimation
    method.

* Update procedure
  * Compatible with existing 4.X applications. Just replace the
    touchgfx folder.
  * Check Known Issues in the documentation.

* Info
  * The evaluation version of TouchGFX is now distributed with source
    code for the hardware abstraction layer instead of a precompiled
    library. This makes it possible to port the evaluation version
```

```
    to custom hardware instead of it being limited to the supported
    eval boards only. Instead, the evaluation version now has a
    TouchGFX watermark which will appear occasionally.
  * Memory consumption reduced due to improved rendering algorithm.
    Will typically allow GUI task stack to be reduced by around 1400
    bytes compared to version 4.2.0 (depending on actual application).
    Additionally the statically allocated memory is also reduced by
    around 1KB.
  * Maximum number of visible widgets limit of 150 removed.
  * Added two new demos for 640x480 and 480x272 resolutions showcasing
    new features, graphs, internationalization and custom widgets.
  * Drawable.setPosition() now calls setXY(), setWidth() and
    setHeight() for easier subclassing.
  * AbstractPainterRGB565 and AbstractPainterBW are recommended as
    base classes when implementing your own painters.
  * CanvasWidgets now have setAlpha() and getAlpha() methods. Your
    custom Painter classes must implement this, or inherit from the
    AbstractPainterRGB565 class
  * Maximum number of registered timer widgets increased from 16 to
    24.
  * touchgfx-env updated to 2.4. The environment does not beep anymore.
  * Board Support Package for STM324x9I-EVAL is now based on the
    STMCubeF4 drivers.

* Bugfixes
  * Screen::handleGestureEvent now converts x/y to relative coordinates
  * Fixed bug when drawing several objects on the same canvas using
    moveTo() more than once.
  * ZoomAnimationImage movement relative to scaling did not use
    correct easing equation.
  * PainterRGB565 did not blend green alpha correctly.
  * RadioButtonGroup now initializes callbacks to zero.
  * ScalableImage now works with bitmaps with transparancy.
  * AnimatedImage would display the start and end of an animation twice.
  * Default implementation of CanvasWidget::getMinimalRect() returned
    coordinates relative to its parent, not itself.
  * ScrollableContainer erroneously unregistered itself as a timer widget
    at every tick, which made it difficult to use with other timer-based
    operations.

* Performance
  * ScalableImage and ZoomAnimationImage has been optimized for better
    performance.


4.2.0
=====

* Release date: January 14th, 2015

* Performance
  * Substantially improved rendering performance, which in most cases
    will result in a 25% reduction of time it takes to render a frame.

    NOTE: This optimization does not necessarily work on all targets
    so it must be manually enabled. See the "Optimization" chapter in
    the porting guide for how to enable this optimization for existing
    portings. It is STRONGLY recommended that the optimization is
    enabled. This optimization is enabled for all appropriate
    evaluation boards in the 4.2.0 board packages.

* Major new features
  * Added CanvasWidgets for smooth, anti-aliased drawing of geometric
    shapes. Currently Line, Circle and a more generic Shape have been
    implemented. CanvasWidgets can be painted with a solid color (+
    alpha), a bitmap (including alpha) or a custom painter. Read more
    on Canvas Widgets and Painters in the documentation.
  * Added support for the Keil compiler and uVision4 IDE. Please refer
    to the "Supported Hardware" section of the TouchGFX Distribution
    chapter in the documentation for a list of Keil-supported targets.

* New features
  * It is now possible to specify an animation start delay on
```

```
    ZoomAnimationImage, MoveAnimator and FadeAnimator.
  * Added Board support for 4.3" TouchGFX Demo board w. LPC4350 (No
    internal flash)
  * RadioButton and RadioButtonGroup widgets added. See
    app/examples/radio_button_example and documentation.
  * LPC43XX and LPC1788 can now fill rectangles using DMA.
  * Visual Studio 2013 is now supported.
  * Preliminary support for Visual Studio 2015 Preview version.
  * Improved performance when generating assets.
  * New canvas_widget_example added to the example directory.
  * The "using namespace touchgfx" present in various header files can
    now be avoided by defining the symbol NO_USING_NAMESPACE_TOUCHGFX
    in your project.

* TouchGFX env
  * The message displayed when starting a shell has been fixed with
    correct path to examples.

* Bugfixes
  * Fixed bug in simulator for 1bpp displays when width and/or height
    was not not a multiple of 8.
  * Fixed bug in ScrollableContainer where CANCEL events where not
    always delegated to correct child, causing e.g. buttons to remain
    pressed when dragging outside SC area.
  * Fixed bug when rendering chromArt fonts with a rotated display.
  * Fixed bug - Keyboard widget setTouchable(false) had no effect.
  * Freescale K70 DMA now checks the appropriate DONE bit in TCD0_CSR.
  * On ST processors fixed bug with rotated texts rendered by ChromArt
    when in non-native display orientation.

* Board support
  * Embedded Artists LPC4357DevKit board package: CPU clocked to
    204Mhz (previously 96Mhz). Now uses SPIFI flash instead of NOR.

* Update procedure
  * Compatible with existing 4.X applications. Just replace the
    touchgfx folder.

* Info
  * Documentation has been updated.


4.1.1
=====

* Release date: October 29th, 2014

* New features
  * Mixin: MoveAnimator added. The MoveAnimator mixin makes the
    template class T able to animate a movement from its current
    position to a specified end position. See
    app/example/move_fade_example.
  * Mixin: FadeAnimator added. The FadeAnimator mixin makes the
    template class T able to animate an alpha fade from its current
    alpha value to a specified end alpha value. See
    app/example/move_fade_example.
  * ScalableImage and ZoomAnimationImage now support alpha per pixel
    bitmaps and alpha per bitmap
  * ScalableImage and ZoomAnimationImage now support ARGB8888 format
    bitmaps

* Bugfixes
  * Fixed a bug causing the Keyboard widget to render incorrectly in
    rare cases.
  * Fixed a bug causing drag event coordinates to be incorrect for
    widgets when placed in a Container with coords != {0,0} which
    itself was placed in a ScrollableContainer.
  * The Application class now properly keeps track of number of times
    registerTimerWidget vs. unregisterTimerWidget is called for a
    given widget, meaning that if registered several times it now
    requires same number of unregisters before widget no longer
    receives tick events.
  * Some ZoomAnimationImage functions were not virtual as they should
```

```
    be.
  * Some widgets were missing certain getter functions.

* Update procedure
  * Compatible with existing 4.X applications. Just replace the
    touchgfx folder.


4.1.0
=====

* Release date: October 17th, 2014

* New features
  * Now supports monochrome 1BPP displays. See manual for details.
  * Support for dynamic screen orientation change (landscape/portrait)
  * Support for scaling images (See ScalableImage and
    ZoomAnimationImage drawables)
* Demo
  * Home Control Demo now support 640x480 mode.
  * Home Control Demo now supports STM324xI-EVAL 5.7" board.

* Board support changes
  * Added support for STM324xI-EVAL 5.7" board (IAR+gcc).
  * Added gcc support for the EmbeddedArtists LPC4357DevKit board.
  * Optimized SPIFI initialization for TouchGFX eval board.

* Bugfixes
  * Adding a persistent Drawable to a ScrollableContainer could cause
    assertion
  * Support for much larger fonts

* Update procedure
  * Compatible with existing 4.X applications. Just replace the
    touchgfx folder.


4.0.0
=====

* Release date: September 26th, 2014

* New features
  * TouchEvent refactoring (API breaking):
    * Drawable::setActive is renamed to Drawable::setTouchable
    * Drawable::isActive is renamed to Drawable::isTouchable
    * Drawable::hijackTouchEvent is deprecated
    * Drawables are now per default not touchables
    * TouchEvents are now always propagated to all containers children
  * Language specific typography and alignment columns support added
    to text converter. Read more about this feature in the
    documentation.
  * Font rendering has been vastly improved with regards to font
    shapes and kerning.
  * Simulator - assert check on new view/presenter/transition size
    when doing screen transition. Failed assert checks probably due
    to missing definition of view/presenter/transition in
    FrontEndHeap.
  * TextArea and ButtonWithLabel now support text rotated 0, 90, 180
    or 270 degrees.
  * Text centering on ButtonWithLabel has been improved in special
    cases.
  * Hardware Accelerated text rendering (4 and 8bpp) on supported ST
    platforms.
  * Ability to cache all items in the bitmap database in external RAM.
  * Support for Freescales K70 MCU.
  * Translation Sheet: Instances of "\<" and "\>" are converted into
    "<" and ">" respectively. This enables literal translated strings
    such as "<Not a wildcard>" using "\<Not a wildcard\>".
  * Support for NXP LPC18XX series of MCU's.

* Bugfixes
  * Rendering error of images with odd width and alpha value less than
```

```
    255
  * Correct handling of TextArea::getTextHeight in case of non
    initialized textArea
  * TextAreaWithWildcard::getTextWidth now includes the width of the
    wildcard text
  * gcc Makefiles now includes *.BMP and *.PNG from image assets.
  * Do not trim leading and trailing white space from any translations
    in the texts sheet.
  * Font converter did not generate font data properly for 8bpp.
  * ButtonWithIcon::setBitmaps - Suppress IAR warning for intentional
    virtual function override.
  * ButtonWithIcon optimized draw functionality
  * In extremely rare cases text could be written slightly outside the
    text area

* Update procedure
  * Due to the TouchEvent refactoring you have to rename functions
    accordingly. You also need to state in any custom widget or
    containers if they need to receive touch events. If you were using
    hijackTouchEvent to prevent children of getting touch events, you
    now need to make sure that all children is not touchable instead.
  * Main.cpp for simulators need to be updated by replacing the line:
        TypedText::registerTypedTextDatabase(TypedTextDatabase::getInstance(), TypedTextDatabase::getInstanceS
    with:
        Texts::setLanguage(0)
    You can also specify a specific language from your text database
    e.g.
        Texts::setLanguage(GB)
    In that case you also need to:
        #include <texts/TextKeysAndLanguages.hpp>
  * Rebuild entire project.

* Info
  * The TouchGFX Manual has been updated considerably.


3.1.0
=====

* New features
  * Added support for FDI uEZGUI-1788-70WVT eval board (NXP LPC-1788
    Cortex M3).
  * Added support for Mjolner TouchGFX Demo Board Rev. 1.1 eval board
    (NXP LPC-4353 Cortex M4/M0 4.3").

* Bugfixes
  * Visual Studio build now rebuild BitmapDatabase.h when new images
    are added to the assets/images folder.

* Update procedure
  * Only if using Visual Studio: Update TouchGFXReleasePath in your
    Visual Studio .props file. Simply edit the file in a text editor.
    The path should be extended with "touchgfx\". See the
    template_application for inspiration.
  * Only if using Visual Studio: Update your Visual Studio project
    file (.vcxproj file). Simply edit the file in a text editor.
    Replace all paths on the form
    "$(TouchGFXReleasePath)\framework\config\msvs\touchgfx_prebuild.targets"
    with
    "$(TouchGFXReleasePath)\config\msvs\touchgfx_prebuild.targets".

* Info
 * Hardware Abstraction Layer architecture has been reworked so that
   all common code for various hardware components (MCUs and drivers)
   is now shared across different target boards. This greatly
   simplifies the porting effort for new/custom boards as long as they
   contain one or more hardware components already supported by
   TouchGFX.


3.0.0
=====
```

```
* New features
  * Visual Studio 2010/2012 support.
  * Added support for png images with alpha channel.
  * Added support for subfolders in assets/bitmaps folder
  * Added support for ST STM32F4X9I-EVAL eval board.
  * Added support for Robert Penners Easing Equations (see
    touchgfx/EasingEquations.hpp).
  * Image converter: Added sanity check of input image file names,
    must not start with digit and must be alphanumeric.
  * Image converter: Added checking against case insensitively file
    name duplicates in input list.
  * Text converter: Added build stopping sanity checks for bpp and
    font_size values.
  * ScrollableContainer: Now supports setScrollbarPadding,
    setScrollbarWidth, setScrollbarColor, and setScrollbarAlpha.
  * ScrollableContainer: Set default value of ScrollThreshold to 5
    pixels, instead of 1.
  * Added support for alpha blending of fonts
    (TextArea::setAlpha(uint8_t alpha))
  * ImageConvert support two different output formats: RGB565 and ARGB8888
  * ImageConvert - two options added to control output format for
    images with/without an alpha channel
  * Touchgfx environment under MinGW is updated due to linker errors
    for large projects. g++ version is updated from 4.6.2 -> 4.8.1
  * Internal RAM footprint improvements
  * Structural changes of target library and hardware abstraction layers

* Bugfixes
  * Fill operation (Box widget) resulted in a crash on the
    lpc4357_emb_artist board
  * Textconvert & fontconvert: Different typographies may now have
    identical properties.
  * Imageconvert & fontconvert: Better error handling for POSIX
    compliant platforms
  * HALSDL: Do not overflow key data type.
  * LanguageXX.cpp files now end with a newline (removing warnings).
  * TextArea::draw now handles non initialized TypedText correctly.

* Update procedure
  * The folders assets/bitmaps and generated/bitmaps must be renamed
    to assets/images and generated/images.
  * Upgrade TouchGFX environment to version 2.0
  * Update any application Makefile to adhere with the Makefile
    specified in the updated template_application
  * Rebuild entire project
  * Convert bmp images that contains the former transparent color to
    png images that uses alpha channels. This can be done
    automatically using a free tool called imagemagick. More info and
    hints can be acquired by writing touchgfx-support@mjolner.com
  * Custom HAL implementations must be updated to conform with the new
    structure

* Info
  * The "magic" transparent color that was previously used for
    transparent color in the bmp format is no longer supported.
    Instead use png images with alpha channel.


2.2.0
=====

* New features
  * Added support for portrait mode with landscape displays at zero
    performance/resource cost.
  * Added kerning support.
  * Added Keyboard example (with IAR project for the Energy Micro
    DK3750 eval board)
  * Changed interface for blitCopy method in LCD.
  * Removed SyncBackBuffer method from HAL.
  * Removed clearLCD method from LCD.
  * Removed fillGradientRect method from LCD.
  * ScrollableContainer supports setScrollbarsVisible(bool visible).
```

```
2.1.0
=====

* First release of TouchGFX as a commercially available framework
```

Chapter 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:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

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 AbstractPainterABGR2222

  A Painter that will paint using a color and an alpha value.
- class AbstractPainterARGB2222

  A Painter that will paint using a color and an alpha value.
- class AbstractPainterARGB8888

  A Painter that will paint using a color and an alpha value.
- class AbstractPainterBGRA2222

  A Painter that will paint using a color and an alpha value.
- 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 AbstractPainterRGBA2222

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.
- class CacheableContainer

  A CacheableContainer is a Container that can have its drawing redirected into a 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 DebugPrinter

  The class DebugPrinter defines the interface for printing debug messages on top of the framebuffer.
- 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 FlashDataReader

  This class is an abstract interface for a class reading data from a flash.

- 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 FrameBufferAllocator

  This class is an abstract interface for a class allocating partial framebuffer blocks.

- 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 LCD16bppSerialFlash

  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 LCD24DebugPrinter

  The class LCD24DebugPrinter implements the DebugPrinter interface for printing debug messages on top of 24bit framebuffer.
- class LCD2bpp

  This class contains the various low-level drawing routines for drawing bitmaps.
- class LCD32bpp

  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 LCD8bpp_ABGR2222

  This class contains the various low-level drawing routines for drawing bitmaps.
- class LCD8bpp_ARGB2222

  This class contains the various low-level drawing routines for drawing bitmaps.
- class LCD8bpp_BGRA2222

  This class contains the various low-level drawing routines for drawing bitmaps.
- class LCD8bpp_RGBA2222

  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 ManyBlockAllocator

  This class is partial framebuffer allocator using multiple blocks.
- 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 underlaying 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 PainterABGR2222

  A Painter that will paint using a color and an alpha value.
- class PainterABGR2222Bitmap

  A Painter that will paint using a bitmap.
- class PainterARGB2222

  A Painter that will paint using a color and an alpha value.
- class PainterARGB2222Bitmap

  A Painter that will paint using a bitmap.
- class PainterARGB8888

  A Painter that will paint using a color and an alpha value.
- class PainterARGB8888Bitmap

  A Painter that will paint using a bitmap.
- class PainterARGB8888L8Bitmap

  A Painter that will paint using a bitmap.
- class PainterBGRA2222

  A Painter that will paint using a color and an alpha value.
- class PainterBGRA2222Bitmap

  A Painter that will paint using a bitmap.
- 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 PainterRGB565L8Bitmap

  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.
- class PainterRGB888L8Bitmap

  A Painter that will paint using a bitmap.
- class PainterRGBA2222

  A Painter that will paint using a color and an alpha value.
- class PainterRGBA2222Bitmap

  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 SingleBlockAllocator

  This class is partial framebuffer allocator using just one block.
- 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 TwoWildcardTextButtonStyle< RepeatButtonTrigger > TwoWildcardTextRepeatButton

  Defines an alias representing the wildcard text repeat button.
- typedef TwoWildcardTextButtonStyle< TouchButtonTrigger > TwoWildcardTextTouchButton

  Defines an alias representing the wildcard text touch button.
- typedef TwoWildcardTextButtonStyle< 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_KERNINGTABLEPOS_BIT8_10 = 0x07, 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.
- enum DMAType { DMA_TYPE_GENERIC, DMA_TYPE_CHROMART }

  Defines a DMAType type.

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.
- void FrameBufferAllocatorWaitOnTransfer ()
  Called by FrameBufferAllocator to wait for a LCD Transfer.
- void FrameBufferAllocatorSignalBlockDrawn ()
  Called by FrameBufferAllocator when a block is drawn.
- 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

BLIT_OP_COPY	Copy the source to the destination.
BLIT_OP_FILL	Fill the destination with color.
BLIT_OP_COPY_WITH_ALPHA	Copy the source to the destination using the given alpha.
BLIT_OP_FILL_WITH_ALPHA	Fill the destination with color using the given alpha.
BLIT_OP_COPY_WITH_TRANSPARENT_PIXELS	Deprecated, ignored. (Copy the source to the destination, but not the transparent pixels)
BLIT_OP_COPY_ARGB8888	Copy the source to the destination, performing per-pixel alpha blending.
BLIT_OP_COPY_ARGB8888_WITH_ALPHA	Copy the source to the destination, performing per-pixel alpha blending and blending the result with an image-wide alpha.
BLIT_OP_COPY_A4	Copy 4-bit source text to destination, performing per-pixel alpha blending.
BLIT_OP_COPY_A8	Copy 8-bit source text to destination, performing per-pixel alpha blending.

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.

Enumerator

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 DMAType

`enum enum DMAType`

Defines a DMAType type.

Enumerator

DMA_TYPE_GENERIC	Generic DMA Implementation.
DMA_TYPE_CHROMART	ChromART hardware DMA Implementation.

6.1.2.6 FrameBuffer

`enum enum FrameBuffer`

Defines a FrameBuffer type.

Enumerator

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

6.1.2.7 GlyphFlags

`enum GlyphFlags`

Glyph flag definitions.

Enumerator

GLYPH_DATA_KERNINGTABLEPOS_BIT8_10	The 8th, 9th and 10th bit of the kerningTablePos.
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".

6.1.2.8 Gradient

`enum enum Gradient`

Defines a gradient type.

Enumerator

GRADIENT_HORIZONTAL	Horizontal gradient.
GRADIENT_VERTICAL	Vertical gradient.

6.1.2.9 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.

6.1.2.10 WideTextAction

```
enum enum WideTextAction
```

See also

TextArea::setWideTextAction

Enumerator

WIDE_TEXT_NONE	Do nothing, simply cut the text in the middle of any character that extends beyond the width of the TextArea.
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

T	The type on which to perform the abs.

Parameters

d	The entity on which to perform the abs.

Returns

The absolute (non-negative) value of d.

6.1.3.2 ceil28_4()

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

Parameters

value	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

x	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	*newScreen*	If non-null, the new screen.
in	*newPresenter*	If non-null, the new presenter.
in	*newTransition*	If non-null, the new transition.

6.1.3.5 fixed28_4Mul()

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

Parameters

a	The fixed28_4 to process.
b	The fixed28_4 to process.

Returns

the result.

6.1.3.6 fixed28_4ToFloat()

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

Parameters

value	The value.

Returns

The value as float.

6.1.3.7 floatToFixed16_16()

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

Parameters

value	The value.

Returns

The value as fixed16_16.

6.1.3.8 floatToFixed28_4()

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

Parameters

value	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

	numerator	The numerator.
	denominator	The denominator.
in,out	*floor*	The floor.
in,out	*mod*	The modifier.

6.1.3.10 FrameBufferAllocatorSignalBlockDrawn()

```
void FrameBufferAllocatorSignalBlockDrawn ( )
```

Called by FrameBufferAllocator when a block is drawn and therefore ready for transfer. The LCD driver should use this method to start a transfer.

6.1.3.11 FrameBufferAllocatorWaitOnTransfer()

```
void FrameBufferAllocatorWaitOnTransfer ( )
```

Called by FrameBufferAllocator to wait for a LCD Transfer, when the allocator has no free blocks. The LCD driver can use this function to synchronize the UI thread with the transfer logic.

6.1.3.12 gcd()

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

Find greatest common divisor of two given numbers.

Template Parameters

T	Generic type parameter.

Parameters

a	The first number.
b	The second number.

Returns

A T.

6.1.3.13 hw_init()

```
void hw_init ( )
```

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

6.1.3.14 LCD2getPixel() [1/2]

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

Parameters

addr	The address.
offset	The offset.

Returns

The pixel value.

6.1.3.15 LCD2getPixel() [2/2]

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

Parameters

addr	The address.
offset	The offset.

Returns

The pixel value.

6.1.3.16 LCD2setPixel() [1/2]

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

Parameters

in	*addr*	The address.
	offset	The offset.
	value	The value.

6.1.3.17 LCD2setPixel() [2/2]

```
FORCE_INLINE_FUNCTION void LCD2setPixel (
            uint16_t * addr,
            int offset,
```

```
            uint8_t value )
```

Parameters

in	*addr*	The address.
	offset	The offset.
	value	The value.

6.1.3.18 LCD2shiftVal()

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

Parameters

offset	The offset.

Returns

The shift value.

6.1.3.19 LCD4getPixel() [1/2]

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

Parameters

addr	The address.
offset	The offset.

Returns

The pixel value.

6.1.3.20 LCD4getPixel() [2/2]

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

Parameters

addr	The address.
offset	The offset.

Returns

The pixel value.

6.1.3.21 LCD4setPixel() [1/2]

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

Parameters

in	*addr*	The address.
	offset	The offset.
	value	The value.

6.1.3.22 LCD4setPixel() [2/2]

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

Parameters

in	*addr*	The address.
	offset	The offset.
	value	The value.

6.1.3.23 lookupBilinearRenderVariant()

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

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

Parameters

bitmap	The bitmap.

Returns

A RenderingVariant based on the bitmap format.

6.1.3.24 lookupNearestNeighborRenderVariant()

```
RenderingVariant lookupNearestNeighborRenderVariant (
```

```
            const Bitmap & bitmap )
```

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

Parameters

bitmap	The bitmap.

Returns

A RenderingVariant based on the bitmap format.

6.1.3.25 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

ScreenType	Class type for the View.
PresenterType	Class type for the Presenter.
TransType	Class type for the Transition.
ModelType	Class type for the Model.

Parameters

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

Returns

Pointer to the new Presenter of the requested type. Incidentally it will be the same value as the old presenter due to memory reuse.

6.1.3.26 memset()

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

Provides utility functions.

Simple implementation of the standard memset function.

Parameters

out	*data*	Address of data to set.
	c	Value to set.
	size	Number of bytes to set.

6.1.3.27 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

	factor1	The first factor.
	factor2	The second factor.
	divisor	The divisor.
out	*remainder*	The remainder.

Returns

An int32_t.

6.1.3.28 operator∗() [1/2]

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

Multiplication operator.

Parameters

multiplicand	The first value to multiply.
multiplier	The second value to multiply.

Returns

The result of the operation.

6.1.3.29 operator*() [2/2]

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

Multiplication operator.

Parameters

multiplicand	The first value to multiply.
multiplier	The second value to multiply.

Returns

The result of the operation.

6.1.3.30 prepareTransition()

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

Parameters

in	*currentScreen*	If non-null, the current screen.
in	*currentPresenter*	If non-null, the current presenter.
in	*currentTrans*	If non-null, the current transaction.

6.1.3.31 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

HALType	The class type of the HAL subclass used for this port.

Parameters

in	*dma*	Reference to the DMA implementation object to use. Can be of type NoDMA to disable the use of DMA for rendering.
in	*display*	Reference to the LCD renderer implementation (subclass of LCD). Could be either LCD16bpp for RGB565 UIs, or LCD1bpp for monochrome UIs or LCD24bpp for 24bit displays using RGB888 UIs.
in	*tc*	Reference to the touch controller driver (or NoTouchController to disable touch input).
	width	The *native* display width of the actual display, in pixels. This value is irrespective of whether the concrete UI should be portrait or landscape mode. It must match what the display itself is configured as.
	height	The *native* display height of the actual display, in pixels. This value is irrespective of whether the concrete UI should be portrait or landscape mode. It must match what the display itself is configured as.
in	*bitmapCache*	Optional pointer to starting address of a memory region in which to place the bitmap cache. Usually in external RAM. Pass 0 if bitmap caching is not used.
	bitmapCacheSize	Size of bitmap cache in bytes. Pass 0 if bitmap cache is not used.
	numberOfDynamicBitmaps	Number of dynamic bitmaps.

Returns

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

6.1.3.32 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

event	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

callback	The callback to be executed. The callback will be given a reference to the AbstractButton.

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

callback	The callback.

7.2.2.4 setAlpha()

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

Parameters

newAlpha	The new alpha.

7.2.2.5 setPressed()

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

Parameters

isPressed	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

hour	The hours (in 12H format).
minute	The minutes (in 12H format).
second	The seconds (in 12H format).
am	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

hour	The hours (in 24H format).
minute	The minutes (in 24H format).
second	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.hpp>`

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

direction	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.

Static Protected Member Functions

- static bool compatibleFramebuffer (Bitmap::BitmapFormat format)
  Check if the provided bitmap format matches the current framebuffer format.

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 compatibleFramebuffer()

```
static inline bool compatibleFramebuffer (
            Bitmap::BitmapFormat format )  [inline], [static], [protected]
```

Helper function to check if the provided bitmap format matches the current framebuffer format.

Parameters

format	A bitmap format.

Returns

True if the format matches the framebuffer format, false otherwise.

7.5.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implemented in PainterARGB8888, PainterGRAY2, PainterGRAY4, PainterRGB565, PainterRGB888, PainterABGR2222, PainterARGB2222, PainterBGRA2222, PainterRGBA2222, PainterARGB8888Bitmap, PainterARGB8888L8Bitmap, PainterGRAY2Bitmap, PainterGRAY4Bitmap, PainterRGB565Bitmap, PainterRGB565L8Bitmap, PainterRGB888Bitmap, PainterRGB888L8Bitmap, PainterABGR2222Bitmap, PainterARGB2222Bitmap, PainterBGRA2222Bitmap, PainterRGBA2222Bitmap, PainterBW, PainterBWBitmap, AbstractPainterRGB565, AbstractPainterABGR2222, AbstractPainterARGB2222, AbstractPainterBGRA2222, AbstractPainterRGBA2222, AbstractPainterARGB8888, AbstractPainterBW, AbstractPainterGRAY2, AbstractPainterGRAY4, and AbstractPainterRGB888.

7.5.3.3 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

offsetX	The offset x coordinate of the invalidated area relative to the widget.
offsetY	The offset y coordinate of the invalidated area relative to the widget.

7.5.3.4 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

alpha	The alpha.

Note

Used internally by Canvas.

7.6 AbstractPainterABGR2222 Class Reference

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

`#include <touchgfx/widgets/canvas/AbstractPainterABGR2222.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 uint8_t mixColors (uint8_t newpix, uint8_t bufpix, uint8_t alpha)

  Mix colors.
- FORCE_INLINE_FUNCTION uint8_t mixColors (uint8_t R, uint8_t G, uint8_t B, uint8_t bufpix, uint8_t alpha)

  Mix colors.

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 (uint8_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.

Additional Inherited Members

7.6.1 Detailed Description

The AbstractPainterABGR2222 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

7.6.2 Member Function Documentation

7.6.2.1 mixColors() [1/2]

```
FORCE_INLINE_FUNCTION uint8_t mixColors (
            uint8_t newpix,
            uint8_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

newpix	The newpix value.
bufpix	The bufpix value.
alpha	The alpha of the newpix.

Returns

The new color to write to the frame buffer.

7.6.2.2 mixColors() [2/2]

```
FORCE_INLINE_FUNCTION uint8_t mixColors (
            uint8_t R,
            uint8_t G,
            uint8_t B,
            uint8_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

R	The red color (0-255).
G	The green color (0-255).
B	The blue color (0-255).
bufpix	The bufpix value.
alpha	The alpha of the newpix.

Returns

The new color to write to the frame buffer.

7.6.2.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterABGR2222, and PainterABGR2222Bitmap.

7.6.2.4 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 PainterABGR2222Bitmap.

7.6.2.5 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterABGR2222, and PainterABGR2222Bitmap.

7.6.2.6 renderPixel()

```
void renderPixel (
            uint8_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 2,2,2.

Parameters

in	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.

7.7 AbstractPainterARGB2222 Class Reference

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

`#include <touchgfx/widgets/canvas/AbstractPainterARGB2222.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 uint8_t mixColors (uint8_t newpix, uint8_t bufpix, uint8_t alpha)

  Mix colors.
- FORCE_INLINE_FUNCTION uint8_t mixColors (uint8_t R, uint8_t G, uint8_t B, uint8_t bufpix, uint8_t alpha)

  Mix colors.

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 (uint8_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.

Additional Inherited Members

7.7.1 Detailed Description

The AbstractPainterARGB2222 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 mixColors() [1/2]

```
FORCE_INLINE_FUNCTION uint8_t mixColors (
            uint8_t newpix,
            uint8_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

newpix	The newpix value.
bufpix	The bufpix value.
alpha	The alpha of the newpix.

Returns

The new color to write to the frame buffer.

7.7.2.2 mixColors() [2/2]

```
FORCE_INLINE_FUNCTION uint8_t mixColors (
            uint8_t R,
            uint8_t G,
            uint8_t B,
```

```
            uint8_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

R	The red color (0-255).
G	The green color (0-255).
B	The blue color (0-255).
bufpix	The bufpix value.
alpha	The alpha of the newpix.

Returns

The new color to write to the frame buffer.

7.7.2.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterARGB2222, and PainterARGB2222Bitmap.

7.7.2.4 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 PainterARGB2222Bitmap.

7.7.2.5 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterARGB2222, and PainterARGB2222Bitmap.

7.7.2.6 renderPixel()

```
void renderPixel (
            uint8_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 2,2,2.

Parameters

in	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.

7.8 AbstractPainterARGB8888 Class Reference

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

`#include <touchgfx/widgets/canvas/AbstractPainterARGB8888.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.
- virtual void renderPixel (uint16_t *p, uint8_t red, uint8_t green, uint8_t blue, uint8_t alpha)

  Renders the pixel.

Protected Attributes

- int currentX

  Current x coordinate relative to the widget.
- int currentY

  Current y coordinate relative to the widget.

Additional Inherited Members

7.8.1 Detailed Description

The AbstractPainterARGB8888 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	*ptr*	Pointer to the row in the RenderingBuffer.

Parameters

	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterARGB8888, PainterARGB8888Bitmap, and PainterARGB8888L8Bitmap.

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 PainterARGB8888Bitmap, and PainterARGB8888L8Bitmap.

7.8.2.3 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterARGB8888, PainterARGB8888Bitmap, and PainterARGB8888L8Bitmap.

7.8.2.4 renderPixel() [1/2]

```
void renderPixel (
             uint16_t * p,
```

```
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [protected], [virtual]
```

Renders the pixel into the frame buffer.

Parameters

in	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.

7.8.2.5 renderPixel() [2/2]

```
void renderPixel (
            uint16_t * p,
            uint8_t red,
            uint8_t green,
            uint8_t blue,
            uint8_t alpha )  [protected], [virtual]
```

Renders the pixel into the frame buffer.

Parameters

in	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.
	alpha	The alpha.

7.9 AbstractPainterBGRA2222 Class Reference

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

`#include <touchgfx/widgets/canvas/AbstractPainterBGRA2222.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 uint8_t mixColors (uint8_t newpix, uint8_t bufpix, uint8_t alpha)

  Mix colors.
- FORCE_INLINE_FUNCTION uint8_t mixColors (uint8_t R, uint8_t G, uint8_t B, uint8_t bufpix, uint8_t alpha)

  Mix colors.

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 (uint8_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.

Additional Inherited Members

7.9.1 Detailed Description

The AbstractPainterBGRA2222 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 uint8_t mixColors (
            uint8_t newpix,
            uint8_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

newpix	The newpix value.
bufpix	The bufpix value.
alpha	The alpha of the newpix.

Returns

The new color to write to the frame buffer.

7.9.2.2 mixColors() [2/2]

```
FORCE_INLINE_FUNCTION uint8_t mixColors (
            uint8_t R,
```

```
            uint8_t G,
            uint8_t B,
            uint8_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

R	The red color (0-255).
G	The green color (0-255).
B	The blue color (0-255).
bufpix	The bufpix value.
alpha	The alpha of the newpix.

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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterBGRA2222, and PainterBGRA2222Bitmap.

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 PainterBGRA2222Bitmap.

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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterBGRA2222, and PainterBGRA2222Bitmap.

7.9.2.6 renderPixel()

```
void renderPixel (
            uint8_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 2,2,2.

Parameters

in	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.

7.10 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.

Additional Inherited Members

7.10.1 Detailed Description

AbstractPainterBW is used for drawing one 1bpp displays. The color is either on or off No transparency is supported.

See also

AbstractPainter

7.10.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterBW, and PainterBWBitmap.

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 PainterBWBitmap.

7.10.2.3 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.

7.11 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.

Additional Inherited Members

7.11.1 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

7.11.2 Member Function Documentation

7.11.2.1 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterGRAY2, and PainterGRAY2Bitmap.

7.11.2.2 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.

7.11.2.3 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	*gray*	The gray (0-15).
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterGRAY2, and PainterGRAY2Bitmap.

7.11.2.4 renderPixel()

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

Renders the pixel into the frame buffer.

Parameters

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

param blue The blue color.

7.12 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.

Additional Inherited Members

7.12.1 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

7.12.2 Member Function Documentation

7.12.2.1 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.

Parameters

	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterGRAY4, and PainterGRAY4Bitmap.

7.12.2.2 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.

7.12.2.3 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	*gray*	The gray (0-15).
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterGRAY4, and PainterGRAY4Bitmap.

7.12.2.4 renderPixel()

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

Renders the pixel into the frame buffer.

Parameters

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

param blue The blue color.

7.13 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, uint8_t alpha)

  Mix colors.
- FORCE_INLINE_FUNCTION uint16_t mixColors (uint16_t R, uint16_t G, uint16_t B, uint16_t bufpix, uint8_t alpha)

  Mix colors.

Static Public Attributes

- static const uint16_t RMASK = 0xF800

  Mask for red (1111100000000000)
- static const uint16_t GMASK = 0x07E0

  Mask for green (0000011111100000)
- static const uint16_t BMASK = 0x001F

  Mask for blue (0000000000011111)

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.

Additional Inherited Members

7.13.1 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

7.13.2 Member Function Documentation

7.13.2.1 mixColors() [1/2]

```
FORCE_INLINE_FUNCTION uint16_t mixColors (
            uint16_t newpix,
            uint16_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

newpix	The newpix value.
bufpix	The bufpix value.
alpha	The alpha of the newpix.

Returns

The new color to write to the frame buffer.

7.13.2.2 mixColors() [2/2]

```
FORCE_INLINE_FUNCTION uint16_t mixColors (
            uint16_t R,
            uint16_t G,
            uint16_t B,
            uint16_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

R	The red color (placed in 0xF800).
G	The green color (placed in 0x03E0).
B	The blue color (placed in 0x001F).
bufpix	The bufpix value.
alpha	The alpha of the newpix.

Returns

The new color to write to the frame buffer.

7.13.2.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterRGB565, PainterRGB565Bitmap, and PainterRGB565L8Bitmap.

7.13.2.4 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, and PainterRGB565L8Bitmap.

7.13.2.5 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterRGB565, PainterRGB565Bitmap, and PainterRGB565L8Bitmap.

7.13.2.6 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	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.

7.14 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.

Additional Inherited Members

7.14.1 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

7.14.2 Member Function Documentation

7.14.2.1 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterRGB888, PainterRGB888Bitmap, and PainterRGB888L8Bitmap.

7.14.2.2 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, and PainterRGB888L8Bitmap.

7.14.2.3 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterRGB888, PainterRGB888Bitmap, and PainterRGB888L8Bitmap.

7.14.2.4 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	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.

7.15 AbstractPainterRGBA2222 Class Reference

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

```
#include <touchgfx/widgets/canvas/AbstractPainterRGBA2222.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 uint8_t mixColors (uint8_t newpix, uint8_t bufpix, uint8_t alpha)

  Mix colors.
- FORCE_INLINE_FUNCTION uint8_t mixColors (uint8_t R, uint8_t G, uint8_t B, uint8_t bufpix, uint8_t alpha)

  Mix colors.

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 (uint8_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.

Additional Inherited Members

7.15.1 Detailed Description

The AbstractPainterRGBA2222 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

7.15.2 Member Function Documentation

7.15.2.1 mixColors() [1/2]

```
FORCE_INLINE_FUNCTION uint8_t mixColors (
            uint8_t newpix,
            uint8_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

newpix	The newpix value.
bufpix	The bufpix value.
alpha	The alpha of the newpix.

Returns

The new color to write to the frame buffer.

7.15.2.2 mixColors() [2/2]

```
FORCE_INLINE_FUNCTION uint8_t mixColors (
            uint8_t R,
            uint8_t G,
            uint8_t B,
            uint8_t bufpix,
            uint8_t alpha )  [inline]
```

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

Parameters

R	The red color (0-255).
G	The green color (0-255).
B	The blue color (0-255).
bufpix	The bufpix value.
alpha	The alpha of the newpix.

Returns

The new color to write to the frame buffer.

7.15.2.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).

Parameters

	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Implements AbstractPainter.

Reimplemented in PainterRGBA2222, and PainterRGBA2222Bitmap.

7.15.2.4 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 PainterRGBA2222Bitmap.

7.15.2.5 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implemented in PainterRGBA2222, and PainterRGBA2222Bitmap.

7.15.2.6 renderPixel()

```
void renderPixel (
             uint8_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 2,2,2.

Parameters

in	*p*	pointer into the frame buffer where the given color should be written.
	red	The red color.
	green	The green color.
	blue	The blue color.

7.16 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.

7.16.1 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.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 ~AbstractPartition()

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

Virtual destructor.

7.16.2.2 AbstractPartition()

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

Default constructor.

7.16.3 Member Function Documentation

7.16.3.1 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

size	The size.

Returns

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

7.16.3.2 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

T	Generic type parameter.

Returns

The address of an empty storage slot.

7.16.3.3 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

index	Zero-based index of the.
size	The size.

Returns

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

7.16.3.4 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

T	Generic type parameter.

Parameters

index	Zero-based index of the.

Returns

The address of the appropriate storage slot.

7.16.3.5 at() [1/2]

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

Gets the object at the specified index.

Template Parameters

T	Generic type parameter.

Parameters

index	The index into the Partition storage where the returned object is located.

Returns

A typed reference to the object at the specified index.

7.16.3.6 at() [2/2]

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

const version of at().

Template Parameters

T	Generic type parameter.

Parameters

index	Zero-based index of the.

Returns

A T&

7.16.3.7 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 >.

7.16.3.8 clear()

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

Prepares the Partition for new allocations. Any objects present in the Partition shall not be used after invoking this method.

7.16.3.9 dec()

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

Decreases number of allocations.

7.16.3.10 element() [1/2]

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

Access to stored element. Used internally.

Parameters

index	Zero-based index of the.

Returns

null if it fails, else a void*.

Implemented in Partition< ListOfTypes, NUMBER_OF_ELEMENTS >.

7.16.3.11 element() [2/2]

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

Access to stored element, const version.

Parameters

index	Zero-based index of the.

Returns

null if it fails, else a void∗.

Implemented in Partition< ListOfTypes, NUMBER_OF_ELEMENTS >.

7.16.3.12 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 >.

7.16.3.13 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

T	Generic type parameter.

Parameters

pT	Pointer to the object to lookup.

Returns

If the object seems to be allocated in the Partition, a Pair object containing a typed pointer to the object and an index into the Partition storage is returned. Otherwise, a Pair< 0, 0 > is returned.

7.16.3.14 getAllocationCount()

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

Gets allocation count.

Returns

The currently allocated storage slots.

7.16.3.15 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

address	The location address to lookup.

Returns

An uint16_t.

7.17 AbstractProgressIndicator Class Reference

An abstract progress indicator.

`#include <touchgfx/containers/progress_indicators/AbstractProgressIndicator.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

7.17.1 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

BoxProgess
CircleProgress
ImageProgress
LineProgress
TextProgress

7.17.2 Constructor & Destructor Documentation

7.17.2.1 AbstractProgressIndicator()

```
AbstractProgressIndicator ( )
```

Initialized the progress indicator with default range 0-100.

See also

setRange

7.17.2.2 ~AbstractProgressIndicator()

```
~AbstractProgressIndicator ( )  [virtual]
```

Destructor.

7.17.3 Member Function Documentation

7.17.3.1 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

range	The range.

Returns

The progress.

See also

setRange, setValue

7.17.3.2 getProgressIndicatorHeight()

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

Gets progress indicator height.

Returns

The progress indicator height.

7.17.3.3 getProgressIndicatorWidth()

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

Gets progress indicator width.

Returns

The progress indicator width.

7.17.3.4 getProgressIndicatorX()

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

Gets progress indicator x coordinate.

Returns

The progress indicator x coordinate.

7.17.3.5 getProgressIndicatorY()

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

Gets progress indicator y coordinate.

Returns

The progress indicator y coordinate.

7.17.3.6 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	*min*	The minimum input value.
out	*max*	The maximum input value.
out	*steps*	The steps in which to report progress.
out	*minStep*	The step which the minimum input value is mapped to.

7.17.3.7 getRange() [2/3]

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

Gets the range set by setRange().

Parameters

out	*min*	The minimum input value.
out	*max*	The maximum input value.
out	*steps*	The steps in which to report progress.

7.17.3.8 getRange() [3/3]

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

Gets the range set by setRange().

Parameters

out	*min*	The minimum input value.
out	*max*	The maximum input value.

7.17.3.9 getValue()

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

Gets the current value set by setValue().

Returns

The value.

7.17.3.10 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

bmpBackground	The background bitmap.

7.17.3.11 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

x	The x coordinate.
y	The y coordinate.
width	The width of the box progress indicator.
height	The height of the box progress indicator.

Reimplemented in LineProgress, CircleProgress, BoxProgress, ImageProgress, and TextProgress.

7.17.3.12 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

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

See also

setValue, getProgress

7.17.3.13 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

value	The value.

Reimplemented in CircleProgress, LineProgress, ImageProgress, TextProgress, and BoxProgress.

7.18 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 setShape (const 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 getAngle (T &angle)

  Gets the abstractShape's angle.
- 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 updateScale (T newXScale, T newYScale)

  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

7.18.1 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.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 AbstractShape()

```
AbstractShape ( )
```

Constructs a new AbstractShape.

7.18.2.2 ~AbstractShape()

```
~AbstractShape ( )  [virtual]
```

Virtual Destructor.

7.18.3 Member Function Documentation

7.18.3.1 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

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Returns

true if it succeeds, false if it fails.

Implements CanvasWidget.

7.18.3.2 getAngle() [1/2]

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

Gets the abstractShape's angle.

Template Parameters

T	Generic type parameter.

Parameters

out	*angle*	The current abstractShape angle relative to 0.

7.18.3.3 getAngle() [2/2]

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

Gets the current angle of the abstractShape.

Returns

The angle of the AbstractShape.

7.18.3.4 getCacheX()

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

Gets cached x coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The cached x coordinate.

Implemented in Shape< POINTS >.

7.18.3.5 getCacheY()

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

Gets cached y coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The cached y coordinate.

Implemented in Shape< POINTS >.

7.18.3.6 getCornerX()

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

Gets the x coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The corner x coordinate.

Implemented in Shape< POINTS >.

7.18.3.7 getCornerY()

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

Gets the y coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The corner y coordinate.

Implemented in Shape< POINTS >.

7.18.3.8 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.

7.18.3.9 getNumPoints()

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

Gets number points used to make up the shape.

Returns

The number points.

Implemented in Shape< POINTS >.

7.18.3.10 getOrigin()

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

Gets the start coordinates for the line.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*dx*	The x coordinate.
out	*dy*	The y coordinate.

7.18.3.11 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

T	Generic type parameter, either int or float.

Parameters

out	*x*	Scaling of x coordinates.
out	*y*	Scaling of y coordinates.

See also

setScale

7.18.3.12 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

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the shapes position (0,0).
y	The y coordinate of the shapes position (0,0).

Note

The area containing the AbstractShape is invalidated before and after the change.

See also

setOrigin()

7.18.3.13 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

T	Generic type parameter.

Parameters

angle	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()

7.18.3.14 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	Zero-based index of the corner.
x	The x coordinate.
y	The y coordinate.

Implemented in Shape< POINTS >.

7.18.3.15 setCorner()

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

Sets a corner of the shape in Q5 format.

Parameters

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

Note

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

See also

updateAbstractShapeCache

Implemented in Shape< POINTS >.

7.18.3.16 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

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the shapes position (0,0).
y	The y coordinate of the shapes position (0,0).

Note

The area containing the AbstractShape is not invalidated.

See also

moveOrigin()

7.18.3.17 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

T	Generic type parameter, either int or float.

Parameters

newXScale	The new scale in the x direction.
newYScale	The new scale in the y direction.

Note

The area containing the AbstractShape is not invalidated.

See also

getScale, updateScale

7.18.3.18 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

T	Generic type parameter, either int or float.

Parameters

scale	The scale in the x direction.

Note

The area containing the AbstractShape is not invalidated.

See also

getScale

7.18.3.19 setShape() [1/2]

```
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

T	Generic type parameter, either int or float.

Parameters

in	*points*	The points that make up the shape.

Note

The area containing the shape is not invalidated.

7.18.3.20 setShape() [2/2]

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

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

Template Parameters

T	Generic type parameter, either int or float.

Parameters

in	*points*	The points that make up the shape.

Note

The area containing the shape is not invalidated.

7.18.3.21 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).

7.18.3.22 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

T	Generic type parameter.

Parameters

angle	The angle to turn the abstractShape.

Note

The area containing the AbstractShape is invalidated before and after the change.

See also

setAngle()

7.18.3.23 updateScale()

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

Scale the AbstractShape the given amounts in the x direction and the y direction.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

newXScale	The new scale in the x direction.
newYScale	The new scale in the y direction.

Note

The area containing the AbstractShape is invalidated before and after the change.

See also

setScale()

7.19 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

7.19.1 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())

7.19.2 Member Function Documentation

7.19.2.1 animationEnabled()

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

Is animation enabled.

Returns

true if animation is enabled.

7.19.2.2 convertHandValueToAngle()

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

Convert hand value to angle.

Parameters

steps	Number of steps the primary hand value is divided into (e.g. 60 for minutes/seconds and 12 for hour).
handValue	The actual value for the hand in question (in the range [0;steps]).
secondHandValue	(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 setHourHandMinuteCorrection(true) or setMinuteHandSecondCorrection(true) is selected.

Returns

The converted value to angle.

7.19.2.3 getAnimationDuration()

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

Returns

The animation duration.

7.19.2.4 getHourHandMinuteCorrection()

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

Gets hour hand minute correction.

Returns

true if hour hand minute correction is active.

See also

setHourHandMinuteCorrection

7.19.2.5 getMinuteHandSecondCorrection()

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

Gets minute hand second correction.

Returns

true if minute hand second correction is active.

See also

setHourHandMinuteCorrection

7.19.2.6 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

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

7.19.2.7 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

hour	The hours (in 24H format).
minute	The minutes (in 24H format).
second	The seconds (in 24H format).

7.19.2.8 setAnimation()

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

[virtual]

Parameters

duration	(Optional) The animation duration.
animationProgressionEquation	(Optional) The animation progression equation.

7.19.2.9 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

backgroundBitmapId	Identifier for the background bitmap.

7.19.2.10 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

backgroundBitmapId	Identifier for the background bitmap.
rotationCenterX	The rotation center x coordinate.
rotationCenterY	The rotation center y coordinate.

7.19.2.11 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

active	true to use hour hand correction.

See also

getHourHandMinuteCorrection

7.19.2.12 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

active	true to use.

See also

setMinuteHandSecondCorrection

7.19.2.13 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

rotationCenterX	The rotation center x coordinate.
rotationCenterY	The rotation center y coordinate.

7.19.2.14 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	*hand*	Reference to the hand being setup.
	bitmapId	The bitmap identifier for the given hand.
	rotationCenterX	The hand rotation center x coordinate.
	rotationCenterY	The hand rotation center y coordinate.

7.19.2.15 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

hourHand↩ BitmapId	Identifier for the hour hand bitmap.
rotationCenterX	The hand rotation center x coordinate.
rotationCenterY	The hand rotation center y coordinate.

7.19.2.16 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

minuteHand↩ BitmapId	Identifier for the minute hand bitmap.
rotationCenterX	The hand rotation center x coordinate.
rotationCenterY	The hand rotation center y coordinate.

7.19.2.17 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

secondHand↩ BitmapId	Identifier for the second hand bitmap.
rotationCenterX	The hand rotation center x coordinate.
rotationCenterY	The hand rotation center y coordinate.

7.19.2.18 updateClock()

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

Updates the visual representation of the clock.

Implements AbstractClock.

7.20 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

7.20.1 Detailed Description

The AnimatedImage is capable of running the animation from start to end or in reverse order, end to start. It is capable doing a single animation or looping the animation until stopped or paused. See animation_example far a demonstration of how to use of this widget.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 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

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

7.20.2.2 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 set setBitmaps function before this widget displays anything.

Parameters

updateInterval	Defines the number of ticks between each animation step. Higher value results in a slower animation.

7.20.3 Member Function Documentation

7.20.3.1 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_ANIMATEDIMAGE.

Reimplemented from Image.

7.20.3.2 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.

7.20.3.3 isAnimatedImageRunning()

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

Gets the running state of the AnimatedImage.

Returns

true if the animation is currently running, false otherwise.

7.20.3.4 isReverse()

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

Query if this object is running in reverse.

Returns

true if the animation is performed in reverse order.

7.20.3.5 isRunning()

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

Gets the running state of the AnimatedImage.

Returns

true if the animation is currently running, false otherwise.

7.20.3.6 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.

7.20.3.7 setBitmap()

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

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

Parameters

bmp	The bitmap.

Reimplemented from Image.

7.20.3.8 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

start	Defines the start of the range of images in the animation.
end	Defines the end of the range of images in the animation.

7.20.3.9 setDoneAction()

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

Associates an action to be performed when the animation of the AnimatedImage is done.

Parameters

callback	The callback is executed when done. The callback is given the animated image.

7.20.3.10 setUpdateTicksInterval()

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

Sets the update interval.

Parameters

updateInterval	Defines the number of ticks between each animation step. Higher value results in a slower animation.

7.20.3.11 startAnimation()

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

Starts the animation.

Parameters

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

7.20.3.12 stopAnimation()

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

Stops and resets the animation.

7.21 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.

7.21.1 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

T	Generic type parameter. Typically a AbstractButtonContainer subclass.

See also

AbstractButtonContainer

7.21.2 Member Function Documentation

7.21.2.1 setBitmaps()

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

Parameters

bmpStart	The bitmap start.
bmpEnd	The bitmap end.

7.21.2.2 setBitmapXY()

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

Parameters

x	An uint16_t to process.
y	An uint16_t to process.

7.21.2.3 setUpdateTicksInterval()

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

Parameters

updateInterval	The update interval.

7.22 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.

7.23 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 Types

- enum AnimationState { ANIMATION_FINISHED = 0, ANIMATION_DELAYED, ANIMATION_RUNNING }

  Values that represent different states during an animation.

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

7.23.1 Detailed Description

A texture mapper with animation capabilities. Note that the angles of the TextureMapper is moved to the [0; 2PI] 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

7.23.2 Member Enumeration Documentation

7.23.2.1 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.

7.23.2.2 AnimationState

```
enum AnimationState  [protected]
```

Enumerator

ANIMATION_FINISHED	The animation is finished.
ANIMATION_DELAYED	The animation is in the delay mode.
ANIMATION_RUNNING	The animation is currently runnnig.

7.23.3 Constructor & Destructor Documentation

7.23.3.1 AnimationTextureMapper()

```
AnimationTextureMapper ( )
```

Default constructor.

7.23.3.2 ~AnimationTextureMapper()

```
~AnimationTextureMapper ( )  [virtual]
```

Destructor. Destroys the AnimationTextureMapper.

7.23.4 Member Function Documentation

7.23.4.1 cancelAnimationTextureMapperAnimation()

```
void cancelAnimationTextureMapperAnimation ( )  [virtual]
```

Cancel move animation.

7.23.4.2 getAnimationStep()

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

Returns

The current animation step.

7.23.4.3 handleTickEvent()

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

The tick handler that handles the actual animation steps.

Reimplemented from Drawable.

7.23.4.4 isTextureMapperAnimationRunning()

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

Gets whether or not the animation is running.

Returns

true if the animation is running.

7.23.4.5 setTextureMapperAnimationEndedAction()

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

Associates an action to be performed when the animation ends.

Parameters

callback	The callback to be executed. The callback will be given a reference to the AnimationTextureMapper.

See also

GenericCallback

7.23.4.6 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

callback	The callback to be executed. The callback will be given a reference to the AnimationTextureMapper.

See also

GenericCallback

7.23.4.7 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

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

7.23.4.8 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.

7.24 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 (const 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 void setDebugPrinter (DebugPrinter *printer)

  Sets the DebugPrinter object to be used by the application.
- static DebugPrinter * getDebugPrinter ()

  Returns the DebugPrinter object associated with the application.
- static void setDebugString (const char *string)

  Sets the debug string to be displayed onscreen.

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 ()

  Procected constructor.

Protected Attributes

- Vector< Drawable *, MAX_TIMER_WIDGETS > timerWidgets

  List of widgets that receive timer ticks.
- uint8_t timerWidgetCounter [MAX_TIMER_WIDGETS]

  A counter for each potentially registered timer widget. Increase when registering for timer events, decrease when unregistering.
- Vector< Rect, 8 > cachedDirtyAreas

  When draw caching is enabled, these rects keeps track of the dirty screen area.
- Vector< Rect, 8 > lastRects

The dirty areas from last frame that needs to be redrawn because we have swapped frame buffers.

- bool drawCacheEnabled

  True when draw caching is active.

- bool transitionHandled

  True if the transition is done and Screen::afterTransition has been called.

Static Protected Attributes

- static Screen * currentScreen

  Pointer to currently displayed Screen.

- static Transition * currentTransition

  Pointer to current transition.

- static Application * instance
- static DebugPrinter * debugPrinter

  Pointer to the DebugPrinter instance.

7.24.1 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

7.24.2 Constructor & Destructor Documentation

7.24.2.1 Application()

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

Procected constructor.

7.24.3 Member Function Documentation

7.24.3.1 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

screenId	An id that maps to the desired screen.

7.24.3.2 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

enableCache	if true, all future draw operations will be cached. If false draw caching is disabled, and the current cache (if not empty) is drawn immediately.

7.24.3.3 clearAllTimerWidgets()

```
void clearAllTimerWidgets ( )
```

Clears all currently registered timer widgets.

7.24.3.4 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.

7.24.3.5 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	*rect*	The area to draw.

7.24.3.6 getCurrentScreen()

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

Gets the current screen.

Returns

The current screen.

7.24.3.7 getDebugPrinter()

```
static const DebugPrinter * getDebugPrinter ( )  [inline], [static]
```

Returns the DebugPrinter object associated with the application.

Returns

DebugPrinter The DebugPrinter object.

7.24.3.8 getInstance()

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

Gets the single instance application.

Returns

The instance of this application.

7.24.3.9 getNumberOfRegisteredTimerWidgets()

```
uint16_t getNumberOfRegisteredTimerWidgets ( ) const
```

gets the number of timer widgets that has been registered.

Returns

The size of timerWidgets.

7.24.3.10 getTimerWidgetCountForDrawable()

```
uint16_t getTimerWidgetCountForDrawable (
            const 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

w	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.

7.24.3.11 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

evt	The ClickEvent.

Reimplemented from UIEventListener.

7.24.3.12 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

evt	The drag event, expressed in absolute coordinates.

Reimplemented from UIEventListener.

7.24.3.13 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

evt	The gesture event.

Reimplemented from UIEventListener.

7.24.3.14 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

c	The incomming character to handle.

Reimplemented from UIEventListener.

7.24.3.15 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.

7.24.3.16 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.

7.24.3.17 invalidateArea()

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

Invalidates this area.

Parameters

area	The area to invalidate.

7.24.3.18 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	*w*	The widget to add.

See also

unregisterTimerWidget

7.24.3.19 setDebugPrinter()

```
static void setDebugPrinter (
            DebugPrinter * printer )  [inline], [static]
```

Sets the DebugPrinter object to be used by the application to print debug messages.

Parameters

in	*printer*	The debug printer to configure.

7.24.3.20 setDebugString()

```
static void setDebugString (
            const char * string )  [inline], [static]
```

Sets the debug string to be displayed onscreen on top of the framebuffer.

Parameters

in	*string*	The debug string to display onscreen.

7.24.3.21 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	*newScreen*	A pointer to the new screen.

7.24.3.22 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	*w*	The widget to remove.

7.24.4 Member Data Documentation

7.24.4.1 instance

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

Pointer to the instance of the Application-derived subclass.

Note

Must be set by subclass constructor!

7.24.4.2 MAX_TIMER_WIDGETS

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

Remarks

Memory impact: x * (sizeof(Drawable*)+1)

7.25 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 ClutFormat { CLUT_FORMAT_L8_ARGB8888, CLUT_FORMAT_L8_RGB888, CLUT_FORMAT_L8_RGB565 }

  Color data of a clut can be stored in the following formats.
- enum BitmapFormat {
  RGB565, RGB888, ARGB8888, BW, BW_RLE, GRAY2, GRAY4,
  ARGB2222, ABGR2222, RGBA2222, BGRA2222, L8 }

  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.
- const uint8_t * getExtraData () const

  Gets a pointer to the extra (alpha) data, if present in the bitmap.
- BitmapFormat getFormat () const

  Gets the format of how the bitmap is stored.
- uint16_t getWidth () const

  Gets the width of the Bitmap in pixels.
- uint16_t getHeight () const

  Gets the height of the Bitmap in pixels.
- Rect getRect () const

  Gets the rectangle describing the dimensions of the Bitmap.
- bool isAlphaPerPixel () const

  Query if this object has an alpha channel.
- Rect getSolidRect () const

  Gets the largest solid rectangle in the bitmap.
- bool hasTransparentPixels () const

  Query if this object has transparent pixels.
- bool operator== (const Bitmap &other) const

  Equality operator.
- bool operator!= (const Bitmap &other) const

  Inequality operator.

Static Public Member Functions

- static void registerBitmapDatabase (const BitmapData *data, const uint16_t n, uint16_t *cachep=0, uint32_t csize=0, uint32_t numberOfDynamicBitmaps=0)

  Registers an array of bitmaps.
- static bool cache (BitmapId id)

  Cache this bitmap into RAM.
- static bool cacheReplaceBitmap (BitmapId out, BitmapId in)

  Replace a bitmap in RAM with another Bitmap.
- static bool cacheRemoveBitmap (BitmapId id)

  Remove this bitmap from the RAM cache.
- static uint8_t * cacheGetAddress (BitmapId id)

  Get address of cache buffer for this bitmap.
- static bool cacheIsCached (BitmapId id)

  Check if the Bitmap is cached.
- static bool cacheAll ()

  Cache all bitmaps from the Bitmap Database into RAM.
- static void clearCache ()

  Clears the cached bitmaps from RAM.

- static BitmapId dynamicBitmapCreate (const uint16_t width, const uint16_t height, BitmapFormat format, ClutFormat clutFormat=CLUT_FORMAT_L8_ARGB8888)

  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 bool dynamicBitmapAddSolidRect (BitmapId id, const Rect &solidRect)

  Updates 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.
- static void removeCache ()

  Removes the bitmap cache.
- static uint8_t * getCacheTopAddress ()

  Gets the address of the first unused memory in the cache.
- static void compactCache ()

  Compact the bitmap cache to get continuous free memory on top.

7.25.1 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.

7.25.2 Member Enumeration Documentation

7.25.2.1 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
ARGB2222	8-bit color
ABGR2222	8-bit color
RGBA2222	8-bit color
BGRA2222	8-bit color
L8	8-bit indexed color

7.25.2.2 ClutFormat

```
enum ClutFormat
```

Color data of a clut can be stored in the following formats.

Enumerator

CLUT_FORMAT_L8_ARGB8888	32-bit, 8 bits for each of red, green, blue and per pixel alpha channel
CLUT_FORMAT_L8_RGB888	24-bit, 8 bits for each of red, green and blue, no per pixel alpha channel
CLUT_FORMAT_L8_RGB565	16-bit, 5 bits for red, 6 bits for green, 5 bits for blue, no per pixel alpha channel

7.25.3 Constructor & Destructor Documentation

7.25.3.1 Bitmap()

```
Bitmap (
            const BitmapId id = BITMAP_INVALID )  [inline], [explicit]
```

Creates and binds a Bitmap instance to the corresponding entry in the BitmapData array.

Parameters

id	The unique bitmap identifier.

7.25.4 Member Function Documentation

7.25.4.1 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

id	The id of the bitmap to cache.

Returns

true if caching went well, false otherwise.

See also

registerBitmapDatabase

7.25.4.2 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 where cached.

See also

cache.

7.25.4.3 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

id	The id of the bitmap in cache.

Returns

Address if bitmap was found, zero otherwise.

7.25.4.4 cacheIsCached()

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

Check if the Bitmap is cached.

Parameters

id	The id of the bitmap.

Returns

true if bitmap is cached.

7.25.4.5 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

id	The id of the bitmap to cache.

Returns

true if bitmap was found and removed, false otherwise.

See also

registerBitmapDatabase

7.25.4.6 cacheReplaceBitmap()

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

Replace a bitmap in RAM with another Bitmap. The Bitmaps must have same size.

Parameters

out	The id of the bitmap to remove from the cache.
in	The id of the bitmap to cache.

Returns

true if the replacement went well, false otherwise.

7.25.4.7 clearCache()

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

Clears the cached bitmaps from RAM.

7.25.4.8 compactCache()

```
static void compactCache ( )  [static]
```

Compact the bitmap cache to get continuous free memory on top.

Note

This method is called by Bitmap::cache when required.

7.25.4.9 dynamicBitmapAddSolidRect()

```
static bool dynamicBitmapAddSolidRect (
             BitmapId id,
             const Rect & solidRect )  [static]
```

Updates the solid rectangle of a dynamic bitmap to include the given rectangle. Only relevant for ARGB8888 bitmaps and 8bpp bitmap formats, as these formats include an alpha channel. The solid part of the bitmap is drawn faster than the transparent parts.

Parameters

id	The identifier.
solidRect	The solid rectangle.

Returns

true if it succeeds, false if it fails.

7.25.4.10 dynamicBitmapCreate()

```
static BitmapId dynamicBitmapCreate (
             const uint16_t width,
             const uint16_t height,
             BitmapFormat format,
             ClutFormat clutFormat = CLUT_FORMAT_L8_ARGB8888 )  [static]
```

Create a dynamic bitmap. The clutFormat parameter is ignored for bitmaps not in L8 format.

Parameters

width	Width of the bitmap.
height	Height of the bitmap.
format	Bitmap format of the bitmap.
clutFormat	Color lookup table format of the bitmap.

Returns

BitmapId of the new bitmap or BITMAP_INVALID if cache memory is 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

7.25.4.11 dynamicBitmapDelete()

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

Delete a dynamic bitmap.

Parameters

id	The BitmapId of the dynamic bitmap.

Returns

true if it succeeds, false if it fails.

7.25.4.12 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

id	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.

7.25.4.13 dynamicBitmapSetSolidRect()

```
static bool dynamicBitmapSetSolidRect (
```

```
BitmapId id,
const Rect & solidRect )  [static]
```

Set the solid rectangle of a dynamic bitmap. Only relevant for ARGB8888 bitmaps and 8bpp bitmap formats, as these formats include an alpha channel. The solid part of the bitmap is drawn faster than the transparent parts.

Parameters

id	The identifier.
solidRect	The solid rectangle.

Returns

true if it succeeds, false if it fails.

7.25.4.14 getAlphaData()

```
const uint8_t * getAlphaData ( ) const
```

Gets a pointer to the alpha data, if present in the bitmap. For images stored in L8 format, a pointer to the CLUT will be returned. For non-opaque RGB565 images, a pointer to the alpha channel will be returned.

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 or CLUT. If no alpha channel or CLUT exist for the given Bitmap, 0 is returned.

See also

getExtraData

7.25.4.15 getCacheTopAddress()

```
static void getCacheTopAddress ( )  [inline], [static]
```

Gets the address of the first unused memory in the cache. Can be used in advanced application to reduce power consumption of external RAM by turning off unused RAM.

Returns

Returns the highest used address in the cache.

7.25.4.16 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.

7.25.4.17 getExtraData()

```
const uint8_t * getExtraData ( ) const
```

Gets a pointer to the extra (alpha) data, if present in the bitmap. For images stored in L8 format, a pointer to the CLUT will be returned. For non-opaque RGB565 images, a pointer to the alpha channel will be returned.

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 or CLUT. If no alpha channel or CLUT exist for the given Bitmap, 0 is returned.

7.25.4.18 getFormat()

```
BitmapFormat getFormat ( ) const
```

Gets the format of how the bitmap is stored.

Returns

The format of how the bitmap data is stored.

7.25.4.19 getHeight()

```
uint16_t getHeight ( ) const
```

Gets the height of the Bitmap in pixels.

Returns

The bitmap height in pixels.

7.25.4.20 getId()

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

Gets the id of this Bitmap.

Returns

The id of this Bitmap.

7.25.4.21 getRect()

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

Gets the rectangle describing the dimensions of the Bitmap.

Returns

a Rect describing the dimensions of this bitmap.

7.25.4.22 getSolidRect()

```
Rect getSolidRect ( ) const
```

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

Returns

The maximum solid rectangle of the bitmap.

7.25.4.23 getWidth()

```
uint16_t getWidth ( ) const
```

Gets the width of the Bitmap in pixels.

Returns

The bitmap width in pixels.

7.25.4.24 hasTransparentPixels()

```
bool hasTransparentPixels ( ) const
```

Returns

True if this bitmap has transparent pixels.

7.25.4.25 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)

7.25.4.26 operator"!=()

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

Inequality operator.

Parameters

other	The bitmap to compare with.

Returns

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

7.25.4.27 operator==()

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

Equality operator.

Parameters

other	The bitmap to compare with.

Returns

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

7.25.4.28 registerBitmapDatabase()

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

Registers an array of bitmaps. All Bitmap instances are bound to this database.

Parameters

	data	A reference to the BitmapData storage array.
	n	The number of bitmaps in the array.
in,out	*cachep*	(Optional) Pointer to memory region in which bitmap data can be cached.
	csize	Size of cache memory region in bytes (0 if unused)
	numberOfDynamicBitmaps	Number of dynamic bitmaps to be allowed in the cache.

7.25.4.29 removeCache()

```
static void removeCache ( )  [static]
```

Removes the bitmap cache. The memory can hereafter be used for other purposes. All dynamic bitmap IDs are not valid after this.

7.25.4.30 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	*cachep*	Pointer to memory region in which bitmap data can be cached.
	csize	Size of cache memory region in bytes.
	numberOfDynamicBitmaps	Number of dynamic bitmaps to be allowed in the cache.

7.26 Bitmap::BitmapData Struct Reference

Data of a bitmap.

`#include <touchgfx/Bitmap.hpp>`

Public Member Functions

- BitmapFormat getFormat () const

  Gets the format.

Public Attributes

- const uint8_t *const data

  The data of this bitmap.
- const uint8_t *const extraData

  The data of either the alpha channel if exist or clut data in case of indexed color bitmap (contains 0 if no alpha channel neither clut 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: 13

  The width of the maximum solid rectangle of the bitmap.
- const uint16_t format_hi: 3

  Determine the format of the data (high 3 bits)

- const uint16_t solidRect_height: 13

  The height of the maximum solid rectangle of the bitmap.
- const uint16_t format_lo: 3

  Determine the format of the data (low 3 bits)

7.26.1 Detailed Description

Data of a bitmap.

7.26.2 Member Function Documentation

7.26.2.1 getFormat()

```
BitmapFormat getFormat ( ) const  [inline]
```

Gets the format by combining the high and low parts (format_hi << 3) | format_lo

Returns

The format.

7.27 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 (pixels or indexes)
- const uint8_t * pClut

  Pointer to the source CLUT entires.
- 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.

- uint8_t srcFormat

  The source format.
- uint8_t dstFormat

  The destination format.

7.27.1 Detailed Description

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

7.27.2 Member Data Documentation

7.27.2.1 dstFormat

```
uint8_t dstFormat
```

See also

BitmapFormat

7.27.2.2 operation

```
uint32_t operation
```

See also

BlitOperations

7.27.2.3 srcFormat

```
uint8_t srcFormat
```

See also

BitmapFormat

7.28 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

7.28.1 Detailed Description

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

See also

Widget

7.28.2 Constructor & Destructor Documentation

7.28.2.1 Box() [1/2]

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

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

7.28.2.2 Box() [2/2]

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

Construct a Box.

Parameters

width	The width of the box.
height	The height of the box.
color	The color of the box.
alpha	The alpha of the box. Default is 255 (solid).

7.28.2.3 ~Box()

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

Destructor.

7.28.3 Member Function Documentation

7.28.3.1 draw()

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

Draws the Box.

Parameters

area	The rectangle to draw, with coordinates relative to the containing layer.

Implements Drawable.

7.28.3.2 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

solid	true if this Box should report as solid, even when not.

7.28.3.3 getAlpha()

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

Returns

Gets the current alpha value of the Box.

7.28.3.4 getColor()

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

Gets the current color of the Box.

Returns

The current color.

7.28.3.5 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.

7.28.3.6 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_BOX.

Reimplemented from Widget.

7.28.3.7 setAlpha()

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

Sets the alpha value for this Box.

Parameters

alpha	The alpha value. 255 = completely solid.

7.28.3.8 setColor()

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

Sets the color of the rectangle.

Parameters

color	The color of the box.

7.28.4 Member Data Documentation

7.28.4.1 reportAsSolid

```
bool reportAsSolid  [protected]
```

See also

forceReportAsSolid.

7.29 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

7.29.1 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

7.29.2 Constructor & Destructor Documentation

7.29.2.1 BoxProgress()

```
BoxProgress ( )
```

Default constructor.

7.29.2.2 ~BoxProgress()

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

Destructor.

7.29.3 Member Function Documentation

7.29.3.1 getAlpha()

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

Gets the alpha of the Box.

Returns

The alpha.

See also

Box

7.29.3.2 getColor()

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

Gets the color of the Box.

Returns

The color.

See also

Box

7.29.3.3 setAlpha()

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

Sets the alpha of the Box.

Parameters

alpha	The alpha.

See also

Box

7.29.3.4 setColor()

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

Sets the color of the Box.

Parameters

color	The color.

See also

Box

7.29.3.5 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

x	The x coordinate.
y	The y coordinate.
width	The width of the box progress indicator.
height	The height of the box progress indicator.

Reimplemented from AbstractProgressIndicator.

7.29.3.6 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

value	The value.

Reimplemented from AbstractProgressIndicator.

7.30 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

7.30.1 Constructor & Destructor Documentation

7.30.1.1 BoxWithBorder()

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

Parameters

width	The width.
height	The height.
color	The color.
borderColor	The border color.
borderSize	Size of the border.
alpha	(Optional) The alpha.

7.30.2 Member Function Documentation

7.30.2.1 draw()

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

Parameters

area	The area.

Implements Drawable.

7.30.2.2 getAlpha()

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

Returns

The alpha.

7.30.2.3 getBorderColor()

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

Returns

The border color.

7.30.2.4 getBorderSize()

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

Returns

The border size.

7.30.2.5 getColor()

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

Returns

The color.

7.30.2.6 getSolidRect()

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

Returns

The solid rectangle.

Implements Drawable.

7.30.2.7 getType()

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

Returns

The type.

Reimplemented from Widget.

7.30.2.8 setAlpha()

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

Parameters

alpha	The alpha.

7.30.2.9 setBorderColor()

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

Parameters

color	The color.

7.30.2.10 setBorderSize()

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

Parameters

size	The size.

7.30.2.11 setColor()

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

Parameters

color	The color.

7.31 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.

7.31.1 Detailed Description

template<class T>
class touchgfx::BoxWithBorderButtonStyle< 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

T	Generic type parameter. Typically a AbstractButtonContainer subclass.

See also

AbstractButtonContainer, BoxWithBorder

7.31.2 Member Function Documentation

7.31.2.1 setBorderSize()

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

Parameters

size	The size.

7.31.2.2 setBoxWithBorderColors()

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

Parameters

colorReleased	The color released.
colorPressed	The color pressed.
borderColorReleased	The border color released.
borderColorPressed	The border color pressed.

7.31.2.3 setBoxWithBorderHeight()

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

Parameters

height	The height.

7.31.2.4 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

x	The x coordinate of this BoxWithBorderButtonStyle.
y	The y coordinate of this BoxWithBorderButtonStyle.
width	The width of this BoxWithBorderButtonStyle.
height	The height of this BoxWithBorderButtonStyle.

7.31.2.5 setBoxWithBorderWidth()

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

Parameters

width	The width.

7.32 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

7.32.1 Detailed Description

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

See also

AbstractButton

7.32.2 Constructor & Destructor Documentation

7.32.2.1 Button()

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

Default constructor.

7.32.2.2 ~Button()

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

Destructor.

7.32.3 Member Function Documentation

7.32.3.1 draw()

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

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Implements Drawable.

Reimplemented in ButtonWithLabel, and ButtonWithIcon.

7.32.3.2 getAlpha()

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

Gets the current alpha value.

Returns

The current alpha value.

7.32.3.3 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.

7.32.3.4 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.

7.32.3.5 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.

7.32.3.6 setAlpha()

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

Sets the alpha value for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

7.32.3.7 setBitmaps()

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

Sets the bitmaps used by this button.

Parameters

bmpReleased	Bitmap to use when button is released.
bmpPressed	Bitmap to use when button is pressed.

Reimplemented in ToggleButton.

7.33 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.

7.33.1 Detailed Description

Interface for sampling external key events.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 ~ButtonController()

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

Destructor.

7.33.3 Member Function Documentation

7.33.3.1 init()

```
void init ( )  [pure virtual]
```

Initializes button controller.

7.33.3.2 reset()

```
void reset ( )  [inline], [virtual]
```

Resets button controller. Does nothing in the default implementation.

7.33.3.3 sample()

```
bool sample (
            uint8_t & key )  [pure virtual]
```

Sample external key events.

Parameters

in,out	*key*	Output parameter that will be set to the key value if a keypress was detected.

Returns

True if a keypress was detected and the "key" parameter is set to a value.

7.34 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.

7.34.1 Member Function Documentation

7.34.1.1 init()

`static void init ( ) [static]`

Perform configuration of IO pins.

7.34.1.2 sample()

`static unsigned int sample ( ) [static]`

Sample button states.

Returns

the sampled state of the buttons.

7.35 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

7.35.1 Detailed Description

A Button specialization that also displays an icon on top of the button bitmap.

See also

Button

7.35.2 Member Function Documentation

7.35.2.1 draw()

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

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Reimplemented from Button.

7.35.2.2 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.

7.35.2.3 getIconX()

```
int16_t getIconX ( ) const  [inline]
```

Gets the x coordinate of the icon bitmap.

Returns

The x coordinate of the icon bitmap.

7.35.2.4 getIconY()

```
int16_t getIconY ( ) const  [inline]
```

Gets the y coordinate of the icon bitmap.

Returns

The y coordinate of the icon bitmap.

7.35.2.5 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_BUTTONWITHICON.

Reimplemented from Button.

7.35.2.6 setBitmaps()

```
void setBitmaps (
            const Bitmap & newBackgroundReleased,
            const Bitmap & newBackgroundPressed,
            const Bitmap & newIconReleased,
            const Bitmap & newIconPressed )  [virtual]
```

Sets the bitmaps used by this button.

Parameters

newBackgroundReleased	Bitmap to use when button is released.
newBackgroundPressed	Bitmap to use when button is pressed.
newIconReleased	The bitmap for the icon in the released/unpressed button state.
newIconPressed	The bitmap for the icon in the pressed button state.

7.35.2.7 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

x	The new x value, relative to the background bitmap. A negative value is allowed.

7.35.2.8 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

x	The new x value, relative to the background bitmap. A negative value is allowed.
y	The new y value, relative to the background bitmap. A negative value is allowed.

7.35.2.9 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.

7.36 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

7.36.1 Detailed Description

A Button specialization that also displays a text on top of the button bitmap.

See also

Button

7.36.2 Constructor & Destructor Documentation

7.36.2.1 ButtonWithLabel()

```
ButtonWithLabel ( )
```

Default constructor.

7.36.3 Member Function Documentation

7.36.3.1 draw()

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

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Reimplemented from Button.

7.36.3.2 getLabelRotation()

```
TextRotation getLabelRotation ( )  [inline]
```

Gets the current rotation of the text on the label.

Returns

The current rotation of the text.

7.36.3.3 getLabelText()

```
TypedText getLabelText ( ) const  [inline]
```

Gets the text used for the label.

Returns

The text used for the label.

7.36.3.4 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.

7.36.3.5 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_BUTTONWITHLABEL.

Reimplemented from Button.

7.36.3.6 setLabelColor()

```
void setLabelColor (
            colortype col,
            bool performInvalidate = false )  [inline]
```

Sets label color.

Parameters

col	The color with which the text label should be drawn.
performInvalidate	Optional parameter. If true, performs an instant invalidation of the button.

7.36.3.7 setLabelColorPressed()

```
void setLabelColorPressed (
            colortype col,
            bool performInvalidate = false )  [inline]
```

Sets label color when the button is pressed.

Parameters

col	The color with which the text label should be drawn when the button is pressed.
performInvalidate	Optional parameter. If true, performs an instant invalidation of the button.

7.36.3.8 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

rotation	the rotation of the text. Default is TEXT_ROTATE_0.

7.36.3.9 setLabelText()

```
void setLabelText (
            TypedText t )  [inline]
```

Sets the text to display on the button. Texts with wildcards are not supported.

Parameters

t	The text to display.

7.36.3.10 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.

7.37 CacheableContainer Class Reference

A CacheableContainer is a Container that can have its drawing redirected into a Bitmap.

`#include <touchgfx/containers/CacheableContainer.hpp>`

Classes

- class CachedImage

  A CachedImage is a specialized Image object that exposes the setupDrawChain() method.

Public Member Functions

- CacheableContainer ()

  Default constructor.
- virtual ~CacheableContainer ()

  Destructor.
- void setCacheBitmap (BitmapId bitmapId)

  Set the dynamic bitmap into which the container content will be rendered.
- void updateCache ()

  Render the container into the attached dynamic bitmap.
- void updateCache (const Rect &rect)

  Render the container into the attached dynamic bitmap.
- void enableCachedMode (bool enable)

  Toggle cached mode on and off.
- virtual void invalidateRect (Rect &invalidatedArea) const

  Request that a subregion of this drawable is redrawn.
- bool isChildInvalidated () const

  Queries the CacheableContainer whether any child widget has been invalidated.

Protected Member Functions

- virtual void setupDrawChain (const Rect &invalidatedArea, Drawable **nextPreviousElement)

  For TouchGFX internal use only.

Additional Inherited Members

7.37.1 Detailed Description

A CacheableContainer 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. A CacheableContainer can also render its content to a dynamic bitmap that could be used as a texture in subsequent drawing operations.

See also

Container
Bitmap

7.37.2 Constructor & Destructor Documentation

7.37.2.1 CacheableContainer()

```
CacheableContainer ( )  [inline]
```

Default constructor.

7.37.2.2 ~CacheableContainer()

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

Destructor.

7.37.3 Member Function Documentation

7.37.3.1 enableCachedMode()

```
void enableCachedMode (
            bool enable )
```

Toggle cached mode on and off. The cacheable container behaves as a regular container when cached mode is off.

Parameters

enable	Enable or disable cached mode.

7.37.3.2 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	*invalidatedArea*	The area of this drawable to redraw expressed in coordinates relative to its parent (e.g. to request a complete redraw, invalidatedArea will be (0, 0, width, height).

Reimplemented from Drawable.

7.37.3.3 isChildInvalidated()

```
void isChildInvalidated ( ) const
```

Queries the CacheableContainer whether any child widget has been invalidated.

Returns

True if a child widget has been invalidated and false otherwise.

7.37.3.4 setCacheBitmap()

```
void setCacheBitmap (
            BitmapId bitmapId )
```

Set the dynamic bitmap into which the container content will be rendered.

Parameters

bitmapId	Id of the dynamic bitmap to serve as a render target.

7.37.3.5 setupDrawChain()

```
virtual void setupDrawChain (
            const Rect & invalidatedArea,
            Drawable ** nextPreviousElement )  [protected], [virtual]
```

Configure linked list for draw chain.

Note

For TouchGFX internal use only.

Parameters

	invalidatedArea	Include drawables that intersect with this area only.
in,out	*nextPreviousElement*	Modifiable element in linked list.

Reimplemented from Container.

7.37.3.6 updateCache() [1/2]

```
void updateCache ( )
```

Render the container into the attached dynamic bitmap.

7.37.3.7 updateCache() [2/2]

```
void updateCache (
            const Rect & rect )
```

Render the container into the attached dynamic bitmap. Only the specified Rect region is updated.

Parameters

rect	Region to update.

7.38 CacheableContainer::CachedImage Class Reference

A CachedImage is a specialized Image object that exposes the setupDrawChain() method.

`#include <touchgfx/containers/CacheableContainer.hpp>`

Public Member Functions

- CachedImage ()
  Default constructor.
- virtual ~CachedImage ()
  Destructor.
- void setupDrawChain (const Rect &invalidatedArea, Drawable **nextPreviousElement)
  For TouchGFX internal use only.

Additional Inherited Members

7.38.1 Detailed Description

A CachedImage is a specialized Image object that exposes the setupDrawChain() method.

See also

CacheableContainer
Image

7.38.2 Constructor & Destructor Documentation

7.38.2.1 CachedImage()

```
CachedImage ( )  [inline]
```

Default constructor.

7.38.2.2 ~CachedImage()

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

Destructor.

7.38.3 Member Function Documentation

7.38.3.1 setupDrawChain()

```
void setupDrawChain (
            const Rect & invalidatedArea,
            Drawable ** nextPreviousElement )  [inline], [virtual]
```

Configure linked list for draw chain.

Note

For TouchGFX internal use only.

Parameters

	invalidatedArea	Include drawables that intersect with this area only.
in,out	*nextPreviousElement*	Modifiable element in linked list.

Reimplemented from Drawable.

7.39 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.

7.39.1 Detailed Description

Cache bookkeeping.

7.40 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.

7.40.1 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

dest_type	The type of the class in which the member function resides.
T1	The type of the first argument in the member function, or void if none.
T2	The type of the second argument in the member function, or void if none.
T3	The type of the third argument in the member function, or void if none.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 Callback() [1/2]

```
Callback ( )  [inline]
```

Initializes an empty callback.

7.40.2.2 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	*pobject*	Pointer to the object on which the function should be called.
in	*pmemfun_3*	Address of member function. This is the version where function takes three arguments.

7.40.3 Member Function Documentation

7.40.3.1 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

t1	This value will be passed as the first argument in the function call.
t2	This value will be passed as the second argument in the function call.
t3	This value will be passed as the third argument in the function call.

Implements GenericCallback< T1, T2, T3 >.

7.40.3.2 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 >.

7.41 Callback< dest_type, T1, T2, 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_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.

7.41.1 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

dest_type	The type of the class in which the member function resides.
T1	The type of the first argument in the member function, or void if none.
T2	The type of the second argument in the member function, or void if none.

7.41.2 Constructor & Destructor Documentation

7.41.2.1 Callback() [1/2]

```
Callback ( )  [inline]
```

Initializes an empty callback.

7.41.2.2 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	*pobject*	Pointer to the object on which the function should be called.
in	*pmemfun_2*	Address of member function. This is the version where function takes two arguments.

7.41.3 Member Function Documentation

7.41.3.1 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

t1	This value will be passed as the first argument in the function call.
t2	This value will be passed as the second argument in the function call.

7.41.3.2 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 >.

7.42 Callback< dest_type, T1, 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_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.

7.42.1 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

dest_type	The type of the class in which the member function resides.
T1	The type of the first argument in the member function, or void if none.

7.42.2 Constructor & Destructor Documentation

7.42.2.1 Callback() [1/2]

```
Callback ( )  [inline]
```

Initializes an empty callback.

7.42.2.2 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	*pobject*	Pointer to the object on which the function should be called.
in	*pmemfun_1*	Address of member function. This is the version where function takes one argument.

7.42.3 Member Function Documentation

7.42.3.1 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

t1	This value will be passed as the first argument in the function call.

See also

isValid()

7.42.3.2 isValid()

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

Query if this object is valid.

Returns

true if valid, false if not.

Implements GenericCallback< T1 >.

7.43 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.

7.43.1 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

dest_type	The type of the class in which the member function resides.

7.43.2 Constructor & Destructor Documentation

7.43.2.1 Callback() [1/2]

```
Callback ( )  [inline]
```

Initializes an empty callback.

7.43.2.2 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	*pobject*	Pointer to the object on which the function should be called.
in	*pmemfun_0*	Address of member function. This is the version where function takes zero arguments.

7.43.3 Member Function Documentation

7.43.3.1 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).

7.43.3.2 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<>.

7.44 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".

7.45 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.

7.45.1 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.

7.45.2 Constructor & Destructor Documentation

7.45.2.1 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

_widget	a pointer to the CanvasWidget using this Canvas. Used for getting the canvas dimensions.
invalidatedArea	the are which should be updated.

7.45.2.2 ~Canvas()

```
~Canvas ( )  [virtual]
```

Destructor. Takes care of unlocking the frame buffer.

7.45.3 Member Function Documentation

7.45.3.1 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

x	The x coordinate for the pen position in Q5 format.
y	The y coordinate for the pen position in Q5 format.

See also

CWRUtil::Q5
moveTo

7.45.3.2 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

T	either int or float.

Parameters

x	The x coordinate for the pen position.
y	The y coordinate for the pen position.

7.45.3.3 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

x	The x coordinate for the pen position in Q5 format.
y	The y coordinate for the pen position in Q5 format.

See also

CWRUtil::Q5
lineTo

7.45.3.4 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

T	Either int or float.

Parameters

x	The x coordinate for the pen position.
y	The y coordinate for the pen position.

7.45.3.5 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.

7.46 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

7.46.1 Detailed Description

Class for drawing complex polygons on the LCD using CanvasWidgetRenderer.

See also

Widget

7.46.2 Constructor & Destructor Documentation

7.46.2.1 CanvasWidget()

```
CanvasWidget ( )
```

Constructor.

7.46.2.2 ~CanvasWidget()

```
~CanvasWidget ( )  [virtual]
```

Destructor. Declared virtual for sub classing purposes.

7.46.3 Member Function Documentation

7.46.3.1 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

invalidatedArea	The invalidated area.

See also

drawCanvasWidget()

Implements Drawable.

7.46.3.2 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

invalidatedArea	The invalidated area.

Returns

true the widget was drawn, false if not.

See also

draw()

Implemented in Circle, Line, and AbstractShape.

7.46.3.3 getAlpha()

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

Gets the current alpha value.

Returns

The current alpha value.

7.46.3.4 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 Circle, AbstractShape, and Line.

7.46.3.5 getPainter()

```
AbstractPainter & getPainter ( ) const  [virtual]
```

Gets the current painter for the CanvasWidget.

Returns

The painter.

See also

AbstractPainter

7.46.3.6 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.

7.46.3.7 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.

7.46.3.8 setAlpha()

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

Sets the alpha channel for the CanvasWidget.

Parameters

alpha	The alpha value. 255 = completely solid.

7.46.3.9 setPainter()

```
void setPainter (
            AbstractPainter & painter )  [virtual]
```

Sets the painter for the CanvasWidget.

Note

The area containing the CanvasWidget is not invalidated.

Parameters

in	*painter*	The painter for the CanvasWidget.

See also

AbstractPainter

7.47 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

7.47.1 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

7.47.2 Member Function Documentation

7.47.2.1 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.

7.47.2.2 getOutlineBuffer()

```
static Cell * getOutlineBuffer ( )  [static]
```

Gets pointer to memory used for Cell objects in Outline.

Returns

Pointer to memory used internally by Outline.

7.47.2.3 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.

7.47.2.4 getScanlineCounts()

```
static void * getScanlineCounts ( )  [static]
```

Gets pointer to memory used for counts in Scanline.

Returns

Pointer to memory used internally by Scanline.

7.47.2.5 getScanlineCovers()

```
static void * getScanlineCovers ( )  [static]
```

Gets pointer to memory used for covers in Scanline.

Returns

Pointer to memory used internally by Scanline.

7.47.2.6 getScanlineStartIndices()

```
static void * getScanlineStartIndices ( )  [static]
```

Gets pointer to memory used for indices in Scanline.

Returns

Pointer to memory used internally by Scanline.

7.47.2.7 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).

7.47.2.8 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.

7.47.2.9 getWriteMemoryUsageReport()

```
static bool getWriteMemoryUsageReport ( )  [static]
```

Gets write memory usage report flag.

Returns

true if it CWR writes memory reports, false if not.

7.47.2.10 hasBuffer()

```
static bool hasBuffer ( )  [static]
```

Query if CanvasWidgetRenderer has been initialized with a buffer.

Returns

True if a buffer has been setup.

7.47.2.11 numCellsMissing()

```
static void numCellsMissing (
            unsigned missing )  [static]
```

Called by the destructor in Outline to help keep track of the memory requirements of CanvasWidgets.

Parameters

missing	Number of Cell objects required, but not available, to Outline.

7.47.2.12 numCellsUsed()

```
static void numCellsUsed (
            unsigned used )  [static]
```

Called by the destructor in Outline to help keep track of the memory requirements of CanvasWidgets.

Parameters

used	Number of Cell objects used from the dedicated buffer.

7.47.2.13 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

width	The width of the scanline on screen.

Returns

true if it succeeds, false if it fails.

7.47.2.14 setupBuffer()

```
static void setupBuffer (
            uint8_t * buffer,
            unsigned bufsize )  [static]
```

Setup the buffers used by CanvasWidget.

Parameters

in	*buffer*	Buffer reserved for CanvasWidget.
	bufsize	The size of the buffer.

7.47.2.15 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 bytesd requied 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

writeUsageReport	true to write report.

See also

setupBuffer

7.48 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).*

7.48.1 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.

7.48.2 Member Function Documentation

7.48.2.1 addCover()

```
void addCover (
            int _cover,
            int _area )  [inline]
```

Adds a cover to a Cell.

Parameters

_cover	The cover to add to the Cell.
_area	The area to add to the Cell.

7.48.2.2 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.

7.48.2.3 set()

```
void set (
            int _x,
            int _y,
            int _cover,
            int _area )  [inline]
```

Sets all the Cell parameters.

Parameters

_x	The x coordinate.
_y	The y coordinate.
_cover	The cover.
_area	The area.

7.48.2.4 setCoord()

```
void setCoord (
            int _x,
            int _y )  [inline]
```

Sets the coordinate of the Cell.

Parameters

↩ _↩ *x*	The Cell's x coordinate.
↩ _↩ *y*	The Cell's y coordinate.

7.48.2.5 setCover()

```
void setCover (
            int _cover,
            int _area )  [inline]
```

Sets the cover of area cell.

Parameters

_cover	The cover.
_area	The area.

7.49 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 updateArc (const T startAngle, const T endAngle)

  Updates the start and 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

7.49.1 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

7.49.2 Constructor & Destructor Documentation

7.49.2.1 Circle()

```
Circle ( )
```

Constructs a new Circle.

7.49.3 Member Function Documentation

7.49.3.1 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

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Returns

true if it succeeds, false if it fails.

Implements CanvasWidget.

7.49.3.2 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

T	Generic type parameter, either int or float.

Parameters

out	*startAngle*	The start.
out	*endAngle*	The end.

See also

setArc

7.49.3.3 getArcEnd() [1/2]

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

Gets the end angle in degrees for the arc.

Template Parameters

T	Generic type parameter, either int (default) or float.

Returns

The finishing angle for the arc.

See also

getArc
setArc

7.49.3.4 getArcEnd() [2/2]

```
template< typename T > void getArcEnd (
            T & angle ) const  [inline]
```

Gets the end angle in degrees for the arc.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

in,out	*angle*	The angle.

7.49.3.5 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

7.49.3.6 getArcStart() [2/2]

```
template< typename T > void getArcStart (
            T & angle ) const  [inline]
```

Gets the start angle in degrees for the arc.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

in,out	*angle*	The angle.

See also

getArc
setArc

7.49.3.7 getCapPrecision()

```
int getCapPrecision ( ) const
```

Gets the precision of the ends of the Circle arc.

Returns

The cap precision in degrees.

See also

getCapPrecision

7.49.3.8 getCenter()

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

Gets the center coordinates of the Circle.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*x*	The x coordinate of the center.
out	*y*	The y coordinate of the center.

7.49.3.9 getLineWidth()

```
template< typename T > void getLineWidth (
            T & width ) const  [inline]
```

Gets line width.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*width*	The width.

See also

setLineWidth

7.49.3.10 getMinimalRect() [1/3]

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

Gets minimal rectangle for the current shape of the circle.

Returns

The minimal rectangle.

Reimplemented from CanvasWidget.

7.49.3.11 getMinimalRect() [2/3]

```
Rect getMinimalRect (
            int16_t arcStart,
            int16_t arcEnd ) const
```

Gets minimal rectangle containing a given circle arc.

Parameters

arcStart	The arc start.
arcEnd	The arc end.

Returns

The minimal rectangle.

7.49.3.12 getMinimalRect() [3/3]

```
Rect getMinimalRect (
            CWRUtil::Q5 arcStart,
            CWRUtil::Q5 arcEnd ) const
```

Gets minimal rectangle containing a given circle arc.

Parameters

arcStart	The arc start.
arcEnd	The arc end.

Returns

The minimal rectangle.

7.49.3.13 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

7.49.3.14 getRadius()

```
template< typename T > void getRadius (
            T & r ) const  [inline]
```

Gets the radius of the Circle.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*r*	The radius.

7.49.3.15 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

T	Generic type parameter, either int or float.

Parameters

startAngle	The start degrees.
endAngle	The end degrees.

Note

The area containing the Circle is not invalidated.

See also

getArc
updateArcStart
updateArcEnd
updateArc

7.49.3.16 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

startAngle	The start degrees.
endAngle	The end degrees.

Note

The area containing the Circle is not invalidated.

See also

getArc
updateArcStart
updateArcEnd
updateArc

7.49.3.17 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

precision	The new cap precision.

Note

The circle is not invalidated.

7.49.3.18 setCenter() [1/2]

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

Sets the center of the Circle.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of center.
y	The y coordinate of center.

Note

The area containing the Circle is not invalidated.

7.49.3.19 setCenter() [2/2]

```
void setCenter (
            const int16_t x,
            const int16_t y )  [inline]
```

Sets the center of the Circle.

Parameters

x	The x coordinate of center.
y	The y coordinate of center.

Note

The area containing the Circle is not invalidated.

7.49.3.20 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

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of center.
y	The y coordinate of center.
r	The radius.

Note

The area containing the Circle is not invalidated.

7.49.3.21 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

x	The x coordinate of center.
y	The y coordinate of center.
r	The radius.

Note

The area containing the Circle is not invalidated.

7.49.3.22 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

T	Generic type parameter, either int or float.

Parameters

width	The width of the line measured in pixels.

Note

The area containing the Circle is not invalidated.
if the new line with is smaller than the old width, the circle should be invalidated before updating the width to ensure that the old circle is completely erased.

7.49.3.23 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

precision	The precision measured in degrees.

Note

The circle is not invalidated.

7.49.3.24 setRadius()

```
template< typename T > void setRadius (
            const T r )  [inline]
```

Sets the radius of the Circle.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

r	The radius.

Note

The area containing the Circle is not invalidated.

7.49.3.25 updateArc()

```
template< typename T > void updateArc (
            const T startAngle,
            const T endAngle )  [inline]
```

Updates the start and end angle in degrees for this Circle arc.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

startAngle	The new start angle in degrees.
endAngle	The new end angle in degrees.

Note

The areas containing the updated Circle arcs are invalidated.

See also

setArc
getArc
updateArcStart
updateArcEnd

7.49.3.26 updateArcEnd()

```
template< typename T > void updateArcEnd (
            const T endAngle )  [inline]
```

Updates the end angle in degrees for this Circle arc.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

endAngle	The end angle in degrees.

Note

The area containing the updated Circle arc is invalidated.

See also

setArc
updateArcStart
updateArc

7.49.3.27 updateArcStart()

```
template< typename T > void updateArcStart (
            const T startAngle )  [inline]
```

Updates the start angle in degrees for this Circle arc.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

startAngle	The start angle in degrees.

Note

The area containing the updated Circle arc is invalidated.

See also

setArc
updateArcEnd
updateArc

7.50 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

7.50.1 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

7.50.2 Constructor & Destructor Documentation

7.50.2.1 CircleProgress()

```
CircleProgress ( )
```

Default constructor.

7.50.2.2 ~CircleProgress()

```
~CircleProgress ( )  [virtual]
```

Destructor.

7.50.3 Member Function Documentation

7.50.3.1 getAlpha()

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

Gets the alpha of the circle.

Returns

The alpha.

See also

setAlpha

7.50.3.2 getCapPrecision()

```
int getCapPrecision ( ) const  [inline], [virtual]
```

Gets the cap precision.

Returns

The cap precision.

7.50.3.3 getCenter()

```
void getCenter (
            int & x,
            int & y ) const  [virtual]
```

Gets the center coordinates.

Parameters

out	*x*	The x coordinate.
out	*y*	The y coordinate.

7.50.3.4 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

7.50.3.5 getLineWidth()

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

Gets line width.

Returns

The line width.

See also

setLineWidth

7.50.3.6 getRadius()

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

Gets the radius.

Returns

The radius.

7.50.3.7 getStartAngle()

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

Gets start angle.

Returns

The start angle.

See also

setStartEndAngle
getEndAngle

7.50.3.8 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

alpha	The alpha.

See also

getAlpha

7.50.3.9 setCapPrecision()

```
void setCapPrecision (
            int precision )  [virtual]
```

Sets the cap precision of the circle arc. This is not used if line width is zero.

Parameters

precision	The cap precision.

See also

Circle::setCapPrecision

7.50.3.10 setCenter()

```
void setCenter (
            int x,
            int y )  [virtual]
```

Sets the center of the circle / arc.

Parameters

x	The int to process.
y	The int to process.

7.50.3.11 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

width	The width.

See also

Circle::setLineWidth

7.50.3.12 setPainter()

```
void setPainter (
            AbstractPainter & painter )  [virtual]
```

Sets the painter to use for drawing.

Parameters

in,out	*painter*	The painter.

See also

Circle::setPainter
AbstractPainter

7.50.3.13 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

x	The x coordinate.
y	The y coordinate.
width	The width of the circle progress indicator.
height	The height of the circle progress indicator.

Reimplemented from AbstractProgressIndicator.

7.50.3.14 setRadius()

```
void setRadius (
            int r )  [virtual]
```

Sets the radius of the circle.

Parameters

r	The int to process.

See also

Circle::setRadius

7.50.3.15 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

startAngle	The start angle.
endAngle	The end angle.

7.50.3.16 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

value	The value.

Reimplemented from AbstractProgressIndicator.

7.51 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

7.51.1 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.

7.51.2 Member Function Documentation

7.51.2.1 handleClickEvent()

```
void handleClickEvent (
            const ClickEvent & event ) [inline], [virtual]
```

Handles the click event described by event. The action callback is called when receiving a REALEASED event in PRESSED state.

Parameters

event	The event.

Reimplemented from Drawable.

7.52 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.

7.52.1 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

7.52.2 Member Enumeration Documentation

7.52.2.1 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.

7.52.3 Constructor & Destructor Documentation

7.52.3.1 ClickEvent()

```
ClickEvent (
            ClickEventType type,
            int16_t x,
            int16_t y,
            int16_t force = 0 )  [inline]
```

Constructor.

Parameters

type	The type of the click event.
x	The x coordinate of the click event.
y	The y coordinate of the click event.
force	The force of the click. On touch displays this usually means how hard the user pressed on the display. On the windows platform, this will always be zero.

7.52.3.2 ~ClickEvent()

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

Destructor.

7.52.4 Member Function Documentation

7.52.4.1 getEventType()

```
Event::EventType getEventType ( )  [inline], [virtual]
```

Gets event type.

Returns

The type of this event.

Implements Event.

7.52.4.2 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.

7.52.4.3 getType()

```
ClickEventType getType ( ) const  [inline]
```

Gets the click type of this event.

Returns

The click type of this event.

7.52.4.4 getX()

```
int16_t getX ( ) const  [inline]
```

Gets the x coordinate of this event.

Returns

The x coordinate of this event.

7.52.4.5 getY()

```
int16_t getY ( ) const  [inline]
```

Gets the y coordinate of this event.

Returns

The y coordinate of this event.

7.52.4.6 setType()

```
void setType (
            ClickEventType type )  [inline]
```

Sets the click type of this event.

Parameters

type	The type to set.

7.52.4.7 setX()

```
void setX (
            int16_t x )  [inline]
```

Sets the x coordinate of this event.

Parameters

x	The x coordinate of this event.

7.52.4.8 setY()

```
void setY (
            int16_t y )  [inline]
```

Sets the y coordinate of this event.

Parameters

y	The y coordinate of this event.

7.53 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.

7.53.1 Detailed Description

template<class T>
class touchgfx::ClickListener< T >

Mix-in class that extends a class with a click action event that is called when the class receives a click event.

Template Parameters

T	specifies the type to extend with the ClickListener behavior.

7.53.2 Constructor & Destructor Documentation

7.53.2.1 ClickListener()

```
ClickListener ( )  [inline]
```

Default constructor.

7.53.3 Member Function Documentation

7.53.3.1 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

event	Information about the click.

See also

Drawable::handleClickEvent()

7.53.3.2 setClickAction()

```
void setClickAction (
            GenericCallback< const T &, const ClickEvent & > & callback )  [inline]
```

Associates an action to be performed when the class T is clicked.

Parameters

callback	The callback to be executed. The callback will be given a reference to T.

See also

GenericCallback

7.54 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.

7.54.1 Detailed Description

Contains functionality for color conversion.

7.54.2 Member Function Documentation

7.54.2.1 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

color	The 16 bit color value.

Returns

The blue part of the color.

7.54.2.2 getColorFrom24BitRGB()

```
static colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values. Depending on your chosen color bit depth, the color will be interpreted internally as either a 16 bit or 24 bit color value. This function can be safely used regardless of whether your application is configured for 16 or 24 bit colors.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

7.54.2.3 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

color	The 16 bit color value.

Returns

The green part of the color.

7.54.2.4 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

color	The color value.

Returns

The red part of the color.

7.55 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.

7.55.1 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.

7.55.2 Constructor & Destructor Documentation

7.55.2.1 colortype() [1/2]

```
colortype ( )  [inline]
```

Default constructor. Creates a black (0) color.

7.55.2.2 colortype() [2/2]

```
colortype (
            uint32_t col )  [inline]
```

Contructor which creates a colortype with the given color. Use Color::getColorFrom24BitRGB() to create a color that will work on both 16 bit LCD and 24 bit LCD.

Parameters

col	The color.

See also

Color::getColorFrom24BitRGB()

7.55.3 Member Function Documentation

7.55.3.1 getColor32()

```
uint32_t getColor32 ( ) const  [inline]
```

Returns

The color 32.

See also

Color::getRedColor()
Color::getGreenColor()
Color::getBlueColor()

7.55.3.2 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.

7.56 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 dataFormatA4, 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 date associated with this glyph.
- virtual int8_t getKerning (Unicode::UnicodeChar prevChar, const GlyphNode *glyph) const =0

  Gets the kerning distance between two characters.
- 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

7.56.1 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

7.56.2 Constructor & Destructor Documentation

7.56.2.1 ConstFont()

```
ConstFont (
            const GlyphNode * list,
            uint16_t size,
            uint16_t height,
            uint8_t pixBelowBase,
            uint8_t bitsPerPixel,
            uint8_t dataFormatA4,
            uint8_t maxLeft,
            uint8_t maxRight,
            const Unicode::UnicodeChar fallbackChar,
            const Unicode::UnicodeChar ellipsisChar )
```

Creates a font instance.

Parameters

list	The array of glyphs known to this font.
size	The number of glyphs in list.
height	The height in pixels of the highest character in this font.
pixBelowBase	The maximum number of pixels that can be drawn below the baseline in this font.
bitsPerPixel	The number of bits per pixel in this font.
dataFormatA4	The glyphs are saved using ST A4 format.
maxLeft	The maximum a character extends to the left.
maxRight	The maximum a character extends to the right.
fallbackChar	The fallback character for the typography in case no glyph is available.
ellipsisChar	The ellipsis character used for truncating long texts.

7.56.3 Member Function Documentation

7.56.3.1 find()

```
const GlyphNode * find (
            Unicode::UnicodeChar unicode ) const
```

Finds the glyph data associated with the specified unicode.

Parameters

unicode	The character to look up.

Returns

A pointer to the glyph node or null if the glyph was not found.

7.56.3.2 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

	unicode	The character to look up.
	pixelData	Pointer to the pixel data for the glyph if the glyph is found. This is set by this method.
out	*bitsPerPixel*	Reference where to place the number of bits per pixel.

Returns

A pointer to the glyph node or null if the glyph was not found.

Implements Font.

7.56.3.3 getKerning()

```
int8_t getKerning (
            Unicode::UnicodeChar prevChar,
            const GlyphNode * glyph ) const  [pure virtual]
```

Gets the kerning distance between two characters.

Parameters

prevChar	The unicode value of the previous character.
glyph	the glyph object for the current character.

Returns

The kerning distance between prevChar and glyph char.

Reimplemented from Font.

Implemented in InternalFlashFont.

7.56.3.4 getPixelData()

```
const uint8_t * getPixelData (
            const GlyphNode * glyph ) const  [pure virtual]
```

Gets the pixel date associated with this glyph.

Parameters

glyph	The glyph to get the pixels data from.

Returns

Pointer to the pixel data of this glyph.

Implemented in InternalFlashFont.

7.57 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

7.57.1 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

7.57.2 Constructor & Destructor Documentation

7.57.2.1 Container()

```
Container ( )  [inline]
```

Default constructor.

7.57.2.2 ~Container()

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

Destructor.

7.57.3 Member Function Documentation

7.57.3.1 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	*d*	The Drawable to add.

Reimplemented in SlideMenu, ScrollableContainer, ModalWindow, ListLayout, and SwipeContainer.

7.57.3.2 contains()

```
bool contains (
             const Drawable & d )  [virtual]
```

Query if this object contains the given drawable.

Parameters

d	The Drawable to look for.

Returns

True if the specified Drawable instance is direct child of this container, false otherwise.

7.57.3.3 draw()

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

Draws the children of this container. Only used when JSMOC is disabled.

Parameters

invalidatedArea	The area to draw.

Implements Drawable.

Reimplemented in Keyboard.

7.57.3.4 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

in	*function*	The function to be executed for each child.

See also

ListLayout::insert for a usage example.

7.57.3.5 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.

7.57.3.6 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.

7.57.3.7 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

	x	The x coordinate of the intersection.
	y	The y coordinate of the intersection.
out	*last*	out parameter in which the result is placed.

See also

Drawable::getLastChild

Implements Drawable.

Reimplemented in ScrollableContainer.

7.57.3.8 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.

7.57.3.9 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.

7.57.3.10 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	*previous*	The Drawable to insert after. If null, insert as header.
in	*d*	The Drawable to insert.

Reimplemented in ListLayout.

7.57.3.11 moveChildrenRelative()

```
void moveChildrenRelative (
            int16_t deltaX,
            int16_t deltaY )  [protected], [virtual]
```

Calls moveRelative on all children.

Parameters

deltaX	Horizontal displacement.
deltaY	Vertical displacement.

Reimplemented in ScrollableContainer.

7.57.3.12 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	*d*	The Drawable to remove.

Reimplemented in SlideMenu, ModalWindow, ListLayout, and SwipeContainer.

7.57.3.13 removeAll()

```
void removeAll ( )  [virtual]
```

Removes all children by resetting their parent and sibling pointers.

Reimplemented in ListLayout.

7.57.3.14 setupDrawChain()

```
virtual void setupDrawChain (
            const Rect & invalidatedArea,
            Drawable ** nextPreviousElement )  [protected], [virtual]
```

Configure linked list for draw chain.

Note

For TouchGFX internal use only.

Parameters

	invalidatedArea	Include drawables that intersect with this area only.
in,out	*nextPreviousElement*	Modifiable element in linked list.

Reimplemented from Drawable.

Reimplemented in Keyboard, and CacheableContainer.

7.57.3.15 unlink()

```
void unlink ( )  [virtual]
```

Removes all children by unlinking the first child. The parent and sibling pointers of the children are not reset.

7.58 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

7.58.1 Detailed Description

template<Direction templateDirection>
class touchgfx::CoverTransition< templateDirection >

A Transition that slides the new screen over the previous.

Template Parameters

templateDirection	Type of the template direction.

See also

Transition

7.58.2 Constructor & Destructor Documentation

7.58.2.1 CoverTransition()

```
CoverTransition (
            const uint8_t transitionSteps = 20 )  [inline]
```

Constructor.

Parameters

transitionSteps	Number of steps in the transition animation.

7.58.2.2 ~CoverTransition()

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

Destructor.

7.58.3 Member Function Documentation

7.58.3.1 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.

7.58.3.2 init()

```
void init ( )  [inline], [virtual]
```

Initializes this object.

See also

Transition::init()

Reimplemented from Transition.

7.58.3.3 initMoveDrawable()

```
void initMoveDrawable (
            Drawable & d )  [inline], [protected], [virtual]
```

Moves the Drawable to its initial position outside of the visible area.

Parameters

in	*d*	The Drawable to move.

7.58.3.4 tearDown()

```
void tearDown ( )  [inline], [virtual]
```

Tear down.

See also

Transition::tearDown()

Reimplemented from Transition.

7.58.3.5 tickMoveDrawable()

```
void tickMoveDrawable (
            Drawable & d )  [inline], [protected], [virtual]
```

Moves the Drawable.

Parameters

in	*d*	The Drawable to move.

7.59 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

- static FORCE_INLINE_FUNCTION Q5 toQ5 (Q5 value)

  Convert a Q5 to itself.
- 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 muldivQ10 (Q10 factor1, Q10 factor2, Q10 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.
- static Q5 mulQ5 (Q5 factor1, Q10 factor2)

  Multiply one Q5 by a Q10 returning a new Q5.

7.59.1 Detailed Description

Helper classes and functions for CanvasWidget. A handful of utility functions can be found here. These include helper functions for converting between float, int and Q5/Q10/Q15 format. There are also functions for calculating sin() and cos() in integers with a high number of bits reserved for fraction. Having sin() and cos() pre- calculated in this way allows very fast drawing of circles without the need for floating point arithmetic.

Using Q5, numbers from -1024.00000 to +1024.96875 with a precision of 1/32 = 0.03125 can be represented.

See also

http://en.wikipedia.org/wiki/Q_%28number_format%29
Widget

7.59.2 Member Function Documentation

7.59.2.1 angle() [1/4]

```
template< typename T > static int angle (
            T x,
            T y )  [inline], [static]
```

Template Parameters

T	Generic type parameter (int or float).

Parameters

x	The x coordinate.
y	The y coordinate.

Returns

The angle of the coordinate.

7.59.2.2 angle() [2/4]

```
template< typename T > static int angle (
            T x,
            T y,
            T & d )  [inline], [static]
```

Template Parameters

T	Generic type parameter (int or float).

Parameters

	x	The x coordinate.
	y	The y coordinate.
out	*d*	The distance from (0,0) to (x,y).

Returns

The angle of the coordinate.

7.59.2.3 angle() [3/4]

```
static int angle (
            Q5 x,
            Q5 y )  [inline], [static]
```

Parameters

x	The x coordinate.
y	The y coordinate.

Returns

The angle of the coordinate.

7.59.2.4 angle() [4/4]

```
static int angle (
            Q5 x,
            Q5 y,
            Q5 & d )  [inline], [static]
```

Parameters

	x	The x coordinate.
	y	The y coordinate.
out	*d*	The distance from (0,0) to (x,y).

Returns

The angle.

7.59.2.5 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

q10	The 10.

Returns

An int8_t.

7.59.2.6 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	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()

7.59.2.7 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	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()

7.59.2.8 muldivQ10()

```
static Q5 muldivQ10 (
             Q10 factor1,
             Q10 factor2,
             Q10 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

factor1	The first factor.
factor2	The second factor.
divisor	The divisor.

Returns

factor1 ∗ factor2 / divisor.

7.59.2.9 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

factor1	The first factor.
factor2	The second factor.
divisor	The divisor.

Returns

factor1 ∗ factor2 / divisor.

7.59.2.10 mulQ5() [1/2]

```
static Q5 mulQ5 (
            Q5 factor1,
            Q5 factor2 )  [inline], [static]
```

Multiply two Q5's returning a new Q5 without overflowing.

Parameters

factor1	The first factor.
factor2	The second factor.

Returns

factor1 ∗ factor2.

7.59.2.11 mulQ5() [2/2]

```
static Q5 mulQ5 (
            Q5 factor1,
            Q10 factor2 )  [inline], [static]
```

Multiply one Q5 by a Q10 returning a new Q5 without overflowing.

Parameters

factor1	The first factor.
factor2	The second factor.

Returns

factor1 ∗ factor2.

7.59.2.12 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<<15) to get the rounded value of of sin(i). By using this function, a complete circle can be drawn without the need for using floating point math.

Parameters

i	the angle in degrees. The angle follows the angles of the clock, 0 being straight up and 90 being 3 o'clock.

Returns

the value of sin(i) with 15 bits precision on the fractional part.

7.59.2.13 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 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	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.

7.59.2.14 sqrtQ10()

```
static Q5 sqrtQ10 (
            Q10 value )  [inline], [static]
```

Parameters

value	The value to find the square root of.

Returns

The square root of the given value.

7.59.2.15 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

T	Should be either int or float.

Parameters

value	the integer to convert.

Returns

the converted integer.

7.59.2.16 toQ5() [1/2]

```
FORCE_INLINE_FUNCTION static Q5 toQ5 (
            Q5 value )  [inline], [static]
```

Convert a Q5 to itself. Allows toQ5 to be called with a variable that is already Q5.

Parameters

value	the Q5.

Returns

the value passed.

7.59.2.17 toQ5() [2/2]

```
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

T	Should be either int or float.

Parameters

value	the integer to convert.

Returns

the converted integer.

7.60 DebugPrinter Class Reference

The class DebugPrinter defines the interface for printing debug messages on top of the framebuffer.

`#include <touchgfx/lcd/LCD.hpp>`

Public Member Functions

- void setDebugString (const char *string)

  Sets the debug string to be displayed on top of the framebuffer.
- void setDebugPosition (uint16_t x, uint16_t y, uint16_t w, uint16_t h)

  Sets the position of the debug string.
- void setDebugScale (uint8_t scale)

  Sets the font scale of the debug string.
- void setDebugColor (colortype fg)

  Sets the foreground color of the debug string.
- virtual void draw (const LCD &lcd) const =0

  Draws the debug string on top of the framebuffer content.
- const Rect & region () const

  Returns the region of the debug string.

Protected Attributes

- const char * debugString

  Debug string to be displayed onscreen.
- Rect debugRegion

  Region onscreen where the debug message is displayed.
- colortype debugForegroundColor

  Font color to use when displaying the debug string.
- uint8_t debugScale

  Font scaling factor to use when displaying the debug string.

7.60.1 Detailed Description

The class DebugPrinter defines the interface for printing debug messages on top of the framebuffer.

7.60.2 Member Function Documentation

7.60.2.1 draw()

```
void draw (
            const LCD & lcd ) const  [pure virtual]
```

Draws the debug string on top of the framebuffer content.

Parameters

in	*lcd*	Reference on the LCD object to use for drawing the debug string.

Implemented in LCD24DebugPrinter.

7.60.2.2 region()

```
const Rect & region ( ) const  [inline]
```

Returns the region where the debug string is displayed.

Returns

Rect The debug string region.

7.60.2.3 setDebugColor()

```
void setDebugColor (
            colortype fg )  [inline]
```

Sets the foreground color of the debug string.

Parameters

in	*fg*	The foreground color of the debug string.

7.60.2.4 setDebugPosition()

```
void setDebugPosition (
            uint16_t x,
            uint16_t y,
            uint16_t w,
            uint16_t h )  [inline]
```

Sets the position onscreen where the debug string will be displayed.

Parameters

in	*x*	The coordinate of the region where the debug string is displayed.
in	*y*	The coordinate of the region where the debug string is displayed.
in	*w*	The width of the region where the debug string is displayed.
in	*h*	The height of the region where the debug string is displayed.

7.60.2.5 setDebugScale()

```
void setDebugScale (
            uint8_t scale ) [inline]
```

Sets the font scale of the debug string.

Parameters

in	*scale*	The font scale of the debug string.

7.60.2.6 setDebugString()

```
void setDebugString (
            const char * string ) [inline]
```

Sets the debug string to be displayed on top of the framebuffer.

Parameters

in	*string*	The string to be displayed.

7.61 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 (colortype 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.

7.61.1 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.

7.61.2 Member Enumeration Documentation

7.61.2.1 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

7.61.3 Constructor & Destructor Documentation

7.61.3.1 DigitalClock()

```
DigitalClock ( )
```

Default constructor.

7.61.3.2 ~DigitalClock()

```
~DigitalClock ( )  [virtual]
```

Destructor.

7.61.4 Member Function Documentation

7.61.4.1 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

displayLeadingZero	true = show leading zero. false = do not show leading zero.

7.61.4.2 getAlpha()

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

Gets the alpha value of the digital clock.

Returns

The alpha value. 255 = completely solid. 0 = invisible.

7.61.4.3 getDisplayMode()

```
DisplayMode getDisplayMode ( ) const  [inline], [virtual]
```

Returns

The display mode.

7.61.4.4 getTextWidth()

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

Returns

The text width of the current content of the digital clock.

7.61.4.5 setAlpha()

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

Sets the alpha value of the digital clock.

Parameters

alpha	The alpha value. 255 = completely solid. 0 = invisible.

7.61.4.6 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

baselineY	The y coordinate of the baseline.

7.61.4.7 setColor()

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

Sets the color of the text. Invalidates the DigitalClock.

Parameters

color	The new text color.

7.61.4.8 setDisplayMode()

```
void setDisplayMode (
            DisplayMode dm )  [inline], [virtual]
```

Parameters

dm	The new DisplayMode.

7.61.4.9 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

height	The height.

Reimplemented from Drawable.

7.61.4.10 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

typedText	Describes the typed text to use.

7.61.4.11 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

width	The width.

Reimplemented from Drawable.

7.61.4.12 updateClock()

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

Updates the visual representation of the clock.

Implements AbstractClock.

7.62 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.

7.62.1 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.

7.62.2 Member Function Documentation

7.62.2.1 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.

7.62.2.2 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.

7.62.2.3 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	*x*	the x part to translate.
in,out	*y*	the y part to translate.
	in	the rectangle defining the coordinate space.

7.62.2.4 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	*x*	the x part to translate.
in,out	*y*	the y part to translate.
	in	the rectangle defining the coordinate space.

7.62.2.5 transformDisplayToFrameBuffer() [5/6]

```
static void transformDisplayToFrameBuffer (
            Rect & r )  [static]
```

Transform rectangle from display to frame buffer coordinates.

Parameters

in,out	*r*	the rectangle to translate.

7.62.2.6 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	*r*	the rectangle to translate.
	in	the rectangle defining the coordinate space.

7.62.2.7 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.

7.63 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 DMAType getDMAType (void)

  Function for obtaining the DMA type of the concrete DMA_Interface implementation.
- 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.

7.63.1 Detailed Description

DMA_Interface provides basic functionality and structure for processing "blit" operations using DMA.

7.63.2 Constructor & Destructor Documentation

7.63.2.1 ~DMA_Interface()

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

Destructor.

7.63.2.2 DMA_Interface()

```
DMA_Interface (
            DMA_Queue & dmaQueue )  [inline], [protected]
```

Constructs a DMA Interface object.

Parameters

in	*dmaQueue*	Reference to the queue of DMA operations.

7.63.3 Member Function Documentation

7.63.3.1 addToQueue()

```
void addToQueue (
            const BlitOp & op )  [virtual]
```

Inserts a BlitOp for processing. This also potentially starts the DMA controller, if not already running.

Parameters

op	The operation to add.

7.63.3.2 disableAlpha()

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

Configures blit-op hardware for solid operation (no alpha-blending)

7.63.3.3 enableAlpha()

```
void enableAlpha (
            uint8_t alpha )  [protected], [virtual]
```

Configures blit-op hardware for alpha-blending.

Parameters

alpha	The alpha-blending value to apply.

7.63.3.4 enableCopyWithTransparentPixels()

```
void enableCopyWithTransparentPixels (
            uint8_t alpha )  [protected], [virtual]
```

Configures blit-op hardware for alpha-blending while simultaneously skipping transparent pixels.

Parameters

alpha	The alpha-blending value to apply.

7.63.3.5 execute()

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

Performs a queued blit-op.

7.63.3.6 executeCompleted()

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

To be called when blit-op has been performed.

7.63.3.7 flush()

```
void flush ( )  [inline], [virtual]
```

This function blocks until all DMA transfers in the queue have been completed.

Reimplemented in NoDMA.

7.63.3.8 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.

7.63.3.9 getBlitCaps()

```
BlitOperations getBlitCaps ( )  [pure virtual]
```

Gets the blit capabilities of this DMA.

Returns

The blit operations supported by this DMA implementation.

Implemented in NoDMA.

7.63.3.10 getDMAType()

```
DMAType getDMAType (
            void  )  [inline], [virtual]
```

Function for obtaining the DMA type of the concrete DMA_Interface implementation. As default, will return DMA_↩ TYPE_GENERIC type value.

Returns

a DMAType value of the concrete DMA_Interface implementation.

7.63.3.11 initialize()

```
void initialize ( )  [inline], [virtual]
```

Perform initialization. Does nothing in this base class.

7.63.3.12 isDmaQueueEmpty()

```
uint8_t isDmaQueueEmpty ( )
```

Query if the DMA queue is empty.

Returns

1 if DMA queue is empty, else 0.

7.63.3.13 isDmaQueueFull()

```
uint8_t isDmaQueueFull ( )
```

Query if the DMA queue is full.

Returns

1 if DMA queue is full, else 0.

7.63.3.14 isDMARunning()

```
bool isDMARunning ( )  [inline]
```

Query if the DMA is running.

Returns

true if a DMA operation is currently in progress.

7.63.3.15 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.

7.63.3.16 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

allowed	true if DMA transfers are allowed.

7.63.3.17 setupDataCopy()

```
void setupDataCopy (
            const BlitOp & blitOp )  [protected], [pure virtual]
```

Configures blit-op hardware for a 2D copy as specified by blitOp.

Parameters

blitOp	The operation to execute.

Implemented in NoDMA.

7.63.3.18 setupDataFill()

```
void setupDataFill (
            const BlitOp & blitOp )  [protected], [pure virtual]
```

Configures blit-op hardware for a 2D fill as specified by blitOp.

Parameters

blitOp	The operation to execute.

Implemented in NoDMA.

7.63.3.19 signalDMAInterrupt()

```
void signalDMAInterrupt ( )  [pure virtual]
```

This function is called automatically by the framework when a DMA interrupt has been received.

Implemented in NoDMA.

7.63.3.20 start()

```
void start ( )  [virtual]
```

Signals that DMA transfers can start. If any elements are in the queue, start it.

7.63.3.21 waitForFrameBufferSemaphore()

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

Waits until frame buffer semaphore is available (i.e. neither DMA or application is accessing the frame buffer).

7.64 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.

7.64.1 Detailed Description

This class provides an interface for a FIFO (circular) list used by DMA_Interface and descendants for storing BlitOp's.

7.64.2 Constructor & Destructor Documentation

7.64.2.1 ~DMA_Queue()

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

Destructor.

7.64.2.2 DMA_Queue()

```
DMA_Queue ( )  [inline], [protected]
```

Default constructor.

7.64.3 Member Function Documentation

7.64.3.1 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.

7.64.3.2 isEmpty()

```
bool isEmpty ( )  [pure virtual]
```

Query if this object is empty.

Returns

true if the queue is empty.

Implemented in LockFreeDMA_Queue.

7.64.3.3 isFull()

```
bool isFull ( )  [pure virtual]
```

Query if this object is full.

Returns

true if the queue is full.

Implemented in LockFreeDMA_Queue.

7.64.3.4 pop()

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

Pops an element from the queue.

Implemented in LockFreeDMA_Queue.

7.64.3.5 pushCopyOf()

```
void pushCopyOf (
            const BlitOp & op )  [pure virtual]
```

Adds the specified blitop to the queue.

Parameters

op	The blitop to add.

Implemented in LockFreeDMA_Queue.

7.65 DragEvent Class Reference

A drag event.

```
#include <touchgfx/events/DragEvent.hpp>
```

Public Types

- enum DragEventType { **DRAGGED** }

  The drag event types.

Public Member Functions

- DragEvent (DragEventType type, int16_t oldX, int16_t oldY, int16_t newX, int16_t newY)

  Constructor.
- virtual ~DragEvent ()

  Destructor.
- int16_t getOldX () const

  Gets the x coordinate where the drag operation was started (dragged from).
- int16_t getOldY () const

  Gets the y coordinate where the drag operation was started (dragged from).
- int16_t getNewX () const

  Gets the new x coordinate (dragged to).
- int16_t getNewY () const

  Gets the new x coordinate (dragged to).
- DragEventType getType () const

  Gets the type of this drag event.
- int16_t getDeltaX () const

  Gets the distance in x coordinates (how long was the drag).
- int16_t getDeltaY () const

  Gets the distance in y coordinates (how long was the drag).
- virtual Event::EventType getEventType ()

  Gets event type.

7.65.1 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

7.65.2 Constructor & Destructor Documentation

7.65.2.1 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

type	The type of the drag event.
oldX	The x coordinate of the drag start position (dragged from)
oldY	The y coordinate of the drag start position (dragged from)
newX	The x coordinate of the new position (dragged to)
newY	The y coordinate of the new position (dragged to)

7.65.2.2 ~DragEvent()

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

Destructor.

7.65.3 Member Function Documentation

7.65.3.1 getDeltaX()

```
int16_t getDeltaX ( ) const  [inline]
```

Gets the distance in x coordinates (how long was the drag).

Returns

The distance of this drag event.

7.65.3.2 getDeltaY()

```
int16_t getDeltaY ( ) const  [inline]
```

Gets the distance in y coordinates (how long was the drag).

Returns

The distance of this drag event.

7.65.3.3 getEventType()

```
Event::EventType getEventType ( )  [inline], [virtual]
```

Gets event type.

Returns

The type of this event.

Implements Event.

7.65.3.4 getNewX()

```
int16_t getNewX ( ) const  [inline]
```

Gets the new x coordinate (dragged to).

Returns

The new x coordinate (dragged to).

7.65.3.5 getNewY()

```
int16_t getNewY ( ) const  [inline]
```

Gets the new x coordinate (dragged to).

Returns

The new y coordinate (dragged to).

7.65.3.6 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).

7.65.3.7 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).

7.65.3.8 getType()

```
DragEventType getType ( ) const  [inline]
```

Gets the type of this drag event.

Returns

The type of this drag event.

7.66 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.

7.66.1 Detailed Description

template<class T>
class touchgfx::Draggable< T >

Mix-in class that extends a class to become draggable.

Template Parameters

T	specifies the type to extend with the draggable behavior.

7.66.2 Constructor & Destructor Documentation

7.66.2.1 Draggable()

```
Draggable ( )  [inline]
```

Default constructor.

7.66.2.2 ~Draggable()

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

Destructor.

7.66.3 Member Function Documentation

7.66.3.1 handleDragEvent()

```
void handleDragEvent (
            const DragEvent & evt )  [inline], [virtual]
```

Called when dragging the draggable object. The object is moved according to the drag event.

Parameters

evt	The drag event.

Reimplemented in Snapper< T >.

7.67 Drawable Class Reference

The Drawable class is an abstract definition of something that can be drawn.

`#include <touchgfx/Drawable.hpp>`

Public Types

- enum DrawableType {
  TYPE_DRAWABLE, **TYPE_WIDGET**, **TYPE_ABSTRACTBUTTON**, **TYPE_ANIMATEDIMAGE**, **TYPE_BOX**, **TYPE_BUTTON**, **TYPE_BUTTONWITHICON**,
  TYPE_BUTTONWITHLABEL, **TYPE_IMAGE**, **TYPE_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.
- void drawToDynamicBitmap (BitmapId id)

  Render the Drawable object into a dynamic bitmap.

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.

7.67.1 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

7.67.2 Member Enumeration Documentation

7.67.2.1 DrawableType

```
enum enum DrawableType
```

Enum defining Drawable types. To be used by automated GUI testing to determine class type of a Drawable object.

7.67.3 Constructor & Destructor Documentation

7.67.3.1 Drawable()

```
Drawable ( )  [inline]
```

Default constructor.

7.67.3.2 ~Drawable()

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

Destructor.

7.67.4 Member Function Documentation

7.67.4.1 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.

7.67.4.2 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

invalidatedArea	The subregion of this drawable that needs to be redrawn, expressed in coordinates relative to its parent (e.g. for a complete redraw, invalidatedArea will be (0, 0, width, height).

Implemented in Keyboard, TextArea, TextAreaWithTwoWildcards, BoxWithBorder, ButtonWithLabel, ScalableImage, Container, TiledImage, ButtonWithIcon, Box, CanvasWidget, TextureMapper, TextAreaWithOneWildcard, Image,

RadioButton, SnapshotWidget, CoverTransition< templateDirection >::FullSolidRect, Button, PixelDataWidget, and TouchArea.

7.67.4.3 drawToDynamicBitmap()

```
void drawToDynamicBitmap (
             BitmapId id )
```

Render the Drawable object into a dynamic bitmap.

Parameters

id	The target dynamic bitmap to use for rendering.

7.67.4.4 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.

7.67.4.5 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.

7.67.4.6 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.

7.67.4.7 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.

7.67.4.8 getHeight()

```
int16_t getHeight ( ) const  [inline]
```

Gets the height of this drawable.

Returns

The height of this drawable.

7.67.4.9 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

	x	The point of intersection expressed in coordinates relative to the parent.
	y	The point of intersection expressed in coordinates relative to the parent.
out	*last*	Result will be placed here.

Implemented in ScrollableContainer, Container, and Widget.

7.67.4.10 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.

7.67.4.11 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.

7.67.4.12 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.

7.67.4.13 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 Cover↩ Transition< templateDirection >::FullSolidRect.

7.67.4.14 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.

7.67.4.15 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, ButtonWithLabel, TiledImage, TextAreaWithOneWildcard, Box, ButtonWithIcon, ListLayout, Image, SnapshotWidget, Button, TouchArea, ToggleButton, AbstractButton, and Widget.

7.67.4.16 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.

7.67.4.17 getWidth()

```
int16_t getWidth ( ) const  [inline]
```

Gets the width of this drawable.

Returns

The width of this drawable.

7.67.4.18 getX()

```
int16_t getX ( ) const  [inline]
```

Gets the x coordinate of this drawable.

Returns

The x value, relative to the parent.

7.67.4.19 getY()

```
int16_t getY ( ) const  [inline]
```

Gets the y coordinate of this drawable.

Returns

The y value, relative to the parent.

7.67.4.20 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

evt	The ClickEvent received from the HAL.

Reimplemented in Slider, Keyboard, ScrollableContainer, ScrollList, ScrollWheelBase, RepeatButton, RepeatButtonTrigger, ToggleButton, ToggleButtonTrigger, RadioButton, TouchArea, AbstractButton, ClickButtonTrigger, TouchButtonTrigger, and SwipeContainer.

7.67.4.21 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

evt	The DragEvent received from the HAL.

Reimplemented in ScrollBase, Slider, Keyboard, ScrollableContainer, ScrollWheelBase, TouchArea, and SwipeContainer.

7.67.4.22 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

evt	The GestureEvent received from the HAL.

Reimplemented in ScrollBase, ScrollableContainer, ScrollWheelBase, and SwipeContainer.

7.67.4.23 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.

7.67.4.24 invalidate()

```
void invalidate ( ) const  [virtual]
```

Tell the framework that this entire drawable needs to be redrawn.

See also

invalidateRect

Reimplemented in CanvasWidget.

7.67.4.25 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	*invalidatedArea*	The area of this drawable to redraw expressed in coordinates relative to its parent (e.g. to request a complete redraw, invalidatedArea will be (0, 0, width, height).

Reimplemented in CacheableContainer.

7.67.4.26 isTouchable()

```
bool isTouchable ( ) const  [inline]
```

Gets whether this Drawable receives touch events or not.

Returns

True if touch events are received.

See also

setTouchable

7.67.4.27 isVisible()

```
bool isVisible ( ) const  [inline]
```

Gets whether this Drawable is visible.

Returns

true if the Drawable is visible.
True if visible.

See also

setVisible

7.67.4.28 moveRelative()

```
void moveRelative (
            int16_t x,
            int16_t y )  [virtual]
```

Moves the drawable.

Note

Will redraw the appropriate areas of the screen.

Parameters

x	The relative position to move to.
y	The relative position to move to.

7.67.4.29 moveTo()

```
void moveTo (
             int16_t x,
             int16_t y )  [inline], [virtual]
```

Moves the drawable.

Note

Will redraw the appropriate areas of the screen.

Parameters

x	The absolute position to move to.
y	The absolute position to move to.

7.67.4.30 resetDrawChainCache()

```
void resetDrawChainCache ( )  [inline], [protected]
```

Reset cached coordinate data.

Note

For TouchGFX internal use only.

7.67.4.31 setHeight()

```
void setHeight (
             int16_t height )  [inline], [virtual]
```

Sets the height of this drawable.

Note

Changing this does not automatically yield a redraw.

Parameters

height	The new height.

Reimplemented in ZoomAnimationImage, DrawableList, DigitalClock, ScrollWheelWithSelectionStyle, and ScrollBase.

7.67.4.32 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

x	The x coordinate of this Drawable.
y	The y coordinate of this Drawable.
width	The width of this Drawable.
height	The height of this Drawable.

Reimplemented in ZoomAnimationImage.

7.67.4.33 setTouchable()

```
void setTouchable (
            bool touch )  [inline]
```

Controls whether this Drawable receives touch events or not.

Parameters

touch	If true it will receive touch events, if false it will not.

7.67.4.34 setupDrawChain()

```
void setupDrawChain (
            const Rect & invalidatedArea,
            Drawable ** nextPreviousElement )  [inline], [protected], [virtual]
```

Configure linked list for draw chain.

Note

For TouchGFX internal use only.

Parameters

	invalidatedArea	Include drawables that intersect with this area only.
in,out	*nextPreviousElement*	Modifiable element in linked list.

Reimplemented in Keyboard, Container, CacheableContainer::CachedImage, and CacheableContainer.

7.67.4.35 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

vis	true if this Drawable should be visible. By default, drawables are visible unless this function has been called with false as argument.

7.67.4.36 setWidth()

```
void setWidth (
            int16_t width )  [inline], [virtual]
```

Sets the width of this drawable.

Note

Changing this does not automatically yield a redraw.

Parameters

width	The new width.

Reimplemented in ZoomAnimationImage, DrawableList, DigitalClock, ScrollBase, and ScrollWheelWithSelectionStyle.

7.67.4.37 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

x	The new x value, relative to the parent. A negative value is allowed.

7.67.4.38 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

x	The new x value, relative to the parent. A negative value is allowed.
y	The new y value, relative to the parent. A negative value is allowed.

7.67.4.39 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

y	The new y value, relative to the parent. A negative value is allowed.

7.67.4.40 translateRectToAbsolute()

```
void translateRectToAbsolute (
            Rect & r ) const  [virtual]
```

Helper function for converting a specified subregion of this Drawable to absolute coordinates.

Parameters

in,out	*r*	The Rect to translate.

7.68 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) const

  Gets item stored in a given Drawable.
- int16_t getDrawableIndices (int16_t itemIndex, int16_t *drawableIndexArray, int16_t arraySize) const

  Gets drawable indices.
- int16_t getDrawableIndex (int16_t itemIndex, int16_t prevDrawableIndex=-1) const

  Gets the drawable index of an item.
- void refreshDrawables ()

  Refresh drawables.
- void itemChanged (int16_t itemIndex)

  Item changed.

Additional Inherited Members

7.68.1 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.

7.68.2 Constructor & Destructor Documentation

7.68.2.1 DrawableList()

```
DrawableList ( )
```

Default constructor.

7.68.2.2 ~DrawableList()

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

Destructor.

7.68.3 Member Function Documentation

7.68.3.1 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

7.68.3.2 getDrawableIndex()

```
int16_t getDrawableIndex (
            int16_t itemIndex,
            int16_t prevDrawableIndex = -1 ) const
```

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

itemIndex	Index of the item.
prevDrawableIndex	(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

7.68.3.3 getDrawableIndices()

```
int16_t getDrawableIndices (
            int16_t itemIndex,
            int16_t * drawableIndexArray,
            int16_t arraySize ) const
```

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

	itemIndex	Zero-based index of the item.
out	*drawableIndexArray*	Array where the drawable indices are stored.
	arraySize	Size of drawable array.

Returns

The number of drawable indices found.

See also

getFirstDrawableIndex
setCircular
getDrawbleIndex

7.68.3.4 getDrawableMargin()

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

Gets drawable margin as set by setDrawables.

Returns

The drawable margin.

7.68.3.5 getDrawableSize()

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

Gets drawable size as set by setDrawables.

Returns

The drawable size.

See also

setDrawables

7.68.3.6 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

7.68.3.7 getItemIndex()

```
int16_t getItemIndex (
            int16_t drawableIndex ) const
```

Gets item stored in a given Drawable.

Parameters

drawableIndex	Zero-based index of the drawable.

Returns

The item index.

7.68.3.8 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
setDrawableSize
getDrawableMargin

7.68.3.9 getNumberOfDrawables()

```
int16_t getNumberOfDrawables ( ) const
```

Gets number of drawables, as set using setDrawables().

Returns

The number of drawables.

See also

setDrawables

7.68.3.10 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

7.68.3.11 getOffset()

```
int32_t getOffset ( ) const
```

Gets offset, as previously set using setOffset().

Returns

The virtual offset.

See also

setOffset

7.68.3.12 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

7.68.3.13 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, possible more than once for cyclic lists, each drawable is request to refresh its content to reflect the new value.

Parameters

itemIndex	Zero-based index of the item.

7.68.3.14 refreshDrawables()

```
void refreshDrawables ( )
```

Refresh drawables. Useful to call if the number or items, their size or other properties have changed.

7.68.3.15 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

circular	True if the list should be circular, false if the list should not be circular.

See also

getCircular

7.68.3.16 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	*drawableListItems*	Number of drawables allocated.
in	*drawableItemIndexOffset*	A callback to get access to a drawable.
in	*updateDrawableCallback*	A callback to update the contents of a drawable.

See also

getRequiredNumberOfDrawables

7.68.3.17 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

drawableSize	The size of the drawable.
drawableMargin	The margin around drawables (before as well as after the drawable).

7.68.3.18 setHeight()

```
void setHeight (
            int16_t height )  [virtual]
```

Parameters

height	The height.

Reimplemented from Drawable.

7.68.3.19 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

horizontal	True to align drawables horizontal, false to align drawables vertically.

Note

Default value is false, i.e. vertical layout.

See also

getHorizontal

7.68.3.20 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

numberOfItems	Number of items.

Note

The DrawableList is refreshed to reflect the change.

7.68.3.21 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

ofs	The virtual coordinate.

See also

getOffset
setDrawables

7.68.3.22 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

width	The width.

Reimplemented from Drawable.

7.69 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.

7.69.1 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

TYPE	Type of the drawables. Can be a simple drawable, such as Image or a more complex container.
SIZE	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.

7.69.2 Constructor & Destructor Documentation

7.69.2.1 DrawableListItems()

```
DrawableListItems ( )  [inline]
```

Default constructor.

7.69.2.2 ~DrawableListItems()

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

Destructor.

7.69.3 Member Function Documentation

7.69.3.1 getDrawable()

```
Drawable * getDrawable (
            int16_t index )  [inline], [virtual]
```

Gets the address of an element.

Parameters

index	Zero-based index of the drawable.

Returns

The drawable for the given index.

Implements DrawableListItemsInterface.

7.69.3.2 getNumberOfDrawables()

```
virtual int16_t getNumberOfDrawables ( )  [inline], [virtual]
```

Gets number of drawables.

Returns

The number of drawables.

Implements DrawableListItemsInterface.

7.69.3.3 operator[]()

```
TYPE & operator[] (
            int index )  [inline]
```

Array indexer operator.

Parameters

index	Zero-based index of elements to access.

Returns

The indexed value.

7.70 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.

7.70.1 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
ScrollWheelWithSelectionStyle::setDrawables

7.70.2 Constructor & Destructor Documentation

7.70.2.1 ~DrawableListItemsInterface()

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

Destuctor.

7.70.3 Member Function Documentation

7.70.3.1 getDrawable()

```
Drawable * getDrawable (
            int16_t index )  [pure virtual]
```

Gets a drawable at a given index.

Parameters

index	Zero-based index of the drawable.

Returns

Null if it fails, else the drawable.

Implemented in DrawableListItems< TYPE, SIZE >.

7.70.3.2 getNumberOfDrawables()

```
int16_t getNumberOfDrawables ( )  [pure virtual]
```

Gets number of drawables.

Returns

The number of drawables.

Implemented in DrawableListItems< TYPE, SIZE >.

7.71 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.

7.72 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.
- uint8_t format: 5
  Determine the format of the data.
- uint8_t inuse: 1
  Zero if not in use.
- uint8_t extra: 2
  Extra data field, dependent on format.

7.72.1 Detailed Description

Data of a dynamic bitmap.

7.73 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)

7.73.1 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/

7.73.2 Member Function Documentation

7.73.2.1 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.2 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.3 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.4 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.5 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.6 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.7 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.8 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.9 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

t	Time. The current time or step.
b	Beginning. The beginning value.

Parameters

c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.10 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.11 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.12 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.13 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.14 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.15 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.16 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

t	Time. The current time or step.
b	Beginning. The beginning value.

Parameters

c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.17 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.18 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.19 linearEaseIn()

```
static int16_t linearEaseIn (
            uint16_t t,
            int16_t b,
            int16_t c,
            uint16_t d )  [static]
```

Simple linear tweening - no easing.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.20 linearEaseInOut()

```
static int16_t linearEaseInOut (
            uint16_t t,
            int16_t b,
            int16_t c,
            uint16_t d )  [static]
```

Simple linear tweening - no easing.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.21 linearEaseNone()

```
static int16_t linearEaseNone (
            uint16_t t,
```

```
            int16_t b,
            int16_t c,
            uint16_t d )  [static]
```

Simple linear tweening - no easing.

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.22 linearEaseOut()

```
static int16_t linearEaseOut (
            uint16_t t,
            int16_t b,
            int16_t c,
            uint16_t d )  [static]
```

Parameters

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.23 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.24 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.25 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.26 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.27 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.28 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

t	Time. The current time or step.
b	Beginning. The beginning value.

Parameters

c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.29 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.30 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.31 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.32 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.33 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.73.2.34 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

t	Time. The current time or step.
b	Beginning. The beginning value.
c	Change. The change between the beginning value and the destination value.
d	Duration. The total time or total number of steps.

Returns

The current value as a function of the current time or step.

7.74 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.

7.74.1 Constructor & Destructor Documentation

7.74.1.1 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

gradients	The gradients for the triangle.
vertices	The vertices for the triangle.
top	The index in the vertices array of the top vertex of this edge.
bottom	The index in the vertices array of the bottom vertex of this edge.

7.74.2 Member Function Documentation

7.74.2.1 step() [1/2]

```
int step ( )  [inline]
```

Perform a step along the edge.

Returns

the Height.

7.74.2.2 step() [2/2]

```
int step (
            int steps )  [inline]
```

Performs a number of steps along the edge.

Parameters

steps	The number of steps the perform.

Returns

height.

7.75 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.

7.75.1 Detailed Description

Simple base class for events.

7.75.2 Member Enumeration Documentation

7.75.2.1 EventType

```
enum enum EventType
```

The events types.

Enumerator

EVENT_CLICK	A click.
EVENT_DRAG	A drag.
EVENT_GESTURE	A gesture.

7.75.3 Constructor & Destructor Documentation

7.75.3.1 ~Event()

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

Destructor.

7.75.4 Member Function Documentation

7.75.4.1 getEventType()

```
EventType getEventType ( )  [pure virtual]
```

Gets event type.

Returns

The type of this event.

Implemented in ClickEvent, DragEvent, and GestureEvent.

7.76 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.

7.76.1 Detailed Description

template<class T>
class touchgfx::FadeAnimator< T >

A FadeAnimator makes the template class T able to animate an alpha fade from its current alpha value to a specified end alpha value. The alpha development can be described by supplying an EasingEquation. The FadeAnimator performs a callback when the animation has finished.

This mixin can be used on any Drawable that has a 'void setAlpha(uint8_t)' and a 'uint8_t getAlpha()' method.

Template Parameters

T	Specifies the type should have the fade animation capability.

7.76.2 Constructor & Destructor Documentation

7.76.2.1 FadeAnimator()

```
FadeAnimator ( )  [inline]
```

Default constructor. Creates and initialize the FadeAnimator.

7.76.2.2 ~FadeAnimator()

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

Destructor. Destroys the FadeAnimator.

7.76.3 Member Function Documentation

7.76.3.1 clearFadeAnimationEndedAction()

```
void clearFadeAnimationEndedAction ( )  [inline]
```

Clears the fade animation ended action previously set by setFadeAnimationEndedAction.

See also

setFadeAnimationEndedAction

7.76.3.2 getFadeAnimationDelay()

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

Gets the current animation delay.

Returns

The current animation delay.

7.76.3.3 handleTickEvent()

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

The tick handler that handles the actual animation steps.

7.76.3.4 isFadeAnimationRunning()

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

Gets whether or not the fade animation is running.

Returns

true if the fade animation is running.

7.76.3.5 isRunning()

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

Gets whether or not the fade animation is running.

Returns

true if the fade animation is running.

7.76.3.6 nextFadeAnimationStep()

```
void nextFadeAnimationStep ( )  [inline], [protected]
```

Execute next step in fade animation and stop the timer if necessary.

7.76.3.7 setFadeAnimationDelay()

```
void setFadeAnimationDelay (
             uint16_t delay )  [inline], [virtual]
```

Sets a delay on animations done by the FadeAnimator.

Parameters

delay	The delay in ticks.

7.76.3.8 setFadeAnimationEndedAction()

```
void setFadeAnimationEndedAction (
            GenericCallback< const FadeAnimator< T > & > & callback )  [inline]
```

Associates an action to be performed when the animation ends.

Parameters

callback	The callback to be executed. The callback will be given a reference to the FadeAnimator.

See also

GenericCallback

7.76.3.9 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

endAlpha	The alpha value of T at animation end.
duration	The duration of the animation measured in ticks.
alphaProgressionEquation	The equation that describes the development of the alpha value during the animation. Default = EasingEquations::linearEaseNone.

7.77 FlashDataReader Class Reference

This class is an abstract interface for a class reading data from a flash.

`#include <touchgfx/hal/FlashDataReader.hpp>`

Public Member Functions

- virtual bool addressIsAddressable (const void *address)=0

  Compute if an address is directly addressable by the MCU.
- virtual void copyData (const void *src, void *dst, uint32_t bytes)=0

  Copy data from flash to a buffer.
- virtual void startFlashLineRead (const void *src, uint32_t bytes)

  Initiate a read operation from flash to a buffer.
- virtual const uint8_t * waitFlashReadComplete ()

  Waits until the previous startFlashLineRead operation is complete.

7.77.1 Detailed Description

This class is an abstract interface for a class reading data from a flash. The flash can be any type, but is mostly used for flashes that are not memory mapped. Applications must implement access to the flash through this interface.

7.77.2 Member Function Documentation

7.77.2.1 addressIsAddressable()

```
bool addressIsAddressable (
            const void * address )  [pure virtual]
```

Compute if an address is directly addressable by the MCU. The data is addressable it should be read direct through a pointer and not through this interface.

Parameters

address	The address in the flash.

Returns

True if the address is addressable by the MCU.

7.77.2.2 copyData()

```
void copyData (
            const void * src,
            void * dst,
            uint32_t bytes )  [pure virtual]
```

Copy data from flash to a buffer. This must be a synchrony method that does not return until the copy is done.

Parameters

src	Address of source data in the flash
dst	Address of destination buffer in RAM
bytes	Number of bytes to copy.

7.77.2.3 startFlashLineRead()

```
void startFlashLineRead (
            const void * src,
            uint32_t bytes )  [virtual]
```

Initiate a read operation from flash to a buffer. This can be an asynchrony operation that is still running after this function returns. Buffers must be handled by the subclass. LCD16bbbSerialFlash will at most copy 4 bytes times the width of the display.

Parameters

src	Address of source data in the flash
bytes	Number of bytes to copy.

7.77.2.4 waitFlashReadComplete()

```
const uint8_t * waitFlashReadComplete ( )  [virtual]
```

Waits until the previous startFlashLineRead operation is complete. If the startFlashLineRead method is asynchrony, this method must wait until the previous operation has completed.

Returns

The address of a buffer containing the read data.

7.78 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.
- virtual uint8_t getDataFormatA4 () const

Are the glyphs saved using ST A4 format.

- 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.

- virtual const uint16_t * getGSUBTable () const

  Gets GSUB table.

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 dataFormatA4, 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: 7

  The number of bits per pixel.

- uint8_t a4: 1

  Are glyphs encoded using A4 format.

- 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 falllbackCharacter

  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)

7.78.1 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.

7.78.2 Constructor & Destructor Documentation

7.78.2.1 Font()

```
Font (
            uint16_t height,
            uint8_t pixBelowBase,
            uint8_t bitsPerPixel,
            uint8_t dataFormatA4,
            uint8_t maxLeft,
            uint8_t maxRight,
            const Unicode::UnicodeChar fallbackChar,
            const Unicode::UnicodeChar ellipsisChar )  [inline], [protected]
```

The protected constructor of a Font.

Parameters

height	The font height in pixels.
pixBelowBase	The number of pixels below the base line.
bitsPerPixel	The number of bits per pixel.
dataFormatA4	The glyphs are saved using ST A4 format.
maxLeft	The maximum left extend for a glyph in the font.
maxRight	The maximum right extend for a glyph in the font.
fallbackChar	The fallback character for the typography in case no glyph is available.
ellipsisChar	The ellipsis character used for truncating long texts.

7.78.3 Member Function Documentation

7.78.3.1 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.

7.78.3.2 getCharWidth()

```
uint16_t getCharWidth (
            const Unicode::UnicodeChar c ) const  [virtual]
```

Gets the width in pixels of the specified character.

Parameters

c	The unicode character.

Returns

The width in pixels of the specified character.

7.78.3.3 getDataFormatA4()

```
uint8_t getDataFormatA4 ( ) const  [inline], [virtual]
```

Are the glyphs saved using ST A4 format.

Returns

True if the font is stored using A4 format, false otherwise.

7.78.3.4 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.

7.78.3.5 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.

7.78.3.6 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.

7.78.3.7 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. Please note that in case of Thai letters where diacritics can be placed relative to the previous character(s), please use TextProvider::getGlyph() instead as it will make a GlyphNode that will be correct with respect to X/Y position.

Parameters

	unicode	The character to look up.
	pixelData	Pointer to the pixel data for the glyph if the glyph is found. This is set by this method.
out	*bitsPerPixel*	Reference where to place the number of bits per pixel.

Returns

A pointer to the glyph node or null if the glyph was not found.

Implemented in ConstFont.

7.78.3.8 getGlyph() [2/2]

```
const GlyphNode * getGlyph (
            Unicode::UnicodeChar unicode ) const  [inline], [virtual]
```

Gets the glyph data associated with the specified unicode. Please note that in case of Thai letters where diacritics can be placed relative to the previous character(s), please use TextProvider::getGlyph() instead as it will make a GlyphNode that will be correct with respect to X/Y position.

Parameters

unicode	The character to look up.

Returns

A pointer to the glyph node or null if the glyph was not found.

See also

TextProvider::getGlyph

7.78.3.9 getGSUBTable()

```
uint16_t * getGSUBTable ( ) const  [inline], [virtual]
```

Gets GSUB table.

Returns

The GSUB table or null if font has GSUB no table

7.78.3.10 getKerning()

```
int8_t getKerning (
            Unicode::UnicodeChar prevChar,
            const GlyphNode * glyph ) const  [inline], [virtual]
```

Gets the kerning distance between two characters.

Parameters

prevChar	The unicode value of the previous character.
glyph	the glyph object for the current character.

Returns

The kerning distance between prevChar and glyph char.

Reimplemented in ConstFont, and InternalFlashFont.

7.78.3.11 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.

7.78.3.12 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.

7.78.3.13 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

text	A zero-terminated unicode string.
...	Variable arguments providing additional information.

Returns

The height if the given text.

7.78.3.14 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.

7.78.3.15 getNumberOfLines()

```
uint16_t getNumberOfLines (
             const Unicode::UnicodeChar * text,
              ...  )  const  [virtual]
```

Count the number of lines in a given text.

Parameters

text	The text.
...	Variable arguments providing additional information.

Returns

The number of lines.

7.78.3.16 getSpacingAbove()

```
uint8_t getSpacingAbove (
            const Unicode::UnicodeChar * text,
             ...  )  const  [virtual]
```

Gets the number of blank pixels at the top of the given text.

Parameters

text	A zero-terminated unicode string.
...	Variable arguments providing additional information.

Returns

The number of blank pixels above the text.

7.78.3.17 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

text	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
...	Variable arguments providing additional information.

Returns

The width in pixels of the longest line of the specified string.

7.78.3.18 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 with of "10 20 30" (with kerning between all characters) but for TEXT_DIRECTION_RTL it is calculated as "10"+" "+"20"+" "+"30" (with kerning only between characters in the substrings and not between substrings). For most fonts there might not be a difference between the two calculations, but some fonts might cause different results.

Parameters

textDirection	The text direction.
text	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
...	Variable arguments providing additional information.

Returns

The width in pixels of the longest line of the specified string.

7.78.3.19 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

textDirection	The text direction.
text	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
pArg	Variable arguments providing additional information.

Returns

The width in pixels of the longest line of the specified string.

7.78.3.20 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

textDirection	The text direction.

Parameters

text	A zero-terminated unicode string with arguments to insert if the text contains wildcards.
pArg	The argument.

Returns

The string width RTL.

7.79 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.

7.79.1 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.

7.79.2 Member Function Documentation

7.79.2.1 getFont()

```
static Font * getFont (
            FontId fontId )  [static]
```

Gets a font.

Parameters

fontId	The font id of the font to get.

Returns

The font with a font id of fontId.

7.79.2.2 setFontProvider()

```
static void setFontProvider (
            FontProvider * fontProvider )  [static]
```

Sets the font provider. Must be initialized with the appropriate FontProvider by the application.

Parameters

in	*fontProvider*	Sets the font provider. Must be initialized with the appropriate FontProvider by the application.

7.80 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.

7.80.1 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.

7.80.2 Constructor & Destructor Documentation

7.80.2.1 ~FontProvider()

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

Destructor.

7.80.3 Member Function Documentation

7.80.3.1 getFont()

```
Font * getFont (
            FontId fontId )  [pure virtual]
```

Gets a font.

Parameters

fontId	The font id of the font to get.

Returns

The font with a font id of fontId.

7.81 FrameBufferAllocator Class Reference

This class is an abstract interface for a class allocating partial framebuffer blocks.

`#include <touchgfx/hal/FrameBufferAllocator.hpp>`

Public Member Functions

- virtual uint16_t allocateBlock (const uint16_t x, const uint16_t y, const uint16_t width, const uint16_t height, uint8_t **block)=0

  Allocates a framebuffer block.
- virtual void markBlockReadyForTransfer ()=0

  Marks a previously allocated block as ready to be transferred to the LCD.
- virtual bool hasBlockReadyForTransfer ()=0

  Check if a block is ready for transfer to the LCD.
- virtual const uint8_t * getBlockForTransfer (Rect &rect)=0

  Get the block ready for transfer.
- virtual void freeBlockAfterTransfer ()=0

  Free a block after transfer to the LCD.

7.81.1 Detailed Description

This class is an abstract interface for a class allocating partial framebuffer blocks. The interface must be implemented by a subclass.

See also

SingleBlockAllocator, ManyBlockAllocator

7.81.2 Member Function Documentation

7.81.2.1 allocateBlock()

```
uint16_t allocateBlock (
             const uint16_t x,
             const uint16_t y,
             const uint16_t width,
             const uint16_t height,
             uint8_t ** block )  [pure virtual]
```

Allocates a framebuffer block. The block will have at least the width requested. The height of the allocated block can be lower than requested if not enough memory is available.

Parameters

x	The absolute x coordinate of the block on the screen.
y	The absolute y coordinate of the block on the screen.
width	The width of the block.

Parameters

height	The height of the block.
block	Pointer to pointer to return the block address in.

Returns

The height of the allocated block.

Implemented in ManyBlockAllocator< block_size, blocks, bytes_pr_pixel >, and SingleBlockAllocator< block_size, bytes_pr_pixel >.

7.81.2.2 freeBlockAfterTransfer()

```
void freeBlockAfterTransfer ( )  [pure virtual]
```

Marks a previously allocated block as transferred and ready to reuse.

Implemented in ManyBlockAllocator< block_size, blocks, bytes_pr_pixel >, and SingleBlockAllocator< block_size, bytes_pr_pixel >.

7.81.2.3 getBlockForTransfer()

```
const uint8_t * getBlockForTransfer (
            Rect & rect )  [pure virtual]
```

Get the block ready for transfer.

Parameters

rect	Reference to rect to write block x, y, width, and height.

Returns

Returns the address of the block ready for transfer.

Implemented in ManyBlockAllocator< block_size, blocks, bytes_pr_pixel >, and SingleBlockAllocator< block_size, bytes_pr_pixel >.

7.81.2.4 hasBlockReadyForTransfer()

```
bool hasBlockReadyForTransfer ( )  [pure virtual]
```

Check if a block is ready for transfer to the LCD.

Returns

True if a block is ready for transfer.

Implemented in ManyBlockAllocator< block_size, blocks, bytes_pr_pixel >, and SingleBlockAllocator< block_size, bytes_pr_pixel >.

7.81.2.5 markBlockReadyForTransfer()

```
void markBlockReadyForTransfer ( )  [pure virtual]
```

Marks a previously allocated block as ready to be transferred to the LCD.

Implemented in ManyBlockAllocator< block_size, blocks, bytes_pr_pixel >, and SingleBlockAllocator< block_size, bytes_pr_pixel >.

7.82 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

7.82.1 Member Function Documentation

7.82.1.1 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

invalidatedArea	The subregion of this drawable that needs to be redrawn, expressed in coordinates relative to its parent (e.g. for a complete redraw, invalidatedArea will be (0, 0, width, height).

Implements Drawable.

7.82.1.2 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.

7.83 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.

7.83.1 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

T1	The type of the first argument in the member function, or void if none.
T2	The type of the second argument in the member function, or void if none.
T3	The type of the third argument in the member function, or void if none.

7.83.2 Constructor & Destructor Documentation

7.83.2.1 ~GenericCallback()

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

Empty virtual destructor.

7.83.3 Member Function Documentation

7.83.3.1 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

val1	This value will be passed as the first argument in the function call.
val2	This value will be passed as the second argument in the function call.
val3	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 > &>.

7.83.3.2 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 > &>.

7.84 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.

7.84.1 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

T1	The type of the first argument in the member function, or void if none.
T2	The type of the second argument in the member function, or void if none.

7.84.2 Constructor & Destructor Documentation

7.84.2.1 ~GenericCallback()

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

Empty virtual destructor.

7.84.3 Member Function Documentation

7.84.3.1 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

val1	This value will be passed as the first argument in the function call.
val2	This value will be passed as the second argument in the function call.

7.84.3.2 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).

7.85 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.

7.85.1 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

T1	The type of the first argument in the member function, or void if none.

7.85.2 Constructor & Destructor Documentation

7.85.2.1 ~GenericCallback()

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

Empty virtual destructor.

7.85.3 Member Function Documentation

7.85.3.1 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

val1	This value will be passed as the first argument in the function call.

7.85.3.2 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).

7.86 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.

7.86.1 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.

7.86.2 Constructor & Destructor Documentation

7.86.2.1 ~GenericCallback()

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

Empty virtual destructor.

7.86.3 Member Function Documentation

7.86.3.1 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).

7.86.3.2 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).

7.87 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.

7.87.1 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

7.87.2 Member Enumeration Documentation

7.87.2.1 GestureType

```
enum enum GestureType
```

Enumerator

SWIPE_HORIZONTAL	An enum constant representing a horizontal swipe.
SWIPE_VERTICAL	An enum constant representing a vertical swipe.

7.87.3 Constructor & Destructor Documentation

7.87.3.1 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

t	The type of the gesture event.
v	The velocity of this gesture (swipe)
x_coord	The x coordinate of the gesture.
y_coord	The y coordinate of the gesture.

7.87.4 Member Function Documentation

7.87.4.1 getEventType()

```
Event::EventType getEventType ( )  [inline], [virtual]
```

Returns

The type of this event.

Implements Event.

7.87.4.2 getType()

```
GestureType getType ( ) const  [inline]
```

Gets the type of this gesture event.

Returns

The type of this gesture event.

7.87.4.3 getVelocity()

```
int16_t getVelocity ( ) const  [inline]
```

Gets the velocity of this gesture event.

Returns

The velocity of this gesture event.

7.87.4.4 getX()

```
int16_t getX ( ) const  [inline]
```

Gets the x coordinate of this gesture event.

Returns

The x coordinate of this gesture event.

7.87.4.5 getY()

```
int16_t getY ( ) const  [inline]
```

Gets the y coordinate of this gesture event.

Returns

The y coordinate of this gesture event.

7.88 Gestures Class Reference

This class implements the detection of gestures.

`#include <touchgfx/hal/Gestures.hpp>`

Public Member Functions

- Gestures ()

  Default constructor.
- void registerEventListener (UIEventListener &l)

  Register the event listener.
- void tick ()

  Has to be called during the timer tick.
- bool registerDragEvent (uint16_t oldX, uint16_t oldY, uint16_t newX, uint16_t newY)

  Register a drag event.
- void registerClickEvent (ClickEvent::ClickEventType evt, uint16_t x, uint16_t y)

  Register a click event and figure out if this is a drag event, too.
- void setDragThreshold (uint16_t val)

  Configure the threshold for reporting drag events.

7.88.1 Detailed Description

This class implements the detection of gestures.

7.88.2 Constructor & Destructor Documentation

7.88.2.1 Gestures()

```
Gestures ( )  [inline]
```

Default constructor. Does nothing.

7.88.3 Member Function Documentation

7.88.3.1 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

evt	The type of the click event.
x	The x coordinate of the click event.
y	The y coordinate of the click event.

7.88.3.2 registerDragEvent()

```
bool registerDragEvent (
             uint16_t oldX,
             uint16_t oldY,
             uint16_t newX,
             uint16_t newY )
```

Register a drag event.

Parameters

oldX	The x coordinate of the drag start position (dragged from)
oldY	The y coordinate of the drag start position (dragged from)
newX	The x coordinate of the new position (dragged to)
newY	The y coordinate of the new position (dragged to)

Returns

True if the drag exceeds threshold value (and therefore was reported as a drag), or false if the drag did not exceed threshold (and therefore was discarded).

7.88.3.3 registerEventListener()

```
void registerEventListener (
             UIEventListener & l )
```

Register the event listener.

Parameters

in	*l*	The EventListener to register.

7.88.3.4 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.

7.88.3.5 tick()

```
void tick ( )
```

Has to be called during the timer tick.

7.89 GlyphNode Struct Reference

struct providing information about a glyph.

`#include <touchgfx/Font.hpp>`

Public Member Functions

- uint16_t kerningTablePos () const

  Gets the "kerningTablePos" value where the 8th and 9th bits is stored in flags.
- 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.
- void setTop (int16_t newTop)

  Sets a top.
- uint16_t advance () const

  Gets the "advance" value where the 9th bit is stored in flags.

Public Attributes

- uint32_t dataOffset

  The index to the data of this glyph.
- Unicode::UnicodeChar unicode

  The unicode of this glyph.
- uint8_t _width

  Width of the actual glyph data.
- uint8_t _height

  Height of the actual glyph data.
- uint8_t _top

  Vertical offset from baseline of the glyph.
- int8_t left

  Horizontal offset from the left of the glyph.
- uint8_t _advance

  Width of the glyph (including space to the left and right)
- uint8_t _kerningTablePos

  Where are the kerning information for this glyph stored in the kerning table.
- uint8_t kerningTableSize

  How many entries are there in the kerning table (following kerningTablePos) for this glyph.
- uint8_t flags

  Additional glyph flags (font encoding and extra precision for width/height/top/advance)

7.89.1 Detailed Description

struct providing information about a glyph. Used by LCD when rendering.

7.89.2 Member Function Documentation

7.89.2.1 advance()

```
uint16_t advance ( ) const  [inline]
```

Gets the "advance" value where the 9th bit is stored in flags.

Returns

the right value of "advance".

7.89.2.2 height()

```
uint16_t height ( ) const  [inline]
```

Gets the "height" value where the 9th bit is stored in flags.

Returns

the right value of "height".

7.89.2.3 kerningTablePos()

```
uint16_t kerningTablePos ( ) const  [inline]
```

Gets the "kerningTablePos" value where the 8th and 9th bits is stored in flags.

Returns

the right value of "kerningTablePos".

7.89.2.4 setTop()

```
void setTop (
            int16_t newTop )  [inline]
```

Parameters

newTop	The new top.

7.89.2.5 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".

7.89.2.6 width()

```
uint16_t width ( ) const  [inline]
```

Gets the "width" value where the 9th bit is stored in flags.

Returns

the right value of "width".

7.90 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.

7.90.1 Detailed Description

Interface class for manipulating GPIOs in order to do performance measurements on target. Not used on the PC simulator.

7.90.2 Member Enumeration Documentation

7.90.2.1 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).

7.90.3 Member Function Documentation

7.90.3.1 clear()

```
static void clear (
            GPIO_ID id )  [static]
```

Sets a pin low.

Parameters

id	the pin to set.

7.90.3.2 get()

```
static bool get (
            GPIO_ID id )  [static]
```

Gets the state of a pin.

Parameters

id	the pin to get.

Returns

true if the pin is high, false otherwise.

7.90.3.3 init()

```
static void init ( )  [static]
```

Perform configuration of IO pins.

7.90.3.4 set()

```
static void set (
            GPIO_ID id )  [static]
```

Sets a pin high.

Parameters

id	the pin to set.

7.90.3.5 toggle()

```
static void toggle (
            GPIO_ID id )  [static]
```

Toggles a pin.

Parameters

id	the pin to toggle.

7.91 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.

7.91.1 Constructor & Destructor Documentation

7.91.1.1 Gradients()

```
Gradients (
            const Point3D * vertices )
```

Parameters

vertices	The vertices.

See also

Point3D

7.92 HAL Class Reference

Hardware Abstraction Layer.

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

Public Types

- enum FrameRefreshStrategy { REFRESH_STRATEGY_DEFAULT, REFRESH_STRATEGY_OPTIM_SINGLE_BUFFER_TFT_CTRL, REFRESH_STRATEGY_PARTIAL_FRAMEBUFFER }

  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.
- void blitCopy (const uint16_t *pSrc, const uint8_t *pClut, uint16_t x, uint16_t y, uint16_t width, uint16_t height, uint16_t srcWidth, uint8_t alpha, bool hasTransparentPixels, uint16_t dstWidth, Bitmap::BitmapFormat srcFormat, Bitmap::BitmapFormat dstFormat)

  Blits a 2D source-array to the frame buffer performing alpha-blending.
- 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, uint16_t dstWidth, Bitmap::BitmapFormat srcFormat, Bitmap::BitmapFormat dstFormat)

  Blits a 2D source-array to the frame buffer performing alpha-blending.
- 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, uint16_t dstWidth, Bitmap::BitmapFormat dstFormat)

  Blits a color value to the frame buffer performing alpha-blending (and transparency keying) as specified.
- 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 rutine when the Back Porch Exit is reached.
- virtual void configureInterrupts ()=0

  Configures the interrupts relevant for TouchGFX.
- virtual void enableInterrupts ()=0

  Enables the DMA and LCD interrupts.
- virtual void disableInterrupts ()=0

  Disables the DMA and LCD interrupts.
- virtual void enableLCDControllerInterrupt ()=0

  Configure the LCD controller to fire interrupts at VSYNC.
- virtual bool sampleKey (uint8_t &key)

  Sample external key event.
- void setDragThreshold (uint8_t value)

  Configure the threshold for reporting drag events.
- virtual void setFrameBufferStartAddress (void ∗adr, uint16_t depth=16, bool useDoubleBuffering=true, bool useAnimationStorage=true)

  Sets the address used for frame buffers, usually located in external memory.
- virtual void setFrameBufferStartAddresses (void ∗frameBuffer, void ∗doubleBuffer, void ∗animationStorage)

  Sets frame buffer start addresses.
- virtual uint16_t configurePartialFrameBuffer (const uint16_t x, const uint16_t y, const uint16_t width, const uint16_t height)

  Configures a partial framebuffer as current framebuffer.
- 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 setFrameBufferAllocator (FrameBufferAllocator ∗allocator)

  Sets a framebuffer allocator.
- FrameBufferAllocator ∗ getFrameBufferAllocator ()

  Gets the framebuffer allocator.
- 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.

- virtual DMAType getDMAType ()

  Function for obtaining the DMA type of the concrete DMA implementation.

- virtual void drawDrawableInDynamicBitmap (Drawable &drawable, BitmapId bitmapId)

  Render a Drawable and its widgets into a dynamic bitmap.

- virtual void drawDrawableInDynamicBitmap (Drawable &drawable, BitmapId bitmapId, const Rect &rect)

  Render a Drawable and its widgets into a dynamic bitmap.

- void setAuxiliaryLCD (LCD ∗auxLCD)

  Set an auxiliary LCD class to be used for offscreen rendering.

- LCD ∗ getAuxiliaryLCD ()

  Get the auxiliary LCD class attached to the HAL instance if any.

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.
- FrameBufferAllocator * frameBufferAllocator

  A reference to an optional FrameBufferAllocator.
- 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.
- LCD * auxiliaryLCD

  Auxiliary LCD class used to render Drawables into dynamic bitmaps.
- Rect partialFrameBufferRect

  The region of the screen covered by the partial framebuffer.

Static Protected Attributes

- static bool isDrawing

  True if currently in the process of rendering a screen.

7.92.1 Detailed Description

Contains functions that are specific to the hardware platform the code is running on.

7.92.2 Member Enumeration Documentation

7.92.2.1 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_BUFFE↩ R_TFT_CTRL	Strategy optimized for single frame buffer on systems with TFT controller.
REFRESH_STRATEGY_PARTIAL_FRAMEBUFFER	Strategy using less than a full frame buffer.

7.92.3 Constructor & Destructor Documentation

7.92.3.1 HAL()

```
HAL (
            DMA_Interface & dmaInterface,
            LCD & display,
            TouchController & touchCtrl,
            uint16_t width,
            uint16_t height )  [inline]
```

Creates a HAL instance.

Parameters

in	*dmaInterface*	Reference to the DMA interface.
in	*display*	Reference to the LCD.
in	*touchCtrl*	Reference to the touch controller.
	width	The width of the LCD display, in pixels.
	height	The height of the LCD display, in pixels.

7.92.3.2 ~HAL()

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

Destructor.

7.92.4 Member Function Documentation

7.92.4.1 allowDMATransfers()

```
void allowDMATransfers ( )  [virtual]
```

Allow the DMA to start transfers. Front Porch Entry is a good place to call this.

7.92.4.2 backPorchExited()

```
void backPorchExited ( )  [inline], [virtual]
```

Has to be called from within the LCD IRQ rutine when the Back Porch Exit is reached.

7.92.4.3 beginFrame()

```
bool beginFrame ( )  [protected], [virtual]
```

Called when beginning to rendering a frame.

Returns

true if rendering can begin, false otherwise.

7.92.4.4 blitCopy() [1/3]

```
void blitCopy (
            const uint16_t * pSrc,
            const uint8_t * pClut,
            uint16_t x,
            uint16_t y,
            uint16_t width,
            uint16_t height,
            uint16_t srcWidth,
            uint8_t alpha,
```

```
            bool hasTransparentPixels,
            uint16_t dstWidth,
            Bitmap::BitmapFormat srcFormat,
            Bitmap::BitmapFormat dstFormat )
```

Blits a 2D source-array to the frame buffer performing alpha-blending (and transparency keying) as specified.

Note

Alpha=255 is assumed "solid" and shall be used if HAL does not support BLIT_OP_COPY_WITH_ALPHA.

Parameters

pSrc	The source-array pointer (points to first value to copy)
pClut	The CLUT pointer (points to CLUT header data which include the type and size of this CLUT followed by colors data)
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
srcWidth	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.
dstWidth	The distance (in elements) from first value of first line, to first value of second line (the destination 2D array width)
srcFormat	The source buffer color format (default is the framebuffer format)
dstFormat	The destination buffer color format (default is the framebuffer format)

7.92.4.5 blitCopy() [2/3]

```
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,
            uint16_t dstWidth,
            Bitmap::BitmapFormat srcFormat,
            Bitmap::BitmapFormat dstFormat )  [virtual]
```

Blits a 2D source-array to the frame buffer performing alpha-blending (and transparency keying) as specified.

Note

Alpha=255 is assumed "solid" and shall be used if HAL does not support BLIT_OP_COPY_WITH_ALPHA.

Parameters

pSrc	The source-array pointer (points to first value to copy)

Parameters

x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
srcWidth	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.
dstWidth	The distance (in elements) from first value of first line, to first value of second line (the destination 2D array width)
srcFormat	The source buffer color format (default is the framebuffer format)
dstFormat	The destination buffer color format (default is the framebuffer format)

7.92.4.6 blitCopy() [3/3]

```
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 using the default lcd format.

Note

Alpha=255 is assumed "solid" and shall be used if HAL does not support BLIT_OP_COPY_WITH_ALPHA.

Parameters

pSrc	The source-array pointer (points to first value to copy)
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
srcWidth	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

7.92.4.7 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

pSrc	The source-array pointer (points to first value to copy)
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
srcWidth	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
alpha	The alpha value to use for blending. This is applied on every pixel, in addition to the per-pixel alpha value (255 = solid, no blending)

7.92.4.8 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

pSrc	The source-array pointer (points to first value to copy)
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
srcWidth	The distance (in elements) from first value of first line, to first value of second line (the source 2D array width)
color	Color of the text.
alpha	The alpha value to use for blending (255 = solid, no blending)
operation	The operation type to use for blit copy.

7.92.4.9 blitFill() [1/2]

```
void blitFill (
            colortype color,
            uint16_t x,
            uint16_t y,
            uint16_t width,
            uint16_t height,
            uint8_t alpha,
            uint16_t dstWidth,
            Bitmap::BitmapFormat dstFormat )  [virtual]
```

Blits a color value to the frame buffer performing alpha-blending (and transparency keying) as specified.

Note

Alpha=255 is assumed "solid" and shall be used if HAL does not support BLIT_OP_FILL_WITH_ALPHA.

Parameters

color	The desired fill-color.
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.
height	The height of desired area of the source 2D array.
alpha	The alpha value to use for blending (255 = solid, no blending)
dstWidth	The distance (in elements) from first value of first line, to first value of second line (the destination 2D array width)
dstFormat	The destination buffer color format (default is the framebuffer format)

7.92.4.10 blitFill() [2/2]

```
void blitFill (
            colortype color,
            uint16_t x,
            uint16_t y,
            uint16_t width,
            uint16_t height,
            uint8_t alpha )  [virtual]
```

Blits a color value to the frame buffer performing alpha-blending (and transparency keying) as specified.

Note

Alpha=255 is assumed "solid" and shall be used if HAL does not support BLIT_OP_FILL_WITH_ALPHA.

Parameters

color	The desired fill-color.
x	The destination x coordinate on the frame buffer.
y	The destination y coordinate on the frame buffer.
width	The width desired area of the source 2D array.

Parameters

height	The height of desired area of the source 2D array.
alpha	The alpha value to use for blending (255 = solid, no blending)

7.92.4.11 blitSetTransparencyKey()

```
void blitSetTransparencyKey (
            uint16_t key )  [virtual]
```

Only present for backwards compatibility in TouchGFX 4.x. Will be removed in TouchGFX 5.

Parameters

key	The "transparent" color value.

Reimplemented in HALSDL2.

7.92.4.12 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	*dest*	Pointer to destination memory.
in	*src*	Pointer to source memory.
	numBytes	Number of bytes to copy.

Returns

true if the copy succeeded, false if copy was not performed.

Reimplemented in HALSDL2.

7.92.4.13 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

str	A pointer to the string which may be in external memory.

Returns

A pointer to an identical string which is guaranteed to be directly readable (ie. a copy if the original string was placed in NAND flash).

7.92.4.14 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.

7.92.4.15 configurePartialFrameBuffer()

```
uint16_t configurePartialFrameBuffer (
            const uint16_t x,
            const uint16_t y,
            const uint16_t width,
            const uint16_t height )  [virtual]
```

Configures a partial framebuffer as current framebuffer. This method uses the assigned FrameBufferAllocator to allocate block of compatible dimensions. The height of the allocated block is returned.

Parameters

x	The absolute x coordinate of the block on the screen.
y	The absolute y coordinate of the block on the screen.
width	The width of the block.
height	The height of the block requested.

Returns

The height of the block allocated.

7.92.4.16 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

meAbs	The frame buffer region to copy.

Returns

A pointer to the memory address containing the copy of the frame buffer.

7.92.4.17 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

meAbs	The frame buffer region to copy.
dst	Address of the buffer to store the copy in.
stride	The width of the target buffer (row length).

Returns

A pointer to the memory address containing the copy of the frame buffer.

7.92.4.18 disableInterrupts()

```
void disableInterrupts ( )  [pure virtual]
```

Disables the DMA and LCD interrupts.

Implemented in HALSDL2.

7.92.4.19 drawDrawableInDynamicBitmap() [1/2]

```
void drawDrawableInDynamicBitmap (
            Drawable & drawable,
            BitmapId bitmapId )  [virtual]
```

Render a Drawable and its widgets into a dynamic bitmap.

Parameters

drawable	A reference on the Drawable object to render.
bitmapId	Dynamic bitmap to be used as a rendertarget.

7.92.4.20 drawDrawableInDynamicBitmap() [2/2]

```
void drawDrawableInDynamicBitmap (
            Drawable & drawable,
            BitmapId bitmapId,
            const Rect & rect )  [virtual]
```

Render a Drawable and its widgets into a dynamic bitmap. Only the specified Rect region is updated.

Parameters

drawable	A reference on the Drawable object to render.
bitmapId	Dynamic bitmap to be used as a rendertarget.
rect	Region to update.

7.92.4.21 enableInterrupts()

```
void enableInterrupts ( )  [pure virtual]
```

Enables the DMA and LCD interrupts.

Implemented in HALSDL2.

7.92.4.22 enableLCDControllerInterrupt()

```
void enableLCDControllerInterrupt ( )  [pure virtual]
```

Configure the LCD controller to fire interrupts at VSYNC. Called automatically once TouchGFX initialization has completed.

Implemented in HALSDL2.

7.92.4.23 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

enabled	If true, set flag to update MCU load.

7.92.4.24 endFrame()

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

Called when a rendering pass is completed.

7.92.4.25 flushDMA()

```
void flushDMA ( )  [virtual]
```

This function blocks until the DMA queue (containing BlitOps) is empty.

7.92.4.26 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.

7.92.4.27 flushFrameBuffer() [2/2]

```
void flushFrameBuffer (
            const Rect & rect )  [virtual]
```

This function is called whenever the framework has performed a partial draw.

Parameters

rect	The area of the screen that has been drawn, expressed in absolute coordinates.

See also

flushFrameBuffer().

Reimplemented in HALSDL2.

7.92.4.28 frontPorchEntered()

```
void frontPorchEntered ( )  [inline]
```

Has to be called from within the LCD IRQ routine when the Front Porch Entry is reached.

7.92.4.29 getAnimationStorage()

```
uint16_t * getAnimationStorage ( ) const  [inline]
```

Gets the optional frame buffer used for animation storage.

Returns

The address or 0 if unused.

7.92.4.30 getAuxiliaryLCD()

```
void getAuxiliaryLCD ( )  [inline]
```

Get the auxiliary LCD class attached to the HAL instance if any.

Returns

A pointer on the axiliary LCD class attached to the HAL instance.

7.92.4.31 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.

7.92.4.32 getButtonController()

```
ButtonController * getButtonController ( ) const  [inline]
```

Gets the associated ButtonController.

Returns

A pointer to the ButtonController, or zero if no ButtonController has been set.

7.92.4.33 getClientFrameBuffer()

```
uint16_t * getClientFrameBuffer ( )  [inline], [protected]
```

Gets client frame buffer.

Returns

The address of the framebuffer currently used by the framework to draw in.

7.92.4.34 getCPUCycles()

```
uint32_t getCPUCycles ( )
```

Gets the current cycle counter.

Returns

the cycle counter.

7.92.4.35 getDisplayHeight()

```
uint16_t getDisplayHeight ( ) const  [inline]
```

Gets display height.

Returns

The display height.

7.92.4.36 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.

7.92.4.37 getDisplayWidth()

```
uint16_t getDisplayWidth ( ) const  [inline]
```

Gets display width.

Returns

The display width.

7.92.4.38 getDMAType()

```
DMAType getDMAType ( )  [inline], [virtual]
```

Function for obtaining the DMA type of the concrete DMA implementation. As default, will return DMA_TYPE_G↩ ENERIC type value.

Returns

a DMAType value of the concrete DMA implementation.

7.92.4.39 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.

7.92.4.40 getFrameBufferAllocator()

```
FrameBufferAllocator * getFrameBufferAllocator ( )  [inline]
```

Gets the framebuffer allocator.

Returns

The framebuffer allocator

7.92.4.41 getFrameRefreshStrategy()

```
FrameRefreshStrategy getFrameRefreshStrategy ( ) const  [inline]
```

Returns

Current frame refresh strategy.

See also

bool setFrameRefreshStrategy(FrameRefreshStrategy s)

7.92.4.42 getInstance()

```
static HAL * getInstance ( )  [inline], [static]
```

Gets the HAL instance.

Returns

The HAL instance.

7.92.4.43 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.

7.92.4.44 getMCULoadPct()

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

Gets the current MCU load.

Returns

mcuLoadPct the MCU Load in %.

7.92.4.45 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 <= value < 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 getTF↩ TCurrentLine function is never called and can be disregarded.

Returns

In this default implementation, 0xFFFF is returned to signify "not implemented".

7.92.4.46 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.

7.92.4.47 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.

7.92.4.48 initialize()

```
void initialize ( )
```

This function is responsible for initializing the entire framework.

7.92.4.49 lcd()

```
static LCD & lcd ( )  [inline], [static]
```

Gets a reference to the LCD.

Returns

A reference to the LCD.

7.92.4.50 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

enableLock	True to lock DMA transfers to the front porch signal. Conservative, default setting. False to disable, which will normally yield substantial performance improvement.

7.92.4.51 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.

7.92.4.52 noTouch()

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

Called by the touch driver to indicate that no touch is currently detected.

7.92.4.53 performDisplayOrientationChange()

```
void performDisplayOrientationChange ( ) [inline], [protected], [virtual]
```

Perform the actual display orientation change.

Reimplemented in HALSDL2.

7.92.4.54 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	*listener*	The listener to register.

7.92.4.55 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	*delayF*	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.

7.92.4.56 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	*str*	Pointer to buffer location.
	length	Buffer length (in UnicodeChar's)

See also

cacheTextString

7.92.4.57 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.

7.92.4.58 setAuxiliaryLCD()

```
void setAuxiliaryLCD (
            LCD * auxLCD )  [inline]
```

Set an auxiliary LCD class to be used for offscreen rendering.

Parameters

auxLCD	A pointer on the axiliary LCD class to use for offscreen rendering.

7.92.4.59 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.

7.92.4.60 setDisplayOrientation()

```
void setDisplayOrientation (
            DisplayOrientation orientation )  [inline], [virtual]
```

Sets the desired display orientation (landscape or portrait). If desired orientation is different from the native orienta-

tion 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.

7.92.4.61 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

value	New threshold value.

7.92.4.62 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∗(fingersize-1)+1. This can be especially useful when trying to interact with small elements on a high ppi display. The finger size will influence which element is chosen as the point of interaction, when clicking, dragging, ... the display. A number of samples will be drawn from within the finger area and a best matching drawable will be chosen. The best matching algorithm will consider the size of the drawable and the distance from the touch point.

Parameters

in	*size*	the size of the finger.

7.92.4.63 setFrameBufferAllocator()

```
void setFrameBufferAllocator (
            FrameBufferAllocator * allocator )  [inline]
```

Sets a framebuffer allocator. The framebuffer allocator is only used in partial framebuffer mode.

Parameters

in	*allocator*	pointer to a framebuffer allocator object

7.92.4.64 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	*adr*	Starting address to use for frame buffers.
	depth	(Optional) Depth of each pixel in bits, default is 16.
	useDoubleBuffering	(Optional) If true, reserve memory for an extra frame buffer.
	useAnimationStorage	(Optional) If true, reserve memory for animation storage.

7.92.4.65 setFrameBufferStartAddresses()

```
void setFrameBufferStartAddresses (
            void * frameBuffer,
            void * doubleBuffer,
            void * animationStorage )  [inline], [virtual]
```

Sets individual frame buffer start addresses.

Parameters

in	*frameBuffer*	Buffer for frame buffer data, must be non-null.
in	*doubleBuffer*	If non-null, buffer for double buffer data. If null double buffering is disabled.
in	*animationStorage*	If non-null, the animation storage. If null animation storage is disabled.

7.92.4.66 setFrameRateCompensation()

```
void setFrameRateCompensation (
            bool enabled )  [inline]
```

Enables or disables compensation for lost frames. See knowledge base article.

Parameters

enabled	true to enable, false to disable.

7.92.4.67 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

s	The desired strategy to use.

Returns

true if the desired strategy will be used, false otherwise.

7.92.4.68 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

active	If true, MCU is registered as being active, inactive otherwise.

7.92.4.69 setMCUInstrumentation()

```
void setMCUInstrumentation (
            MCUInstrumentation * mcuInstr )  [inline]
```

Stores a pointer to an instance of an MCU specific instrumentation class.

Parameters

in	*mcuInstr*	pointer to MCU instrumentation.

7.92.4.70 setTFTFrameBuffer()

```
void setTFTFrameBuffer (
            uint16_t * address )  [protected], [pure virtual]
```

Sets the frame buffer address used by the TFT controller.

Parameters

in	*address*	New frame buffer address.

Implemented in HALSDL2.

7.92.4.71 setTouchSampleRate()

```
void setTouchSampleRate (
            int8_t sampleRateInTicks )  [inline]
```

Sets the number of ticks between each touch screen sample.

Parameters

sampleRateInTicks	Sample rate. Default is 1 (every tick).

7.92.4.72 signalDMAInterrupt()

```
void signalDMAInterrupt ( )  [inline]
```

Notify the framework that a DMA interrupt has occurred.

7.92.4.73 swapFrameBuffers()

```
void swapFrameBuffers ( )
```

Swaps the two frame buffers.

7.92.4.74 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

ms	Number of milliseconds to wait.

See also

void registerTaskDelayFunction(void (∗delayF)(uint16_t))

7.92.4.75 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.

7.92.4.76 tick()

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

This function is called at each timer tick, depending on platform implementation.

7.92.4.77 touch()

```
void touch (
             int32_t x,
             int32_t y )  [protected], [virtual]
```

Called by the touch driver to indicate a touch.

Parameters

x	The x coordinate of the touch.
y	The y coordinate of the touch.

7.92.4.78 unlockFrameBuffer()

```
void unlockFrameBuffer ( )  [virtual]
```

Unlocks the frame buffer (MUST be called exactly once for each call to lockFrameBuffer()).

7.92.4.79 vSync()

```
void vSync ( )  [inline]
```

Called by the VSync interrupt for counting of LCD refreshes.

7.93 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.
- static uint8_t * scaleTo24bpp (uint8_t *dst, uint16_t *src, Bitmap::BitmapFormat format)

  Scale framebuffer to 24bpp.
- static uint8_t * doRotate (uint8_t *dst, uint8_t *src)

  Rotates a framebuffer if the display is rotated.

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

7.93.1 Detailed Description

HAL implementation for the TouchGFX simulator.

See also

HAL

7.93.2 Constructor & Destructor Documentation

7.93.2.1 HALSDL2()

```
HALSDL2 (
            DMA_Interface & dma,
            LCD & lcd,
            TouchController & touchCtrl,
            uint16_t width,
            uint16_t height )  [inline]
```

Constructor. Initializes members.

Parameters

in	*dma*	Reference to DMA interface.
in	*lcd*	Reference to the LCD.
in	*touchCtrl*	Reference to Touch Controller driver.
	width	Width of the display.
	height	Height of the display.

7.93.3 Member Function Documentation

7.93.3.1 blitSetTransparencyKey()

```
void blitSetTransparencyKey (
            uint16_t key )  [virtual]
```

If Blit-operations are supported, transparency-keying support is implicitly assumed.

Parameters

key	The "transparent" color value.

Reimplemented from HAL.

7.93.3.2 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	*dest*	Pointer to destination memory.
	src	Pointer to source memory.
	numBytes	Number of bytes to copy.

Returns

true if the copy succeeded, false if copy was not performed.

Reimplemented from HAL.

7.93.3.3 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.

7.93.3.4 configureLCDInterrupt()

```
void configureLCDInterrupt ( )  [inline], [protected], [virtual]
```

Configures LCD interrupt.

7.93.3.5 copyScreenshotToClipboard()

```
void copyScreenshotToClipboard ( )  [virtual]
```

Copies the screenshot to clipboard.

7.93.3.6 disableInterrupts()

```
void disableInterrupts ( )  [inline], [protected], [virtual]
```

Disables the DMA and LCD interrupts.

Implements HAL.

7.93.3.7 doRotate()

```
uint8_t * doRotate (
            uint8_t * dst,
            uint8_t * src )  [static]
```

Rotates a framebuffer if the display is rotated.

Parameters

out	*dst*	Destination for the rotated framebuffer. must be non-null if the screen is rotated.
in	*src*	The framebuffer.

Returns

Null if it fails, else a pointer to an uint8_t.

7.93.3.8 doSampleTouch()

```
bool doSampleTouch (
            int32_t & x,
            int32_t & y ) const
```

Samples the position of the mouse cursor.

Parameters

out	*x*	The x coordinate.
out	*y*	The y coordinate.

Returns

True if touch detected, false otherwise.

7.93.3.9 enableInterrupts()

```
void enableInterrupts ( )  [inline], [protected], [virtual]
```

Enables the DMA and LCD interrupts.

Implements HAL.

7.93.3.10 enableLCDControllerInterrupt()

```
void enableLCDControllerInterrupt ( )  [inline], [protected], [virtual]
```

Enables the LCD interrupt.

Implements HAL.

7.93.3.11 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.

7.93.3.12 flushFrameBuffer() [2/2]

```
void flushFrameBuffer (
            const Rect & rect )  [virtual]
```

This function is called whenever the framework has performed a partial draw.

Parameters

rect	The area of the screen that has been drawn, expressed in absolute coordinates.

See also

flushFrameBuffer(). This function is called whenever the framework has performed a partial draw.

Reimplemented from HAL.

7.93.3.13 getArgv()

```
static char ** getArgv (
            int * argc )  [static]
```

Gets the argc and argv for a Windows program.

Parameters

argc	Pointer to where to store number of arguments

Returns

The argv list of arguments.

7.93.3.14 getTFTFrameBuffer()

```
uint16_t * getTFTFrameBuffer ( ) const  [protected], [virtual]
```

Gets TFT frame buffer.

Returns

null if it fails, else the TFT frame buffer.

Implements HAL.

7.93.3.15 getWindowTitle()

```
const char * getWindowTitle ( ) const
```

Gets window title.

Returns

null "TouchGFX simulator" unless set to something else using setWindowTitle().

See also

setWindowTitle

7.93.3.16 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

orientation	The orientation.
x	The x coordinate.
y	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 as skin is set, the entire framebuffer is rendered through SDL whenever there is a change. Without a skin, only the areas with changes is rendered through SDL.

7.93.3.17 performDisplayOrientationChange()

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

Perform the actual display orientation change.

Reimplemented from HAL.

7.93.3.18 renderLCD_FrameBufferToMemory()

```
void renderLCD_FrameBufferToMemory (
            const Rect & _rectToUpdate,
            uint8_t * frameBuffer )  [protected], [virtual]
```

Update frame buffer using an SDL Surface.

Parameters

	_rectToUpdate	Area to update.
in	*frameBuffer*	Target frame buffer.

7.93.3.19 sampleKey()

```
bool sampleKey (
            uint8_t & key )  [virtual]
```

Sample key event from keyboard.

Parameters

out	*key*	Output parameter that will be set to the key value if a key press was detected.

Returns

True if a key press was detected and the "key" parameter is set to a value.

Reimplemented from HAL.

7.93.3.20 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

n	Number of screenshots to save. These are added to any ongoing amount of screenshots in queue.

7.93.3.21 saveScreenshot() [1/2]

```
void saveScreenshot ( )
```

Saves a screenshot to the default folder and default filename.

7.93.3.22 saveScreenshot() [2/2]

```
void saveScreenshot (
            char * folder,
            char * filename )  [virtual]
```

Saves a screenshot.

Parameters

in	*folder*	Folder name to place the screenshot in.
in	*filename*	Filename to save the screenshot to.

7.93.3.23 scaleTo24bpp()

```
uint8_t * scaleTo24bpp (
            uint8_t * dst,
            uint16_t * src,
            Bitmap::BitmapFormat format )  [static]
```

Scale framebuffer to 24bpp. The format of the framebuffer (src) is given in parameter format. The result is placed in the pre-allocated memory pointed to by parameter dst. If the frambebuffer is in format Bitmap::RGB888, parameter dst is not used and the parameter src is simply returned.

Parameters

out	*dst*	Destination for the framebuffer. must be non-null unless format is Bitmap::RGB888.
in	*src*	The framebuffer.
	format	Describes the format of the framebuffer (lcd().framebufferFormat()).

Returns

Null if it fails, else a pointer to an uint8_t.

7.93.3.24 sdl_init()

```
bool sdl_init (
```

```
            int argcount,
            char ** args )  [virtual]
```

Initializes SDL.

Parameters

	argcount	Number of arguments.
in	*args*	Arguments.

Returns

True if init went well, false otherwise.

7.93.3.25 setTFTFrameBuffer()

```
void setTFTFrameBuffer (
            uint16_t * addr )  [protected], [virtual]
```

Sets TFT frame buffer.

Parameters

in	*addr*	The address of the TFT frame buffer.

Implements HAL.

7.93.3.26 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

ms	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 $*$ 16.6667ms = 4250ms = 4.25s 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.

7.93.3.27 setWindowTitle()

```
void setWindowTitle (
            const char * title )
```

Sets window title of the TouchGFX simulator.

Parameters

title	The title, if null the original "TouchGFX simulator" will be used.

See also

getWindowTitle

7.93.3.28 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.

7.94 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.

7.94.1 Detailed Description

Platform independent interface for I2C drivers.

7.94.2 Constructor & Destructor Documentation

7.94.2.1 I2C()

```
I2C (
            uint8_t ch )  [inline]
```

Stores the channel of the I2C bus to be configured.

Parameters

ch	I2C channel.

7.94.2.2 ~I2C()

```
~I2C ( )  [inline], [virtual]
```

Destructor.

7.94.3 Member Function Documentation

7.94.3.1 init()

```
void init ( )  [pure virtual]
```

Initializes the I2C driver.

7.94.3.2 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

	addr	The I2C device address.
	reg	The register.
out	*data*	Pointer to buffer in which to place the result.
	cnt	Size of buffer in bytes.

Returns

true on success, false otherwise.

7.94.3.3 writeRegister()

```
bool writeRegister (
            uint8_t addr,
            uint8_t reg,
            uint8_t val )  [pure virtual]
```

Writes the specified value in a register.

Parameters

addr	The I2C device address.
reg	The register.
val	The new value.

Returns

true on success, false otherwise.

7.95 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.

7.95.1 Detailed Description

Specific I2C-enabled type of Touch Controller.

See also

TouchController

7.95.2 Constructor & Destructor Documentation

7.95.2.1 I2CTouchController()

```
I2CTouchController (
            I2C & i2c )  [inline]
```

Constructor. Initializes I2C driver.

Parameters

in,out	*i2c*	I2C driver.

7.95.2.2 ~I2CTouchController()

```
~I2CTouchController ( )  [inline], [virtual]
```

Destructor.

7.95.3 Member Function Documentation

7.95.3.1 init()

```
void init ( )  [pure virtual]
```

Initializes touch controller.

Implements TouchController.

7.95.3.2 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	*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.

7.96 IconButtonStyle< T > 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.

7.96.1 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: IconButton↩ Style<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

T	Generic type parameter. Typically a AbstractButtonContainer subclass.

See also

AbstractButtonContainer

7.96.2 Member Function Documentation

7.96.2.1 getCurrentlyDisplayedIcon()

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

Returns

The currently displayed icon.

7.96.2.2 getIconX()

```
int16_t getIconX ( ) const  [inline]
```

Returns

The icon x coordinate.

7.96.2.3 getIconY()

```
int16_t getIconY ( ) const  [inline]
```

Returns

The icon y coordinate.

7.96.2.4 setIconBitmaps()

```
void setIconBitmaps (
            const Bitmap & newIconReleased,
            const Bitmap & newIconPressed )  [inline], [virtual]
```

Parameters

newIconReleased	The new icon released.
newIconPressed	The new icon pressed.

7.96.2.5 setIconX()

```
void setIconX (
            int16_t x )  [inline]
```

Parameters

x	The x coordinate.

7.96.2.6 setIconXY()

```
void setIconXY (
            int16_t x,
            int16_t y )  [inline]
```

Parameters

x	The x coordinate.
y	The y coordinate.

7.96.2.7 setIconY()

```
void setIconY (
            int16_t y )  [inline]
```

Parameters

y	The y coordinate.

7.97 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

7.97.1 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

7.97.2 Constructor & Destructor Documentation

7.97.2.1 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

bmp	The bitmap to display.

7.97.3 Member Function Documentation

7.97.3.1 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

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Implements Drawable.

Reimplemented in TiledImage.

7.97.3.2 getAlpha()

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

Gets the current alpha value.

Returns

The current alpha value.

7.97.3.3 getBitmap()

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

Gets the BitmapId currently contained by the widget.

Returns

The current BitmapId of the widget.

7.97.3.4 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.

7.97.3.5 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.

7.97.3.6 setAlpha()

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

Sets the alpha channel for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

7.97.3.7 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

bmp	The bitmap instance.

See also

Bitmap

Reimplemented in AnimatedImage, and TiledImage.

7.98 ImageButtonStyle< T > 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.

7.98.1 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

T	Generic type parameter. Typically a AbstractButtonContainer subclass.

See also

AbstractButtonContainer

7.98.2 Member Function Documentation

7.98.2.1 getCurrentlyDisplayedBitmap()

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

Returns

The currently displayed bitmap.

7.98.2.2 setBitmaps()

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

Parameters

bmpReleased	The bitmap released.
bmpPressed	The bitmap pressed.

7.98.2.3 setBitmapXY()

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

Parameters

x	An uint16_t to process.
y	An uint16_t to process.

7.99 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

7.99.1 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.
```

7.99.2 Constructor & Destructor Documentation

7.99.2.1 ImageProgress()

```
ImageProgress ( )
```

Default constructor.

7.99.2.2 ~ImageProgress()

```
~ImageProgress ( )  [virtual]
```

Destructor.

7.99.3 Member Function Documentation

7.99.3.1 getAlpha()

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

Returns

The the alpha of the image.

See also

setAlpha
Image::getAlpha

7.99.3.2 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

7.99.3.3 getBitmap()

```
touchgfx::BitmapId getBitmap ( ) const  [virtual]
```

Gets the image.

Returns

The image.

See also

setBitmap

7.99.3.4 setAlpha()

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

Sets the alpha of the image.

Parameters

alpha	The alpha.

See also

getAlpha
Image::setAlpha

7.99.3.5 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

anchorAtZero	true to anchor at zero, false to anchor at current progress.

See also

getAnchoredAtZero

7.99.3.6 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

bitmapId	The bitmap id.

See also

getBitmap
TiledImage

7.99.3.7 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

x	The x coordinate.
y	The y coordinate.
width	The width of the image progress indicator.
height	The height of the image progress indicator.

Reimplemented from AbstractProgressIndicator.

7.99.3.8 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

value	The value.

Reimplemented from AbstractProgressIndicator.

7.100 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 bits↩ PerPixel, uint8_t dataFormatA4, 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

7.100.1 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

7.100.2 Constructor & Destructor Documentation

7.100.2.1 InternalFlashFont()

```
InternalFlashFont (
            const GlyphNode * list,
            uint16_t size,
            uint16_t height,
            uint8_t pixBelowBase,
            uint8_t bitsPerPixel,
            uint8_t dataFormatA4,
            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

list	The array of glyphs known to this font.
size	The number of glyphs in list.
height	The height in pixels of the highest character in this font.
pixBelowBase	The maximum number of pixels that can be drawn below the baseline in this font.
bitsPerPixel	The number of bits per pixel in this font.
dataFormatA4	The glyphs are saved using ST A4 format.
maxLeft	The maximum a character extends to the left.
maxRight	The maximum a character extends to the right.
glyphDataInternalFlash	Pointer to the glyph data for the font, placed in internal flash.
kerningList	pointer to the kerning data for the font, placed in internal flash.
fallbackChar	The fallback character for the typography in case no glyph is available.
ellipsisChar	The ellipsis character used for truncating long texts.

7.100.3 Member Function Documentation

7.100.3.1 getKerning()

```
int8_t getKerning (
            Unicode::UnicodeChar prevChar,
            const GlyphNode * glyph ) const  [virtual]
```

Gets the kerning distance between two characters.

Parameters

prevChar	The unicode value of the previous character.
glyph	the glyph object for the current character.

Returns

The kerning distance between prevChar and glyph char.

Implements ConstFont.

7.100.3.2 getPixelData()

```
const uint8_t * getPixelData (
            const GlyphNode * glyph ) const  [virtual]
```

Obtains a RAM-based pointer to the pixel data for the specified glyph.

Parameters

glyph	The glyph to get the pixels data of.

Returns

The pixel data of the glyph.

Implements ConstFont.

7.101 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.

7.101.1 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.

7.101.2 Constructor & Destructor Documentation

7.101.2.1 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

scanline	The scanline to iterate.

7.101.3 Member Function Documentation

7.101.3.1 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.

7.101.3.2 getNumPix()

```
int getNumPix ( ) const  [inline]
```

Gets number of consecutive pixels in the current run on the Scanline.

Returns

The number of consecutive pixels.

7.101.3.3 next()

```
int next ( )  [inline]
```

Gets the next element on the Scanline.

Returns

An the next index in the array of Scanline elements.

7.102 JSMOCHelper Class Reference

Helper class providing caching of certain information while the JSMOC algorithm runs during draw operations.

`#include <touchgfx/JSMOCHelper.hpp>`

Public Member Functions

- JSMOCHelper ()

  Default constructor.
- void setWidget (Drawable *newWidget)

  Sets a widget.
- Drawable * getWidget ()

  Gets the widget.
- Rect & getCachedVisibleRect ()

  Gets the visible rect for the widget of this helper.
- int16_t getCachedAbsX ()

  Gets the absolute x coordinate for the widget of this helper.
- int16_t getCachedAbsY ()

  Gets the absolute y coordinate for the widget of this helper.
- int16_t getWidth ()

  Gets the width of the widget of this helper.
- int16_t getHeight ()

  Gets the height of the widget of this helper.
- void draw (const Rect &invalidatedArea)

  Draws the widget of this helper.

7.102.1 Detailed Description

Helper class providing caching of certain information while the JSMOC algorithm runs during draw operations. Not intented for application-level use.

7.102.2 Constructor & Destructor Documentation

7.102.2.1 JSMOCHelper()

```
JSMOCHelper ( )  [inline]
```

Default constructor.

7.102.3 Member Function Documentation

7.102.3.1 draw()

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

Draws the widget of this helper.

Parameters

invalidatedArea	The area of the widget to draw.

7.102.3.2 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.

7.102.3.3 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.

7.102.3.4 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.

7.102.3.5 getHeight()

```
int16_t getHeight ( )  [inline]
```

Gets the height of the widget of this helper.

Returns

The height of the widget of this helper.

7.102.3.6 getWidget()

```
Drawable * getWidget ( )  [inline]
```

Gets the widget.

Returns

The widget this helper operates on.

7.102.3.7 getWidth()

```
int16_t getWidth ( )  [inline]
```

Gets the width of the widget of this helper.

Returns

The width of the widget of this helper.

7.102.3.8 setWidget()

```
void setWidget (
            Drawable * newWidget )  [inline]
```

Sets a widget.

Parameters

in	*newWidget*	The widget to operate on.

7.103 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.

7.103.1 Detailed Description

Structure providing information about a kerning for a given char pair. Used by LCD when rendering.

7.104 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".

7.105 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 cancelIsEmitted

  Tells if a cancel is emitted to check when a key is released.

Additional Inherited Members

7.105.1 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

7.105.2 Constructor & Destructor Documentation

7.105.2.1 Keyboard()

```
Keyboard ( )
```

Creates a new Keyboard.

7.105.2.2 ~Keyboard()

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

Destructor.

7.105.3 Member Function Documentation

7.105.3.1 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

invalidatedArea	The area to draw.

Reimplemented from Container.

7.105.3.2 getBuffer()

```
Unicode::UnicodeChar * getBuffer ( ) const  [inline]
```

Gets the buffer.

Returns

The buffer containing entered text currently being displayed.

7.105.3.3 getBufferPosition()

```
uint16_t getBufferPosition ( )  [inline]
```

Gets buffer position.

Returns

the buffer position, i.e. the current index where new characters will be placed.

7.105.3.4 getCallbackAreaForCoordinates()

```
CallbackArea getCallbackAreaForCoordinates (
            int16_t x,
            int16_t y ) const  [protected]
```

Gets the callback area defined by the layout for the specified coordinates.

Parameters

x	The x coordinate to perform key look up with.
y	The y coordinate to perform key look up with.

Returns

The CallbackArea, which is empty if not found.

7.105.3.5 getCharForKey()

```
Unicode::UnicodeChar getCharForKey (
            uint8_t keyId ) const  [protected]
```

Maps a keyId to the UnicodeChar being displayed by that key.

Parameters

keyId	The id of the key to perform lookup with.

Returns

the UnicodeChar used for the specified key.

7.105.3.6 getKeyForCoordinates()

```
Key getKeyForCoordinates (
            int16_t x,
            int16_t y ) const  [protected]
```

Gets key for coordinates.

Parameters

x	The x coordinate to perform key look up with.
y	The y coordinate to perform key look up with.

Returns

The key for the given coordinates.

7.105.3.7 getKeyMappingList()

```
const KeyMappingList * getKeyMappingList ( ) const  [inline]
```

Gets key mapping list.

Returns

The KeyMappingList used by the Keyboard.

7.105.3.8 getLayout()

```
const Layout * getLayout ( ) const  [inline]
```

Gets the layout.

Returns

The layout used by the Keyboard.

7.105.3.9 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_KEYBOARD.

Reimplemented from Container.

7.105.3.10 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

evt	The ClickEvent.

Reimplemented from Drawable.

7.105.3.11 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

evt	The DragEvent.

Reimplemented from Drawable.

7.105.3.12 setBuffer()

```
void setBuffer (
            Unicode::UnicodeChar * newBuffer,
            uint16_t newBufferSize )
```

Sets the buffer to be used by the keyboard.

Parameters

in	*newBuffer*	Pointer to a buffer holding the text edited by the keyboard. If the buffer is not empty, the edit position for the keyboard will be set to the end of the provided text.
	newBufferSize	Length of the buffer, i.e. number of UnicodeChar's.

7.105.3.13 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

newPos	The buffer position.

7.105.3.14 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 Callback to invoke.

7.105.3.15 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

newKeyMappingList	The new KeyMappingList.

7.105.3.16 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

newLayout	The new layout.

7.105.3.17 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.

7.105.3.18 setupDrawChain()

```
void setupDrawChain (
            const Rect & invalidatedArea,
            Drawable ** nextPreviousElement )  [protected], [virtual]
```

Note

For TouchGFX internal use only.

Parameters

	invalidatedArea	Include drawables that intersect with this area only.
in,out	*nextPreviousElement*	Modifiable element in linked list.

Reimplemented from Container.

7.106 Keyboard::KeyMapping Struct Reference

Mapping from key id to Unicode character.

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

Public Attributes

- uint8_t keyId

  Id of a key.
- Unicode::UnicodeChar keyValue

  Unicode equivalent of the key id.

7.107 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.

7.108 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.

7.109 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.
- 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.

- uint16_t * copyFrameBufferRegionToMemory (const Rect &region, const BitmapId bitmapId=BITMAP_ANIMATION_STORAGE)

  Copies a part of the frame buffer.
- virtual uint16_t * copyFrameBufferRegionToMemory (const Rect &visRegion, const Rect &absRegion, const BitmapId bitmapId)=0

  Copies part of the frame buffer region to memory.
- virtual void fillRect (const Rect &rect, colortype color, uint8_t alpha=255)=0

  Draws a filled rectangle in the specified color.
- void drawHorizontalLine (int16_t x, int16_t y, uint16_t width, uint16_t lineWidth, colortype color, uint8_t alpha=255)

  Draws a horizontal line with the specified color.
- void drawVerticalLine (int16_t x, int16_t y, uint16_t height, uint16_t lineWidth, colortype color, uint8_t alpha=255)

  Draws a vertical line with the specified color.
- void drawRect (const Rect &rect, colortype color, uint8_t alpha=255)

  Draws a rectangle using the specified line color.
- void drawBorder (const Rect &rect, uint16_t lineWidth, colortype color, uint8_t alpha=255)

  Draws a rectangle width the specified line width and color.
- void drawString (Rect widgetArea, const Rect &invalidatedArea, const 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const =0

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const =0

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const =0

  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.
- virtual uint8_t getRedColor (colortype color) const =0

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const =0

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const =0

  Gets the blue color part of a color.
- 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.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint8_t div255 (uint16_t num)

  Approximates an integer division of a 16bit value by 255.
- static FORCE_INLINE_FUNCTION uint32_t div255rb (uint32_t pixelxAlpha)

  Divides the red and blue components of pixelxAlpha by 255.
- static FORCE_INLINE_FUNCTION uint32_t div255g (uint32_t pixelxAlpha)

  Divides the green component of pixelxAlpha by 255.

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, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)=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.

7.109.1 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!

7.109.2 Constructor & Destructor Documentation

7.109.2.1 ~LCD()

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

Destructor.

7.109.3 Member Function Documentation

7.109.3.1 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 LCD16bppSerialFlash, LCD16bpp, LCD24bpp, LCD32bpp, LCD2bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.2 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 perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD24bpp, LCD32bpp, LCD2bpp, LCD4bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, and LCD1bpp.

7.109.3.3 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

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD24bpp, LCD32bpp, LCD2bpp, LCD4bpp, LCD8bpp_AB↩GR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, and LCD1bpp.

7.109.3.4 copyFrameBufferRegionToMemory() [1/2]

```
uint16_t * copyFrameBufferRegionToMemory (
            const Rect & region,
            const BitmapId bitmapId = BITMAP_ANIMATION_STORAGE )  [inline]
```

Copies a part of the frame buffer to a bitmap. The bitmap must be a dynamic bitmap or animation storage (default). Only the part specified with by parameter region is copied.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

region	The part to copy.
bitmap↩Id	(Optional) The bitmap to store the data in. Default parameter is Animation Storage.

Returns

A pointer to the copy.

See also

blitCopy

7.109.3.5 copyFrameBufferRegionToMemory() [2/2]

```
uint16_t * copyFrameBufferRegionToMemory (
```

```
            const Rect & visRegion,
            const Rect & absRegion,
            const BitmapId bitmapId )  [pure virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANIMATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD24bpp, LCD32bpp, LCD2bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.6 div255()

```
FORCE_INLINE_FUNCTION static uint8_t div255 (
            uint16_t num )  [inline], [static]
```

Divides numerator num (e.g. the sum resulting from an alpha-blending operation) by 255.

Parameters

in	*num*	The numerator to divide by 255.

Returns

The result of a division by 255.

7.109.3.7 div255g()

```
FORCE_INLINE_FUNCTION static uint32_t div255g (
            uint32_t pixelxAlpha )  [inline], [static]
```

Divides the green component of pixelxAlpha by 255.

Parameters

in	*pixelxAlpha*	The green component of a 32bit ARGB pixel multiplied by an alpha factor.

Returns

pixelxAlpha with its green component divided by 255.

7.109.3.8 div255rb()

```
FORCE_INLINE_FUNCTION static uint32_t div255rb (
            uint32_t pixelxAlpha )  [inline], [static]
```

Divides the red and blue components of pixelxAlpha by 255.

Parameters

in	*pixelxAlpha*	The red and blue components of a 32bit ARGB pixel multiplied by an alpha factor.

Returns

pixelxAlpha with its red and blue components divided by 255.

7.109.3.9 drawBorder()

```
void drawBorder (
            const Rect & rect,
            uint16_t lineWidth,
            colortype color,
            uint8_t alpha = 255 )
```

Draws a rectangle width the specified line width and color.

Parameters

rect	The rectangle x, y, width, height in absolute coordinates.
lineWidth	The width of the line.
color	The color to use.
alpha	The alpha value to use (default=solid)

7.109.3.10 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,
            uint8_t dataFormatA4,
            colortype color,
            uint8_t bitsPerPixel,
            uint8_t alpha,
            TextRotation rotation )  [protected], [pure virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public function drawString().

Parameters

out	*wbuf*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw inside.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD2bpp, LCD32bpp, LCD24bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.11 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

x	The x coordinate of the starting point, in absolute coordinates.
y	The y coordinate of the starting point, in absolute coordinates.
width	The length of the line.
lineWidth	The width of the line.
color	The color to use.
alpha	The alpha value to use (default=solid)

7.109.3.12 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

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0,0) on the screen.
y	The absolute y coordinate to place pixel (0,0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD24bpp, LCD32bpp, LCD2bpp, LCD4bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, and LCD1bpp.

7.109.3.13 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

rect	The rectangle x, y, width, height in absolute coordinates.
color	The color to use.
alpha	The alpha value to use (default=solid)

7.109.3.14 drawString()

```
void drawString (
             Rect widgetArea,
             const Rect & invalidatedArea,
             const StringVisuals & stringVisuals,
             const Unicode::UnicodeChar * format,
              ...  )
```

Draws the specified unicode string. Breaks line on newline.

Parameters

widgetArea	The area covered by the drawing widget in absolute coordinates.
invalidatedArea	The (sub)region of the widget area to draw, expressed relative to the widget area. If the widgetArea is (x, y, width, height) = (10, 10, 20, 20) and invalidatedArea is (x, y, width, height) = (5, 5, 6, 6) the widgetArea drawn on the LCD is (x, y, width, height) = (15, 15, 6, 6).
stringVisuals	The string visuals (font, alignment, line space, color) with which to draw this string.
format	A pointer to a zero terminated text string with optional additional wildcard arguments.
...	Variable arguments providing additional information.

7.109.3.15 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

	widgetArea	The area covered by the drawing widget in absolute coordinates.
	invalidatedArea	The (sub)region of the widget area to draw, expressed relative to the widget area. If the widgetArea is (x, y, width, height) = (10, 10, 20, 20) and invalidatedArea is (x, y, width, height) = (5, 5, 6, 6) the widgetArea drawn on the LCD is (x, y, width, height) = (15, 15, 6, 6).
in	*visuals*	The string visuals (font, alignment, line space, color) with which to draw this string.
	format	A pointer to a zero terminated text string with optional additional wildcard arguments.
	pArg	Variable arguments providing additional information.

See also

drawString

7.109.3.16 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

	widgetArea	The area covered by the drawing widget in absolute coordinates.

Parameters

	invalidatedArea	The (sub)region of the widget area to draw, expressed relative to the widget area. If the widgetArea is (x, y, width, height) = (10, 10, 20, 20) and invalidatedArea is (x, y, width, height) = (5, 5, 6, 6) the widgetArea drawn on the LCD is (x, y, width, height) = (15, 15, 6, 6).
in	*visuals*	The string visuals (font, alignment, line space, color) with which to draw this string.
	format	A pointer to a zero terminated text string with optional additional wildcard arguments.
	pArg	Variable arguments providing additional information.

See also

drawString

7.109.3.17 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionLength	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD2bpp, LCD24bpp, LCD32bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.18 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
vertices	The vertices of the triangle.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	the alpha. Default = 255.
subDivisionSize	the size of the subdivisions of the scan line. Default = 12.

7.109.3.19 drawVerticalLine()

```
void drawVerticalLine (
            int16_t x,
            int16_t y,
            uint16_t height,
            uint16_t lineWidth,
            colortype color,
            uint8_t alpha = 255 )
```

Draws a vertical line with the specified color.

Parameters

x	The x coordinate of the starting point, in absolute coordinates.
y	The y coordinate of the starting point, in absolute coordinates.
height	The length of the line.
lineWidth	The width of the line.
color	The color to use.
alpha	The alpha value to use (default=solid)

7.109.3.20 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 )  [pure virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD24bpp, LCD32bpp, LCD2bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.21 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [pure virtual]
```

Framebuffer format used by the display

Returns

A Bitmap::BitmapFormat.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD24bpp, LCD32bpp, LCD2bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.22 framebufferStride()

```
uint16_t framebufferStride ( ) const  [pure virtual]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD24bpp, LCD32bpp, LCD2bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.23 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [pure virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD2bpp, LCD32bpp, LCD24bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.24 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [pure virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values. Depending on your chosen color bit depth, the color will be interpreted internally as either a 16 bit or 24 bit color value. This function can be safely used regardless of whether your application is configured for 16 or 24 bit colors.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD24bpp, LCD32bpp, LCD2bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.25 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [pure virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD2bpp, LCD32bpp, LCD24bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.26 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	*textProvider*	The text provider.
	wideTextAction	The wide text action in case lines are longer than the width of the text area.
	textDirection	The text direction (LTR or RTL).
	font	The font.
	width	The width.

Returns

The number lines.

7.109.3.27 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [pure virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implemented in LCD16bppSerialFlash, LCD16bpp, LCD2bpp, LCD32bpp, LCD24bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, LCD8bpp_RGBA2222, LCD4bpp, and LCD1bpp.

7.109.3.28 init()

```
void init ( )  [inline], [virtual]
```

Performs initialization.

Reimplemented in LCD16bppSerialFlash, LCD16bpp, LCD2bpp, LCD4bpp, LCD24bpp, LCD32bpp, LCD8bpp_ABGR2222, LCD8bpp_ARGB2222, LCD8bpp_BGRA2222, and LCD8bpp_RGBA2222.

7.109.3.29 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	*widgetArea*	The widget containing the point.
	x	The x coordinate.
	y	The y coordinate.
	rotation	Rotation to perform.

Returns

The absolute x coordinate after applying appropriate rotation.

7.109.3.30 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	*widgetArea*	The widget containing the point.
	x	The x coordinate.
	y	The y coordinate.
	rotation	Rotation to perform.

Returns

The absolute y coordinate after applying appropriate rotation.

7.109.3.31 rotateRect()

```
static void rotateRect (
             Rect & rect,
             const Rect & canvas,
             const TextRotation rotation )  [static], [protected]
```

Rotate a rectangle inside another rectangle.

Parameters

in,out		
	rect	The rectangle to rotate.
	canvas	The rectangle containing the rect to rotate.
	rotation	Rotation to perform on rect.

7.109.3.32 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 ligatures read from the given TextProvider. After the introduction of Arabic, Thai, Hindi and other languages, ligatures are counted instead of characters. For Latin languages, number of characters equal number of ligatures.

Parameters

in,out		
	textProvider	The text provider.
	font	The font.
	numChars	Number of characters (ligatures).
	textDirection	The text direction.

Returns

An int16_t.

7.110 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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.

- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (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.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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 *wbuf16, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offsetY, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  Private version of draw-glyph with explicit destination buffer pointer argument.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  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 blitCopyL8 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.
- static void blitCopyL8_ARGB8888 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.
- static void blitCopyL8_RGB565 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.
- static void blitCopyL8_RGB888 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source 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.

7.110.1 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

7.110.2 Member Function Documentation

7.110.2.1 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.

7.110.2.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.110.2.3 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 perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD16 supports source data formats: RGB565 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.110.2.4 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 perfoming alpha-blending per pixel as specified. Always performs a software blend.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16- bits RGB565 values.
alphaData	The alpha channel array pointer (points to the beginning of the data)
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.110.2.5 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 perfoming alpha-blending per pixel as specified. If ARGB8888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.110.2.6 blitCopyL8()

```
static void blitCopyL8 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if indexed format is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.110.2.7 blitCopyL8_ARGB8888()

```
static void blitCopyL8_ARGB8888 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_ARGB8888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries stored as 32- bits (ARGB8888) format.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.110.2.8 blitCopyL8_RGB565()

```
static void blitCopyL8_RGB565 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_RGB565 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer points to the beginning of the CLUT color format and size data followed by colors entries stored as 16- bits (RGB565) format. If the source have per pixel alpha channel, then alpha channel data will be following the clut entries data.
source	The location and dimension of the source.

Parameters

blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.110.2.9 blitCopyL8_RGB888()

```
static void blitCopyL8_RGB888 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_RGB888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries stored as 32- bits (ARGB8888) format.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.110.2.10 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
             const Rect & visRegion,
             const Rect & absRegion,
             const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANI↩ MATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.110.2.11 drawGlyph()

```
void drawGlyph (
              uint16_t * wbuf16,
              Rect widgetArea,
              int16_t x,
              int16_t y,
              uint16_t offsetX,
              uint16_t offsetY,
              const Rect & invalidatedArea,
              const GlyphNode * glyph,
              const uint8_t * glyphData,
              uint8_t dataFormatA4,
              colortype color,
              uint8_t bitsPerPixel,
              uint8_t alpha,
              TextRotation rotation )  [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	*wbuf16*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.110.2.12 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

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.110.2.13 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.110.2.14 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 )  [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.110.2.15 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::RGB565.

Implements LCD.

7.110.2.16 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.110.2.17 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.110.2.18 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.110.2.19 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.110.2.20 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

7.110.2.21 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.110.2.22 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.110.2.23 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.110.2.24 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.110.2.25 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.110.2.26 init()

```
void init ( )  [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.110.2.27 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.110.2.28 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.111 LCD16bppSerialFlash Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

`#include <platform/driver/lcd/LCD16bppSerialFlash.hpp>`

Public Member Functions

- LCD16bppSerialFlash (FlashDataReader &flashReader)

  Creates a LCD16bppSerialFlash object.
- 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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.
- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (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.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation=TEXT_ROTATE_0)

  Private version of draw-glyph with explicit destination buffer pointer argument.
- void blitCopyARGB8888 (const uint32_t *sourceData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D source-array to the framebuffer.
- void blitCopyL8 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.
- void blitCopyL8_ARGB8888 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.

- void blitCopyL8_RGB565 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next line.

Protected Attributes

- FlashDataReader & flashReader

  Flash reader. Used by routines to read pixel data from the flash.

Static Protected Attributes

- static const uint16_t TRANSPARENT_COL = 0xABCD

  Transparency color. Deprecated, do not use.

7.111.1 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

7.111.2 Constructor & Destructor Documentation

7.111.2.1 LCD16bppSerialFlash()

```
LCD16bppSerialFlash (
            FlashDataReader & flashReader )
```

Creates a LCD16bppSerialFlash object. The FlashDataReader object is used to fetch data from the external flash.

Parameters

in	*flashReader*	Reference to a FlashDataReader object

7.111.3 Member Function Documentation

7.111.3.1 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.

7.111.3.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.111.3.3 blitCopy() [2/2]

```
void blitCopy (
            const uint8_t * sourceData,
            Bitmap::BitmapFormat sourceFormat,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha,
            bool hasTransparentPixels )  [virtual]
```

Blits a 2D source-array to the framebuffer perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD16 supports source data formats: RGB565 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.111.3.4 blitCopyARGB8888()

```
static void blitCopyARGB8888 (
             const uint32_t * sourceData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [protected]
```

Blits a 2D source-array to the framebuffer perfoming alpha-blending per pixel as specified if ARGB8888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.111.3.5 blitCopyL8()

```
void blitCopyL8 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if indexed format is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.111.3.6 blitCopyL8_ARGB8888()

```
void blitCopyL8_ARGB8888 (
            const uint8_t * sourceData,
            const uint8_t * clutData,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_ARGB8888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries stored as 32- bits (ARGB8888) format.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.111.3.7 blitCopyL8_RGB565()

```
void blitCopyL8_RGB565 (
            const uint8_t * sourceData,
            const uint8_t * clutData,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_RGB565 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer points to the beginning of the CLUT color format and size data followed by colors entries stored as 16- bits (RGB565) format. If the source have per pixel alpha channel, then alpha channel data will be following the clut entries data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.111.3.8 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
            const Rect & visRegion,
```

```
            const Rect & absRegion,
            const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANI↩ MATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.111.3.9 drawGlyph()

```
void drawGlyph (
            uint16_t * wbuf16,
            Rect widgetArea,
            int16_t x,
            int16_t y,
            uint16_t offsetX,
            uint16_t offsetY,
            const Rect & invalidatedArea,
            const GlyphNode * glyph,
            const uint8_t * glyphData,
            uint8_t dataFormatA4,
            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	*wbuf16*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.

Parameters

	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.111.3.10 drawPartialBitmap()

```
void drawPartialBitmap (
             const Bitmap & bitmap,
             int16_t x,
             int16_t y,
             const Rect & rect,
             uint8_t alpha = 255,
             bool useOptimized = true )  [virtual]
```

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.111.3.11 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.111.3.12 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 )  [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.111.3.13 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::RGB565.

Implements LCD.

7.111.3.14 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.111.3.15 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.111.3.16 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.111.3.17 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.111.3.18 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

7.111.3.19 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.111.3.20 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.111.3.21 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.111.3.22 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.111.3.23 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.111.3.24 init()

```
void init ( )  [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.111.3.25 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.111.3.26 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.112 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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.
- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (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.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  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.
- void copyRect (const uint8_t *srcAddress, uint16_t srcStride, uint8_t srcPixelOffset, uint8_t *RESTRICT dstAddress, uint16_t dstStride, uint8_t dstPixelOffset, uint16_t width, uint16_t height) const

  Copies a rectangular area.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next line.
- static void fillMemory (void *RESTRICT dst, colortype color, uint16_t bytesToFill)

  Fill memory efficiently.

Additional Inherited Members

7.112.1 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

7.112.2 Member Function Documentation

7.112.2.1 bitDepth()

```
uint8_t bitDepth ( ) const  [inline], [virtual]
```

Number of bits per pixel used by the display.

Returns

1.

Implements LCD.

7.112.2.2 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

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (0 = invisible, otherwise solid).
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.112.2.3 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 perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	Ignored

Implements LCD.

7.112.2.4 blitCopyRLE()

```
void blitCopyRLE (
            const uint16_t * _sourceData,
            const Rect & source,
```

```
            const Rect & blitRect,
            uint8_t alpha )  [protected], [virtual]
```

Blits a run-length encoded2D source-array to the frame buffer unless alpha is zero.

Parameters

_sourceData	The source-array pointer (points to the beginning of the data). Data stored in RLE format, where each byte indicates number of pixels with certain color, alternating between black and white. First byte represents black.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (0 = invisible, otherwise solid).

7.112.2.5 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
            const Rect & visRegion,
            const Rect & absRegion,
            const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANI↩ MATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.112.2.6 copyRect()

```
void copyRect (
            const uint8_t * srcAddress,
```

```
            uint16_t srcStride,
            uint8_t srcPixelOffset,
            uint8_t *RESTRICT dstAddress,
            uint16_t dstStride,
            uint8_t dstPixelOffset,
            uint16_t width,
            uint16_t height ) const  [protected]
```

Copies a rectangular area.

Parameters

	srcAddress	Source address (byte address).
	srcStride	Source stride (number of bytes to advance to next line).
	srcPixelOffset	Source pixel offset (first pixel in first source byte).
in	*dstAddress*	If destination address (byte address).
	dstStride	Destination stride (number of bytes to advance to next line).
	dstPixelOffset	Destination pixel offset (first pixel in destination byte).
	width	The width of area (in pixels).
	height	The height of area (in pixels).

7.112.2.7 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,
            uint8_t dataFormatA4,
            colortype color,
            uint8_t bitsPerPixel,
            uint8_t alpha,
            TextRotation rotation )  [protected], [virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public function drawString().

Parameters

in	*wbuf*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.

Parameters

	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.112.2.8 drawPartialBitmap()

```
void drawPartialBitmap (
             const Bitmap & bitmap,
             int16_t x,
             int16_t y,
             const Rect & rect,
             uint8_t alpha = 255,
             bool useOptimized = true )  [virtual]
```

Draws a portion of a bitmap.

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value (0 = invisible, otherwise solid). Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.112.2.9 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 )  [inline], [protected], [virtual]
```

Parameters

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.

Parameters

texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.112.2.10 fillMemory()

```
static void fillMemory (
             void *RESTRICT dst,
             colortype color,
             uint16_t bytesToFill )  [static], [protected]
```

Fill memory efficiently. Try to get 32bit aligned or 16bit aligned and then copy as quickly as possible.

Parameters

out	*dst*	Pointer to memory to fill.
	color	Color to write to memory, either 0 => 0x00000000 or 1 => 0xFFFFFFFF.
	bytesToFill	Number of bytes to fill.

7.112.2.11 fillRect()

```
void fillRect (
             const Rect & rect,
             colortype color,
             uint8_t alpha = 255 )  [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color (values other than 0 or 1 are treated as being 1).
alpha	The rectangle opacity (0 = invisible, otherwise solid). Default is 255 (solid).

Implements LCD.

7.112.2.12 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::BW.

Implements LCD.

7.112.2.13 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.112.2.14 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.112.2.15 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.112.2.16 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.112.2.17 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

7.112.2.18 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.112.2.19 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.112.2.20 getGreenFromColor()

```
uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.112.2.21 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.112.2.22 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.112.2.23 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.112.2.24 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.113 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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.
- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (uint8_t red, uint8_t green, uint8_t blue)

  Gets color from RGB.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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 *wbuf16, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offsetY, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  Private version of draw-glyph with explicit destination buffer pointer argument.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  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 blitCopyL8 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.
- static void blitCopyL8_ARGB8888 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.
- static void blitCopyL8_RGB888 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.

Static Protected Attributes

- static const uint16_t TRANSPARENT_COL = 0xABCD

  Transparency color.

7.113.1 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

7.113.2 Member Function Documentation

7.113.2.1 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.

7.113.2.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.113.2.3 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 perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD16 supports source data formats: RGB888 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.113.2.4 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 perfoming alpha-blending per pixel as specified if ARGB8888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.113.2.5 blitCopyL8()

```
static void blitCopyL8 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if indexed format is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.113.2.6 blitCopyL8_ARGB8888()

```
static void blitCopyL8_ARGB8888 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_ARGB8888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries stored as 32- bits (ARGB8888) format.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.113.2.7 blitCopyL8_RGB888()

```
static void blitCopyL8_RGB888 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_RGB888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries stored as 32- bits (RGB888) format.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.113.2.8 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
             const Rect & visRegion,
             const Rect & absRegion,
```

```
            const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANI↩ MATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.113.2.9 drawGlyph()

```
void drawGlyph (
            uint16_t * wbuf16,
            Rect widgetArea,
            int16_t x,
            int16_t y,
            uint16_t offsetX,
            uint16_t offsetY,
            const Rect & invalidatedArea,
            const GlyphNode * glyph,
            const uint8_t * glyphData,
            uint8_t dataFormatA4,
            colortype color,
            uint8_t bitsPerPixel,
            uint8_t alpha,
            TextRotation rotation )  [protected], [virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public function drawString().

Parameters

in	*wbuf16*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.

Parameters

	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.113.2.10 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

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.113.2.11 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.113.2.12 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 )  [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.113.2.13 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::RGB888.

Implements LCD.

7.113.2.14 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.113.2.15 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.113.2.16 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.113.2.17 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.113.2.18 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Parameters

red	The red.
green	The green.
blue	The blue.

Returns

The color from RGB.

7.113.2.19 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.113.2.20 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.113.2.21 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.113.2.22 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.113.2.23 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.113.2.24 init()

```
void init ( )  [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.113.2.25 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.113.2.26 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.114 LCD24DebugPrinter Class Reference

The class LCD24DebugPrinter implements the DebugPrinter interface for printing debug messages on top of 24bit framebuffer.

```
#include <platform/driver/lcd/LCD24bpp.hpp>
```

Public Member Functions

- virtual void draw (const LCD &lcd) const

Draws the debug string on top of the framebuffer content.

Additional Inherited Members

7.114.1 Detailed Description

The class LCD24DebugPrinter implements the DebugPrinter interface for printing debug messages on top of 24bit framebuffer.

See also

DebugPrinter

7.114.2 Member Function Documentation

7.114.2.1 draw()

```
virtual void draw (
            const LCD & lcd ) const  [virtual]
```

Draws the debug string on top of the framebuffer content.

Parameters

in	*lcd*	Reference on the LCD object to use for drawing the debug string.

Implements DebugPrinter.

7.115 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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.

- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getRedFromColor (colortype color) const

  Gets red from color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (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.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  Private version of draw-glyph with explicit destination buffer pointer argument.
- void copyRect (const uint8_t *srcAddress, uint16_t srcStride, uint8_t srcPixelOffset, uint8_t *RESTRICT dstAddress, uint16_t dstStride, uint8_t dstPixelOffset, uint16_t width, uint16_t height) const

  Copies a rectangular area.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  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)

  Blit 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 [256]

  The alpha lookup table to avoid arithmetics when alpha blending.

7.115.1 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

7.115.2 Member Function Documentation

7.115.2.1 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.

7.115.2.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.115.2.3 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 perfoming alpha-blending (and tranparency 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

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.115.2.4 blitCopyAlphaPerPixel()

```
static void blitCopyAlphaPerPixel (
            const uint16_t * sourceData16,
            const uint8_t * sourceAlphaData,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [static], [protected]
```

Blit a 2D source-array to the framebuffer performing alpha-blending per pixel as specified Performs always a software blend.

Parameters

sourceData16	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 2bpp GRAY2 values.
sourceAlphaData	The alpha channel array pointer (points to the beginning of the data)
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.115.2.5 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
            const Rect & visRegion,
            const Rect & absRegion,
            const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANIMATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.115.2.6 copyRect()

```
void copyRect (
            const uint8_t * srcAddress,
            uint16_t srcStride,
```

```
            uint8_t srcPixelOffset,
            uint8_t *RESTRICT dstAddress,
            uint16_t dstStride,
            uint8_t dstPixelOffset,
            uint16_t width,
            uint16_t height ) const  [protected]
```

Copies a rectangular area.

Parameters

	srcAddress	Source address (byte address).
	srcStride	Source stride (number of bytes to advance to next line).
	srcPixelOffset	Source pixel offset (first pixel in first source byte).
in	*dstAddress*	If destination address (byte address).
	dstStride	Destination stride (number of bytes to advance to next line).
	dstPixelOffset	Destination pixel offset (first pixel in destination byte).
	width	The width of area (in pixels).
	height	The height of area (in pixels).

7.115.2.7 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,
            uint8_t dataFormatA4,
            colortype color,
            uint8_t bitsPerPixel,
            uint8_t alpha,
            TextRotation rotation )  [protected], [virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public function drawString().

Parameters

in	*wbuf*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.

Parameters

	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.115.2.8 drawPartialBitmap()

```
void drawPartialBitmap (
             const Bitmap & bitmap,
             int16_t x,
             int16_t y,
             const Rect & rect,
             uint8_t alpha = 255,
             bool useOptimized = true )  [virtual]
```

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.115.2.9 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.

Parameters

leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.115.2.10 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 )  [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.115.2.11 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::GRAY2.

Implements LCD.

7.115.2.12 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.115.2.13 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.115.2.14 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.115.2.15 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.115.2.16 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

7.115.2.17 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.115.2.18 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.115.2.19 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.115.2.20 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.115.2.21 getRedFromColor()

```
uint8_t getRedFromColor (
            colortype color ) const  [inline], [virtual]
```

Parameters

color	The color.

Returns

The red from color.

7.115.2.22 init()

```
void init ( )  [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.115.2.23 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.115.2.24 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.116 LCD32bpp Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

`#include <platform/driver/lcd/LCD32bpp.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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.
- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (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.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  Private version of draw-glyph with explicit destination buffer pointer argument.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next line.
- static void blitCopyRGB888 (const uint16_t *sourceData16, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D source-array to the framebuffer.
- static void blitCopyRGB565 (const uint16_t *sourceData16, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D source-array to the framebuffer.
- static void blitCopyL8 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.
- static void blitCopyL8_ARGB8888 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.
- static void blitCopyL8_RGB888 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.
- static void blitCopyL8_RGB565 (const uint8_t *sourceData, const uint8_t *clutData, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blits a 2D indexed 8-bit source to the framebuffer.

Additional Inherited Members

7.116.1 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

7.116.2 Member Function Documentation

7.116.2.1 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.

7.116.2.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32-bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.116.2.3 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 perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD32 supports source data formats: RGB565, RGB888 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.116.2.4 blitCopyL8()

```
static void blitCopyL8 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if indexed format is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.116.2.5 blitCopyL8_ARGB8888()

```
static void blitCopyL8_ARGB8888 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_ARGB8888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries stored as 32- bits (ARGB8888) format.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.116.2.6 blitCopyL8_RGB565()

```
static void blitCopyL8_RGB565 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_RGB565 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries stored as 16- bits (RGB565) format.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.116.2.7 blitCopyL8_RGB888()

```
static void blitCopyL8_RGB888 (
             const uint8_t * sourceData,
             const uint8_t * clutData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D indexed 8-bit source to the framebuffer perfoming alpha-blending per pixel as specified if L8_RGB888 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-indexes pointer (points to the beginning of the data). The sourceData must be stored as 8- bits indexes.
clutData	The source-clut pointer (points to the beginning of the CLUT color format and size data followed by colors entries stored as 32- bits (RGB888) format.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.116.2.8 blitCopyRGB565()

```
static void blitCopyRGB565 (
             const uint16_t * sourceData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D source-array to the framebuffer perfoming alpha-blending per pixel as specified. If! RGB565 is not supported by the DMA a software blend is performed.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16- bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.

Parameters

alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.116.2.9 blitCopyRGB888()

```
static void blitCopyRGB888 (
             const uint16_t * sourceData16,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blits a 2D source-array to the framebuffer perfoming alpha-blending per pixel as specified. If RGB888 is not supported by the DMA a software blend is performed.

Parameters

sourceData16	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 24- bits RGB888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.116.2.10 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
             const Rect & visRegion,
             const Rect & absRegion,
             const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANI↩MATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.116.2.11 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,
             uint8_t dataFormatA4,
             colortype color,
             uint8_t bitsPerPixel,
             uint8_t alpha,
             TextRotation rotation )  [protected], [virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public version of drawGlyph().

Parameters

in	*wbuf*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.116.2.12 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

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.116.2.13 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.116.2.14 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 )  [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.116.2.15 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::ARGB8888.

Implements LCD.

7.116.2.16 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.116.2.17 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.116.2.18 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.116.2.19 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.116.2.20 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).

Parameters

green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color from RGB.

7.116.2.21 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.116.2.22 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.116.2.23 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.116.2.24 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.116.2.25 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.116.2.26 init()

```
void init ( )  [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.116.2.27 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.116.2.28 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.117 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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.
- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.

- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (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.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  Private version of draw-glyph with explicit destination buffer pointer argument.
- void copyRect (const uint8_t *srcAddress, uint16_t srcStride, uint8_t srcPixelOffset, uint8_t *RESTRICT dstAddress, uint16_t dstStride, uint8_t dstPixelOffset, uint16_t width, uint16_t height) const

  Copies a rectangular area.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  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)

  Blit a 2D source-array to the framebuffer.

Static Protected Attributes

- static const uint16_t TRANSPARENT_COL = 0xABCD

  Transparency color. Deprecated, do not use.

7.117.1 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

7.117.2 Member Function Documentation

7.117.2.1 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.

7.117.2.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.117.2.3 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 perfoming alpha-blending (and tranparency 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

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.117.2.4 blitCopyAlphaPerPixel()

```
static void blitCopyAlphaPerPixel (
              const uint16_t * sourceData16,
              const uint8_t * sourceAlphaData,
              const Rect & source,
              const Rect & blitRect,
              uint8_t alpha )  [static], [protected]
```

Blit a 2D source-array to the framebuffer performing alpha-blending per pixel as specified Performs always a software blend.

Parameters

sourceData16	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 4bpp GRAY4 values.
sourceAlphaData	The alpha channel array pointer (points to the beginning of the data)
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.117.2.5 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
             const Rect & visRegion,
             const Rect & absRegion,
             const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANIMATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

Implements LCD.

7.117.2.6 copyRect()

```
void copyRect (
             const uint8_t * srcAddress,
             uint16_t srcStride,
             uint8_t srcPixelOffset,
             uint8_t *RESTRICT dstAddress,
             uint16_t dstStride,
             uint8_t dstPixelOffset,
             uint16_t width,
             uint16_t height ) const  [protected]
```

Copies a rectangular area.

Parameters

	srcAddress	Source address (byte address).
	srcStride	Source stride (number of bytes to advance to next line).
	srcPixelOffset	Source pixel offset (first pixel in first source byte).
in	*dstAddress*	If destination address (byte address).
	dstStride	Destination stride (number of bytes to advance to next line).
	dstPixelOffset	Destination pixel offset (first pixel in destination byte).
	width	The width of area (in pixels).
	height	The height of area (in pixels).

7.117.2.7 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,
            uint8_t dataFormatA4,
            colortype color,
            uint8_t bitsPerPixel,
            uint8_t alpha,
            TextRotation rotation )  [protected], [virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public function drawString().

Parameters

in	*wbuf*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.117.2.8 drawPartialBitmap()

```
void drawPartialBitmap (
            const Bitmap & bitmap,
            int16_t x,
            int16_t y,
            const Rect & rect,
            uint8_t alpha = 255,
            bool useOptimized = true )  [virtual]
```

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.

Parameters

y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.117.2.9 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.117.2.10 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 )  [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.117.2.11 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::GRAY4.

Implements LCD.

7.117.2.12 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.117.2.13 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.117.2.14 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.117.2.15 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.117.2.16 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

7.117.2.17 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.117.2.18 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.117.2.19 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.117.2.20 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.117.2.21 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.117.2.22 init()

```
void init ( )  [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.117.2.23 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.117.2.24 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.118 LCD8bpp_ABGR2222 Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

`#include <platform/driver/lcd/LCD8bpp_ABGR2222.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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.
- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.

- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (uint8_t red, uint8_t green, uint8_t blue)

  Gets color from RGB.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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 *wbuf8, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offsetY, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  Private version of draw-glyph with explicit destination buffer pointer argument.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  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)

  Blit a 2D source-array to the framebuffer.
- static void blitCopyAlphaPerPixel (const uint16_t *sourceData16, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blit a 2D source-array to the framebuffer.

Static Protected Attributes

- static const uint16_t TRANSPARENT_COL = 0xABCD

  Transparency color. Deprecated, do not use.

7.118.1 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

7.118.2 Member Function Documentation

7.118.2.1 bitDepth()

```
uint8_t bitDepth ( ) const  [inline], [virtual]
```

Number of bits per pixel used by the display.

Returns

8.

Implements LCD.

7.118.2.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.118.2.3 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 perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD16 supports source data formats: RGB565 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.118.2.4 blitCopyAlphaPerPixel()

```
static void blitCopyAlphaPerPixel (
            const uint16_t * sourceData16,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [static], [protected]
```

Blit a 2D source-array to the framebuffer performing alpha-blending per pixel as specified Performs always a software blend.

Parameters

sourceData16	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 8-bits ABGR2222 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.118.2.5 blitCopyARGB8888()

```
static void blitCopyARGB8888 (
             const uint32_t * sourceData,
             const Rect & source,
             const Rect & blitRect,
             uint8_t alpha )  [static], [protected]
```

Blit 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

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.118.2.6 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
             const Rect & visRegion,
             const Rect & absRegion,
             const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANI↩ MATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.118.2.7 drawGlyph()

```
void drawGlyph (
            uint16_t * wbuf8,
            Rect widgetArea,
            int16_t x,
            int16_t y,
            uint16_t offsetX,
            uint16_t offsetY,
            const Rect & invalidatedArea,
            const GlyphNode * glyph,
            const uint8_t * glyphData,
            uint8_t dataFormatA4,
            colortype color,
            uint8_t bitsPerPixel,
            uint8_t alpha,
            TextRotation rotation )  [protected], [virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public function drawString().

Parameters

in	*wbuf8*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.118.2.8 drawPartialBitmap()

```
void drawPartialBitmap (
            const Bitmap & bitmap,
            int16_t x,
            int16_t y,
            const Rect & rect,
            uint8_t alpha = 255,
            bool useOptimized = true )  [virtual]
```

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.

Parameters

y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.118.2.9 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.118.2.10 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 ) [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.118.2.11 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::ABGR2222.

Implements LCD.

7.118.2.12 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.118.2.13 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.118.2.14 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.118.2.15 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.118.2.16 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Parameters

red	The red.
green	The green.
blue	The blue.

Returns

The color from RGB.

7.118.2.17 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.118.2.18 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.118.2.19 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.118.2.20 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.118.2.21 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color ) [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.118.2.22 init()

```
void init ( ) [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.118.2.23 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation ) [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.118.2.24 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation ) [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.119 LCD8bpp_ARGB2222 Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

`#include <platform/driver/lcd/LCD8bpp_ARGB2222.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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.
- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (uint8_t red, uint8_t green, uint8_t blue)

  Gets color from RGB.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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 *wbuf8, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offsetY, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  Private version of draw-glyph with explicit destination buffer pointer argument.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  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)

  Blit a 2D source-array to the framebuffer.
- static void blitCopyAlphaPerPixel (const uint16_t *sourceData16, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blit a 2D source-array to the framebuffer.

Static Protected Attributes

- static const uint16_t TRANSPARENT_COL = 0xABCD

  Transparency color. Deprecated, do not use.

7.119.1 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

7.119.2 Member Function Documentation

7.119.2.1 bitDepth()

```
uint8_t bitDepth ( ) const  [inline], [virtual]
```

Number of bits per pixel used by the display.

Returns

8.

Implements LCD.

7.119.2.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.119.2.3 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 perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD16 supports source data formats: RGB565 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.119.2.4 blitCopyAlphaPerPixel()

```
static void blitCopyAlphaPerPixel (
            const uint16_t * sourceData16,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [static], [protected]
```

Blit a 2D source-array to the framebuffer performing alpha-blending per pixel as specified Performs always a software blend.

Parameters

sourceData16	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 8-bits ARGB2222 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.119.2.5 blitCopyARGB8888()

```
static void blitCopyARGB8888 (
            const uint32_t * sourceData,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [static], [protected]
```

Blit 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

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.119.2.6 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
              const Rect & visRegion,
              const Rect & absRegion,
              const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANI↩ MATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.119.2.7 drawGlyph()

```
void drawGlyph (
              uint16_t * wbuf8,
              Rect widgetArea,
              int16_t x,
              int16_t y,
              uint16_t offsetX,
              uint16_t offsetY,
```

```
                const Rect & invalidatedArea,
                const GlyphNode * glyph,
                const uint8_t * glyphData,
                uint8_t dataFormatA4,
                colortype color,
                uint8_t bitsPerPixel,
                uint8_t alpha,
                TextRotation rotation )  [protected], [virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public function drawString().

Parameters

in	*wbuf8*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.119.2.8 drawPartialBitmap()

```
void drawPartialBitmap (
                const Bitmap & bitmap,
                int16_t x,
                int16_t y,
                const Rect & rect,
                uint8_t alpha = 255,
                bool useOptimized = true )  [virtual]
```

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.119.2.9 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.119.2.10 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 )  [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.119.2.11 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::ARGB2222.

Implements LCD.

7.119.2.12 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.119.2.13 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.119.2.14 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.119.2.15 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.119.2.16 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Parameters

red	The red.
green	The green.
blue	The blue.

Returns

The color from RGB.

7.119.2.17 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.119.2.18 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.119.2.19 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.119.2.20 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.119.2.21 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.119.2.22 init()

```
void init ( )  [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.119.2.23 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.119.2.24 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.120 LCD8bpp_BGRA2222 Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

`#include <platform/driver/lcd/LCD8bpp_BGRA2222.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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.
- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (uint8_t red, uint8_t green, uint8_t blue)

  Gets color from RGB.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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 *wbuf8, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offsetY, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  Private version of draw-glyph with explicit destination buffer pointer argument.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  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)

  Blit a 2D source-array to the framebuffer.
- static void blitCopyAlphaPerPixel (const uint16_t *sourceData16, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blit a 2D source-array to the framebuffer.

Static Protected Attributes

- static const uint16_t TRANSPARENT_COL = 0xABCD

  Transparency color. Deprecated, do not use.

7.120.1 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

7.120.2 Member Function Documentation

7.120.2.1 bitDepth()

```
uint8_t bitDepth ( ) const  [inline], [virtual]
```

Number of bits per pixel used by the display.

Returns

8.

Implements LCD.

7.120.2.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.120.2.3 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 perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD16 supports source data formats: RGB565 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.120.2.4 blitCopyAlphaPerPixel()

```
static void blitCopyAlphaPerPixel (
            const uint16_t * sourceData16,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [static], [protected]
```

Blit a 2D source-array to the framebuffer performing alpha-blending per pixel as specified Performs always a software blend.

Parameters

sourceData16	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 8-bits BGRA2222 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.120.2.5 blitCopyARGB8888()

```
static void blitCopyARGB8888 (
            const uint32_t * sourceData,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [static], [protected]
```

Blit 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

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.120.2.6 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
            const Rect & visRegion,
            const Rect & absRegion,
            const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANI↩ MATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.120.2.7 drawGlyph()

```
void drawGlyph (
            uint16_t * wbuf8,
            Rect widgetArea,
            int16_t x,
            int16_t y,
            uint16_t offsetX,
            uint16_t offsetY,
```

```
            const Rect & invalidatedArea,
            const GlyphNode * glyph,
            const uint8_t * glyphData,
            uint8_t dataFormatA4,
            colortype color,
            uint8_t bitsPerPixel,
            uint8_t alpha,
            TextRotation rotation )  [protected], [virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public function drawString().

Parameters

in	*wbuf8*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.0
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.120.2.8 drawPartialBitmap()

```
void drawPartialBitmap (
            const Bitmap & bitmap,
            int16_t x,
            int16_t y,
            const Rect & rect,
            uint8_t alpha = 255,
            bool useOptimized = true )  [virtual]
```

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.120.2.9 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.120.2.10 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 ) [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.120.2.11 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::BGRA2222.

Implements LCD.

7.120.2.12 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.120.2.13 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.120.2.14 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.120.2.15 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.120.2.16 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Parameters

red	The red.
green	The green.
blue	The blue.

Returns

The color from RGB.

7.120.2.17 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.120.2.18 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.120.2.19 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.120.2.20 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.120.2.21 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color ) [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.120.2.22 init()

```
void init ( ) [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.120.2.23 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation ) [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.120.2.24 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation ) [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.121 LCD8bpp_RGBA2222 Class Reference

This class contains the various low-level drawing routines for drawing bitmaps.

`#include <platform/driver/lcd/LCD8bpp_RGBA2222.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 &visRegion, const Rect &absRegion, const BitmapId bitmapId)

  Copies part of the frame buffer region to memory.
- 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.
- virtual Bitmap::BitmapFormat framebufferFormat () const

  Framebuffer format used by the display.
- virtual uint16_t framebufferStride () const

  Framebuffer stride in bytes.
- virtual colortype getColorFrom24BitRGB (uint8_t red, uint8_t green, uint8_t blue) const

  Generates a color representation to be used on the LCD, based on 24 bit RGB values.
- virtual uint8_t getRedColor (colortype color) const

  Gets the red color part of a color.
- virtual uint8_t getGreenColor (colortype color) const

  Gets the green color part of a color.
- virtual uint8_t getBlueColor (colortype color) const

  Gets the blue color part of a color.

Static Public Member Functions

- static FORCE_INLINE_FUNCTION uint16_t getFramebufferStride ()

  Framebuffer stride in bytes.
- static FORCE_INLINE_FUNCTION colortype getColorFromRGB (uint8_t red, uint8_t green, uint8_t blue)

  Gets color from RGB.
- static FORCE_INLINE_FUNCTION uint8_t getRedFromColor (colortype color)

  Gets red from color.
- static FORCE_INLINE_FUNCTION uint8_t getGreenFromColor (colortype color)

  Gets green from color.
- static FORCE_INLINE_FUNCTION uint8_t getBlueFromColor (colortype color)

  Gets blue from color.

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 *wbuf8, Rect widgetArea, int16_t x, int16_t y, uint16_t offsetX, uint16_t offsetY, const Rect &invalidatedArea, const GlyphNode *glyph, const uint8_t *glyphData, uint8_t dataFormatA4, colortype color, uint8_t bitsPerPixel, uint8_t alpha, TextRotation rotation)

  Private version of draw-glyph with explicit destination buffer pointer argument.

Static Protected Member Functions

- static int nextPixel (bool rotatedDisplay, TextRotation textRotation)

  Find out how much to advance in the display buffer to get to the next pixel.
- static int nextLine (bool rotatedDisplay, TextRotation textRotation)

  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)

  Blit a 2D source-array to the framebuffer.
- static void blitCopyAlphaPerPixel (const uint16_t *sourceData16, const Rect &source, const Rect &blitRect, uint8_t alpha)

  Blit a 2D source-array to the framebuffer.

Static Protected Attributes

- static const uint16_t TRANSPARENT_COL = 0xABCD

  Transparency color. Deprecated, do not use.

7.121.1 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

7.121.2 Member Function Documentation

7.121.2.1 bitDepth()

```
uint8_t bitDepth ( ) const  [inline], [virtual]
```

Number of bits per pixel used by the display.

Returns

8.

Implements LCD.

7.121.2.2 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 perfoming alpha-blending (and tranparency keying) as specified Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 16-bits RGB565 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.121.2.3 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 perfoming alpha-blending (and tranparency keying) as specified. Performs a software blend if HAL does not support BLIT_COPY_WITH_ALPHA and alpha != 255. LCD16 supports source data formats: RGB565 and ARGB8888.

Parameters

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored in a format suitable for the selected display.
sourceFormat	The bitmap format used in the source data.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending (255 = solid, no blending)
hasTransparentPixels	If true, this data copy contains transparent pixels and require hardware support for that to be enabled.

Implements LCD.

7.121.2.4 blitCopyAlphaPerPixel()

```
static void blitCopyAlphaPerPixel (
            const uint16_t * sourceData16,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [static], [protected]
```

Blit a 2D source-array to the framebuffer performing alpha-blending per pixel as specified Performs always a software blend.

Parameters

sourceData16	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 8-bit RGBA2222 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.121.2.5 blitCopyARGB8888()

```
static void blitCopyARGB8888 (
            const uint32_t * sourceData,
            const Rect & source,
            const Rect & blitRect,
            uint8_t alpha )  [static], [protected]
```

Blit 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

sourceData	The source-array pointer (points to the beginning of the data). The sourceData must be stored as 32- bits ARGB8888 values.
source	The location and dimension of the source.
blitRect	A rectangle describing what region is to be drawn.
alpha	The alpha value to use for blending applied to the whole image (255 = solid, no blending)

7.121.2.6 copyFrameBufferRegionToMemory()

```
uint16_t * copyFrameBufferRegionToMemory (
              const Rect & visRegion,
              const Rect & absRegion,
              const BitmapId bitmapId )  [virtual]
```

Copies part of the framebuffer region to memory. The memory is given as BitmapId, which can be BITMAP_ANI↩ MATION_STORAGE. The two regions given are the visible region and the absolute region on screen. This is used to copy only a part of an area. This might be the case if a SnapshotWidget is placed inside a Container where parts of the SnapshowWidget is outside the area defined by the Container. The visible region must be completely inside the absolute region.

Note

There is only one instance of animation storage. The content of the animation storage outside the given region is undefined.

Parameters

visRegion	The visible region.
absRegion	The absolute region.
bitmapId	Identifier for the bitmap.

Returns

Null if it fails, else a pointer to the data in the given bitmap.

See also

blitCopy

Implements LCD.

7.121.2.7 drawGlyph()

```
void drawGlyph (
              uint16_t * wbuf8,
              Rect widgetArea,
              int16_t x,
              int16_t y,
              uint16_t offsetX,
              uint16_t offsetY,
```

```
            const Rect & invalidatedArea,
            const GlyphNode * glyph,
            const uint8_t * glyphData,
            uint8_t dataFormatA4,
            colortype color,
            uint8_t bitsPerPixel,
            uint8_t alpha,
            TextRotation rotation )  [protected], [virtual]
```

Private version of draw-glyph with explicit destination buffer pointer argument. For all parameters (except the buffer pointer) see the public function drawString().

Parameters

in	*wbuf8*	The destination (frame) buffer to draw to.
	widgetArea	The canvas to draw the glyph inside.
	x	Horizontal offset to start drawing the glyph.
	y	Vertical offset to start drawing the glyph.
	offsetX	Horizontal offset in the glyph to start drawing from.
	offsetY	Vertical offset in the glyph to start drawing from.
	invalidatedArea	The area to draw within.
	glyph	Specifications of the glyph to draw.
	glyphData	Data containing the actual glyph (dense format)
	dataFormatA4	The glyph is saved using ST A4 format.
	color	The color of the glyph.
	bitsPerPixel	Bit depth of the glyph.
	alpha	The transparency of the glyph.
	rotation	Rotation to do before drawing the glyph.

Implements LCD.

7.121.2.8 drawPartialBitmap()

```
void drawPartialBitmap (
            const Bitmap & bitmap,
            int16_t x,
            int16_t y,
            const Rect & rect,
            uint8_t alpha = 255,
            bool useOptimized = true )  [virtual]
```

Parameters

bitmap	The bitmap to draw.
x	The absolute x coordinate to place pixel (0, 0) on the screen.
y	The absolute y coordinate to place pixel (0, 0) on the screen.
rect	A rectangle describing what region of the bitmap is to be drawn.
alpha	Optional alpha value. Default is 255 (solid).
useOptimized	if false, do not attempt to substitute (parts of) this bitmap with faster fillrects.

Implements LCD.

7.121.2.9 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

dest	The description of where the texture is drawn - can be used to issue a draw off screen.
gradients	The gradients using in interpolation across the scan line.
leftEdge	The left edge of the scan line.
rightEdge	The right edge of the scan line.
texture	The texture.
absoluteRect	The containing rectangle in absolute coordinates.
dirtyAreaAbsolute	The dirty area in absolute coordinates.
renderVariant	The render variant - includes the algorithm and the pixel format.
alpha	The alpha.
subDivisionSize	The size of the subdivisions of the scan line. A value of 1 will give a completely perspective correct texture mapped scan line. A large value will give an affine texture mapped scan line.

Implements LCD.

7.121.2.10 fillRect()

```
void fillRect (
            const Rect & rect,
            colortype color,
            uint8_t alpha = 255 )  [virtual]
```

Draws a filled rectangle in the specified color.

Parameters

rect	The rectangle to draw in absolute coordinates.
color	The rectangle color.
alpha	The rectangle opacity (255=solid)

Implements LCD.

7.121.2.11 framebufferFormat()

```
Bitmap::BitmapFormat framebufferFormat ( ) const  [inline], [virtual]
```

Framebuffer format used by the display

Returns

Bitmap::RGBA2222.

Implements LCD.

7.121.2.12 framebufferStride()

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

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

Implements LCD.

7.121.2.13 getBlueColor()

```
uint8_t getBlueColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The blue part of the color.

Implements LCD.

7.121.2.14 getBlueFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getBlueFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The blue from color.

7.121.2.15 getColorFrom24BitRGB()

```
colortype getColorFrom24BitRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue ) const  [inline], [virtual]
```

Generates a color representation to be used on the LCD, based on 24 bit RGB values.

Parameters

red	Value of the red part (0-255).
green	Value of the green part (0-255).
blue	Value of the blue part (0-255).

Returns

The color representation depending on LCD color format.

Implements LCD.

7.121.2.16 getColorFromRGB()

```
FORCE_INLINE_FUNCTION static colortype getColorFromRGB (
            uint8_t red,
            uint8_t green,
            uint8_t blue )  [inline], [static]
```

Parameters

red	The red.
green	The green.
blue	The blue.

Returns

The color from RGB.

7.121.2.17 getFramebufferStride()

```
FORCE_INLINE_FUNCTION static uint16_t getFramebufferStride ( )  [inline], [static]
```

Framebuffer stride in bytes. The distance (in bytes) from the start of one framebuffer row, to the next.

Returns

The number of bytes in one framebuffer row.

7.121.2.18 getGreenColor()

```
uint8_t getGreenColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The 16 bit color value.

Returns

The green part of the color.

Implements LCD.

7.121.2.19 getGreenFromColor()

```
FORCE_INLINE_FUNCTION static getGreenFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The green from color.

7.121.2.20 getRedColor()

```
uint8_t getRedColor (
            colortype color ) const  [inline], [virtual]
```

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

color	The color value.

Returns

The red part of the color.

Implements LCD.

7.121.2.21 getRedFromColor()

```
FORCE_INLINE_FUNCTION static uint8_t getRedFromColor (
            colortype color )  [inline], [static]
```

Parameters

color	The color.

Returns

The red from color.

7.121.2.22 init()

```
void init ( )  [virtual]
```

Performs initialization.

Reimplemented from LCD.

7.121.2.23 nextLine()

```
static int nextLine (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next line.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next line.

7.121.2.24 nextPixel()

```
static int nextPixel (
            bool rotatedDisplay,
            TextRotation textRotation )  [static], [protected]
```

Find out how much to advance in the display buffer to get to the next pixel.

Parameters

rotatedDisplay	Is the display running in portrait mode?
textRotation	Rotation to perform.

Returns

How much to advance to get to the next pixel.

7.122 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.

7.122.1 Member Function Documentation

7.122.1.1 get()

```
static bool get (
            uint8_t nr )  [static]
```

Get state of a LED.

Parameters

nr	of the LED to toggle.

Returns

the state of the LED.

7.122.1.2 init()

```
static void init ( )  [static]
```

Perform configuration of IO pins.

7.122.1.3 off()

```
static void off (
            uint8_t nr )  [static]
```

Turn off a LED.

Parameters

nr	of the LED to switch off.

7.122.1.4 on()

```
static void on (
            uint8_t nr )  [static]
```

Turn on a LED.

Parameters

nr	of the LED to switch on.

7.122.1.5 toggle()

```
static void toggle (
            uint8_t nr )  [static]
```

Toggles a LED.

Parameters

nr	of the LED to toggle.

7.123 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.

- void updateLengthAndAngle (CWRUtil::Q5 length, CWRUtil::Q5 angle)

  Update the end point for this Line.

Additional Inherited Members

7.123.1 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

7.123.2 Member Enumeration Documentation

7.123.2.1 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.

7.123.3 Constructor & Destructor Documentation

7.123.3.1 Line()

```
Line ( )
```

Construct a new line.

7.123.4 Member Function Documentation

7.123.4.1 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

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Returns

true if it succeeds, false if it fails.

Implements CanvasWidget.

7.123.4.2 getEnd()

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

Gets the end coordinates for the line.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*x*	The x coordinate.
out	*y*	The y coordinate.

See also

setEnd
setLine

7.123.4.3 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

7.123.4.4 getLineWidth() [1/2]

```
template< typename T > void getLineWidth (
            T & width ) const  [inline]
```

Gets line width.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*width*	The line width.

See also

setLineWidth

7.123.4.5 getLineWidth() [2/2]

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

Gets line width.

Template Parameters

T	Generic type parameter, either int (default) or float.

Returns

The line width.

See also

setLineWidth

param [out] width The line width.

7.123.4.6 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.

7.123.4.7 getStart()

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

Gets the start coordinates for the line.

Template Parameters

T	Generic type parameter, either int or float.

Parameters

out	*x*	The x coordinate.
out	*y*	The y coordinate.

See also

setStart
setLine

7.123.4.8 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

precision	The new cap precision.

7.123.4.9 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

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the end point.
y	The y coordinate of the end point.

See also

updateEnd
getEnd

7.123.4.10 setEnd() [2/2]

```
void setEnd (
            CWRUtil::Q5 xQ5,
            CWRUtil::Q5 yQ5 )
```

Sets the end point for this Line.

Parameters

xQ5	The x coordinate of the end point in Q5 format.
yQ5	The y coordinate of the end point in Q5 format.

Note

The area containing the Line is not invalidated.

See also

updateEnd
getEnd

7.123.4.11 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

T	Generic type parameter, either int or float.

Parameters

x1	The x coordinate of the start point.
y1	The y coordinate of the start point.
x2	The x coordinate of the end point.
y2	The y coordinate of the end point.

See also

setStart
setEnd

7.123.4.12 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

lineEnding	The line ending style.

See also

LINE_ENDING_STYLE
getLineEndingStyle

7.123.4.13 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

T	Generic type parameter, either int or float.

Parameters

width	The width of the line measured in pixels.

See also

updateLineWidth

7.123.4.14 setLineWidth() [2/2]

```
void setLineWidth (
            CWRUtil::Q5 widthQ5 )  [inline]
```

Sets the width for this Line.

Parameters

widthQ5	The width of the line measured in pixels in Q5 format.

Note

The area containing the Line is not invalidated.

See also

updateLineWidth

7.123.4.15 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

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the start point.
y	The y coordinate of the start point.

See also

updateStart
getStart
setLine
setEnd

7.123.4.16 setStart() [2/2]

```
void setStart (
            CWRUtil::Q5 xQ5,
            CWRUtil::Q5 yQ5 )
```

Sets the start point for this Line.

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 not invalidated.

See also

updateStart
getStart
setLine
setEnd

7.123.4.17 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

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the end point.
y	The y coordinate of the end point.

7.123.4.18 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

xQ5	The x coordinate of the end point in Q5 format.
yQ5	The y coordinate of the end point in Q5 format.

Note

The area containing the Line is invalidated before and after the change.

7.123.4.19 updateLengthAndAngle()

```
void updateLengthAndAngle (
            CWRUtil::Q5 length,
            CWRUtil::Q5 angle )
```

Update the end point for this Line given the new length and angle. The rectangle that surrounds the line before and after will be invalidated.

Parameters

length	The new length of the line in Q5 format.
angle	The new angle of the line in Q5 format.

Note

The area containing the Line is invalidated before and after the change.

7.123.4.20 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

T	Generic type parameter, either int or float.

Parameters

width	The width of the line measured in pixels.

See also

setLineWidth

7.123.4.21 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

widthQ5	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

7.123.4.22 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

T	Generic type parameter, either int or float.

Parameters

x	The x coordinate of the start point.

Parameters

y	The y coordinate of the start point.

See also

setStart
updateEnd

7.123.4.23 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

7.124 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

7.124.1 Detailed Description

Using Line from canvas widgets, progress will be rendered as a line. The line does not need to 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 CanvasWidget↩ Rendere::setBuffer().

7.124.2 Constructor & Destructor Documentation

7.124.2.1 LineProgress()

```
LineProgress ( )
```

Default constructor.

7.124.2.2 ~LineProgress()

```
~LineProgress ( )  [virtual]
```

Destructor.

7.124.3 Member Function Documentation

7.124.3.1 getAlpha()

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

Gets the alpha.

Returns

The alpha.

See also

Line::getAlpha

7.124.3.2 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	*x*	The x coordinate.
out	*y*	The y coordinate.

7.124.3.3 getLineEndingStyle()

```
Line::LINE_ENDING_STYLE getLineEndingStyle ( ) const  [virtual]
```

Gets line ending style.

Returns

The line ending style.

7.124.3.4 getLineWidth()

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

Gets line width.

Returns

The line width.

7.124.3.5 getStart()

```
void getStart (
            int & x,
            int & y ) const  [virtual]
```

Gets the coordinates of the starting point of the line.

Parameters

out	*x*	The x coordinate.
out	*y*	The y coordinate.

7.124.3.6 setAlpha()

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

Sets the alpha for the line.

Parameters

alpha	The alpha.

See also

Line::setAlpha

7.124.3.7 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

x	The x coordinate of the end point.
y	The y coordinate of the end point.

See also

setStart

7.124.3.8 setLineEndingStyle()

```
void setLineEndingStyle (
            Line::LINE_ENDING_STYLE lineEndingStyle )  [virtual]
```

Sets line ending style.

Parameters

lineEndingStyle	The line ending style.

See also

Line::setLineEndingStyle

7.124.3.9 setLineWidth()

```
void setLineWidth (
            int width )  [virtual]
```

Sets line width.

Parameters

width	The width.

See also

Line::setLineWidth

7.124.3.10 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	*painter*	The painter.

7.124.3.11 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

x	The x coordinate.
y	The y coordinate.
width	The width of the line progress indicator.
height	The height of the line progress indicator.

Reimplemented from AbstractProgressIndicator.

7.124.3.12 setStart()

```
void setStart (
             int x,
             int y )  [virtual]
```

Sets a starting point for the line.

Parameters

x	The x coordinate of the start point.
y	The y coordinate of the start point.

See also

setEnd

7.124.3.13 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

value	The value.

Reimplemented from AbstractProgressIndicator.

7.125 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

7.125.1 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

7.125.2 Constructor & Destructor Documentation

7.125.2.1 ListLayout()

```
ListLayout (
            const Direction d = SOUTH )  [inline]
```

Constructor. Constructs a ListLayout instance that arranges the added elements in the specified Direction.

Parameters

d	The direction to grow in when adding children.

7.125.2.2 ~ListLayout()

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

Destructor.

7.125.3 Member Function Documentation

7.125.3.1 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	*d*	The Drawable to add.

Reimplemented from Container.

7.125.3.2 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()

7.125.3.3 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_LISTLAYOUT.

Reimplemented from Container.

7.125.3.4 insert()

```
void insert (
            Drawable * previousElement,
            Drawable & d )  [virtual]
```

Inserts a Drawable.

Parameters

in	*previousElement*	The element to insert the new element after.
in	*d*	The element to insert.

Reimplemented from Container.

7.125.3.5 remove()

```
void remove (
            Drawable & d )  [virtual]
```

Removes a Drawable. Safe to call even if drawable has not been added.

Parameters

in	*d*	The drawable to remove.

Reimplemented from Container.

7.125.3.6 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.

7.125.3.7 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

d	The new Direction to grow in when added children (either SOUTH or EAST).

See also

getDirection()

7.126 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.

7.126.1 Detailed Description

This implements a simple lock-free FIFO queue (single producer, single consumer)

See also

DMA_Queue

7.126.2 Constructor & Destructor Documentation

7.126.2.1 LockFreeDMA_Queue()

```
LockFreeDMA_Queue (
            BlitOp * mem,
            atomic_t n )
```

Constructs a lockfree queue.

Parameters

out	*mem*	Pointer to the memory used by the queue to store elements.
	n	Number of elements the memory provided can contain.

7.126.3 Member Function Documentation

7.126.3.1 first()

```
const BlitOp * first ( )  [protected], [virtual]
```

Returns

the first blitop.

Implements DMA_Queue.

7.126.3.2 isEmpty()

```
bool isEmpty ( )  [virtual]
```

Query if this object is empty.

Returns

true if empty, false if not.

Implements DMA_Queue.

7.126.3.3 isFull()

```
bool isFull ( )  [virtual]
```

Query if this object is full.

Returns

true if full, false if not.

Implements DMA_Queue.

7.126.3.4 pop()

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

Removes the top-of-stack object.

Implements DMA_Queue.

7.126.3.5 pushCopyOf()

```
void pushCopyOf (
            const BlitOp & op )  [virtual]
```

Push copy of the given operation.

Parameters

op	The operation.

Implements DMA_Queue.

7.127 ManyBlockAllocator< block_size, blocks, bytes_pr_pixel > Class Template Reference

This class is partial framebuffer allocator using multiple blocks.

```
#include <touchgfx/hal/FrameBufferAllocator.hpp>
```

Public Member Functions

- virtual uint16_t allocateBlock (const uint16_t x, const uint16_t y, const uint16_t width, const uint16_t height, uint8_t **block)

  Allocates a framebuffer block.
- virtual void markBlockReadyForTransfer ()

  Marks a previously allocated block as ready to be transferred to the LCD.
- virtual bool hasBlockReadyForTransfer ()

  Check if a block is ready for transfer to the LCD.
- virtual const uint8_t * getBlockForTransfer (Rect &rect)

  Get the block ready for transfer.
- virtual void freeBlockAfterTransfer ()

  Free a block after transfer to the LCD.

7.127.1 Detailed Description

template<uint32_t block_size, int32_t blocks, uint32_t bytes_pr_pixel>
class touchgfx::ManyBlockAllocator< block_size, blocks, bytes_pr_pixel >

This class is partial framebuffer allocator using multiple blocks. New buffers can be allocated until no free blocks are available. After transfer to LCD, a block is queued for allocation again.

See also

SingleBlockAllocator

7.127.2 Member Function Documentation

7.127.2.1 allocateBlock()

```
uint16_t allocateBlock (
            const uint16_t x,
            const uint16_t y,
            const uint16_t width,
            const uint16_t height,
            uint8_t ** block )  [inline], [virtual]
```

Allocates a framebuffer block. The block will have at least the width requested. The height of the allocated block can be lower than requested if not enough memory is available.

Parameters

x	The absolute x coordinate of the block on the screen.
y	The absolute y coordinate of the block on the screen.
width	The width of the block.
height	The height of the block.
block	Pointer to pointer to return the block address in.

Returns

The height of the allocated block.

Implements FrameBufferAllocator.

7.127.2.2 freeBlockAfterTransfer()

```
void freeBlockAfterTransfer ( )  [inline], [virtual]
```

Marks a previously allocated block as transferred and ready to reuse.

Implements FrameBufferAllocator.

7.127.2.3 getBlockForTransfer()

```
const uint8_t * getBlockForTransfer (
            Rect & rect )  [inline], [virtual]
```

Get the block ready for transfer.

Parameters

rect	Reference to rect to write block x, y, width, and height.

Returns

Returns the address of the block ready for transfer.

Implements FrameBufferAllocator.

7.127.2.4 hasBlockReadyForTransfer()

```
bool hasBlockReadyForTransfer ( )  [inline], [virtual]
```

Check if a block is ready for transfer to the LCD.

Returns

True if a block is ready for transfer.

Implements FrameBufferAllocator.

7.127.2.5 markBlockReadyForTransfer()

```
void markBlockReadyForTransfer ( )  [inline], [virtual]
```

Marks a previously allocated block as ready to be transferred to the LCD.

Implements FrameBufferAllocator.

7.128 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].

7.128.1 Detailed Description

This class represents row major 4x4 homogeneous matrices.

7.128.2 Constructor & Destructor Documentation

7.128.2.1 Matrix4x4()

```
Matrix4x4 ( )
```

Default constructor.

7.128.3 Member Function Documentation

7.128.3.1 concatenateXRotation()

```
Matrix4x4 & concatenateXRotation (
            float radians )
```

Concatenate x coordinate rotation.

Parameters

radians	The radians.

Returns

A matrix_4x4&

7.128.3.2 concatenateXScale()

```
Matrix4x4 & concatenateXScale (
            float distance )
```

Concatenate x coordinate scale.

Parameters

distance	The distance.

Returns

A matrix_4x4&

7.128.3.3 concatenateXTranslation()

```
Matrix4x4 & concatenateXTranslation (
            float distance )
```

Concatenate x coordinate translation.

Parameters

distance	The distance.

Returns

A matrix_4x4&

7.128.3.4 concatenateYRotation()

```
Matrix4x4 & concatenateYRotation (
            float radians )
```

Concatenate y coordinate rotation.

Parameters

radians	The radians.

Returns

A matrix_4x4&

7.128.3.5 concatenateYScale()

```
Matrix4x4 & concatenateYScale (
            float distance )
```

Concatenate y coordinate scale.

Parameters

distance	The distance.

Returns

A matrix_4x4&

7.128.3.6 concatenateYTranslation()

```
Matrix4x4 & concatenateYTranslation (
            float distance )
```

Concatenate y coordinate translation.

Parameters

distance	The distance.

Returns

A matrix_4x4&

7.128.3.7 concatenateZRotation()

```
Matrix4x4 & concatenateZRotation (
            float radians )
```

Concatenate z coordinate rotation.

Parameters

radians	The radians.

Returns

A matrix_4x4&

7.128.3.8 concatenateZScale()

```
Matrix4x4 & concatenateZScale (
            float distance )
```

Concatenate z coordinate scale.

Parameters

distance	The distance.

Returns

A matrix_4x4&

7.128.3.9 concatenateZTranslation()

```
Matrix4x4 & concatenateZTranslation (
            float distance )
```

Concatenate z coordinate translation.

Parameters

distance	The distance.

Returns

A matrix_4x4&

7.128.3.10 getElement()

```
float getElement (
            int row,
            int column ) const  [inline]
```

Gets an element.

Parameters

row	The row.
column	The column.

Returns

The element.

7.128.3.11 identity()

```
static Matrix4x4 identity ( )  [inline], [static]
```

Gets the identity. Instead of using "Matrix4x4::identity()" consider using "Matrix4x4()" instead.

Returns

A matrix_4x4.

7.128.3.12 setElement()

```
Matrix4x4 setElement (
            int row,
```

```
            int column,
            float value )  [inline]
```

Sets an element.

Parameters

row	The row.
column	The column.
value	The value.

Returns

A matrix_4x4&

7.128.3.13 setViewDistance()

```
void setViewDistance (
            float distance )
```

Sets view distance.

Parameters

distance	The distance.

7.129 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.

7.129.1 Detailed Description

Interface for instrumenting processors to measure MCU load via measured CPU cycles.

7.129.2 Constructor & Destructor Documentation

7.129.2.1 MCUInstrumentation()

```
MCUInstrumentation ( )  [inline]
```

Constructor. Initializes members.

7.129.2.2 ~MCUInstrumentation()

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

Virtual destructor.

7.129.3 Member Function Documentation

7.129.3.1 getCCConsumed()

```
uint32_t getCCConsumed ( )  [inline], [virtual]
```

Gets number of consumed clock cycles.

Returns

clock cycles.

7.129.3.2 getCPUCycles()

```
unsigned int getCPUCycles (
            void  )  [pure virtual]
```

Gets CPU cycles from register.

Returns

CPU cycles.

7.129.3.3 getElapsedUS()

```
unsigned int getElapsedUS (
            unsigned int start,
            unsigned int now,
            unsigned int clockfrequency )  [pure virtual]
```

Gets elapsed microseconds based on clock frequency.

Parameters

start	Start time.
now	Current time.
clockfrequency	Clock frequency of the system expressed in MHz.

Returns

Elapsed microseconds start and now.

7.129.3.4 init()

```
void init ( )  [pure virtual]
```

Initialize.

7.129.3.5 setCCConsumed()

```
void setCCConsumed (
            uint32_t val )  [inline], [virtual]
```

Sets number of consumed clock cycles.

Parameters

val	number of clock cycles.

7.129.3.6 setMCUActive()

```
void setMCUActive (
            bool active )  [inline], [virtual]
```

Sets MCU activity high.

Parameters

active	if True, inactive otherwise.

7.130 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 getBackgroundHeight () 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

7.130.1 Detailed Description

Container for displaying a modal window and hijacking touch event to underlaying 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.

7.130.2 Constructor & Destructor Documentation

7.130.2.1 ModalWindow()

```
ModalWindow ( )
```

Default constructor.

7.130.2.2 ~ModalWindow()

```
~ModalWindow ( )  [virtual]
```

Destructor.

7.130.3 Member Function Documentation

7.130.3.1 add()

```
void add (
            Drawable & d )  [virtual]
```

Adds a drawable to the ModalWindow. The drawable will be placed relative to the background image.

Parameters

in	*d*	The drawable to add.

Reimplemented from Container.

7.130.3.2 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.

7.130.3.3 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.

7.130.3.4 getShadeAlpha()

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

Gets the alpha value of the background shade.

Returns

The background shades alpha.

7.130.3.5 getShadeColor()

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

Gets the color of the background shade.

Returns

The color of the background shade.

7.130.3.6 hide()

```
void hide ( )  [virtual]
```

Make the ModalWindow invisible.

7.130.3.7 isShowing()

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

Query if this ModalWindow is showing.

Returns

true if showing, false if not.

7.130.3.8 remove()

```
void remove (
            Drawable & d )  [virtual]
```

Removes the drawable from the ModalWindow.

Parameters

in	*d*	The drawable to remove.

Reimplemented from Container.

7.130.3.9 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

bmpId	Identifier for the background bitmap.

7.130.3.10 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

bmpId	Identifier for the bitmap.
backgroundX	The background x coordinate.
backgroundY	The background y coordinate.

7.130.3.11 setShadeAlpha()

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

Sets the alpha value of the background shade. Default = 96.

Parameters

alpha	The new alpha.

7.130.3.12 setShadeColor()

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

Sets the color of the background shade. Default = Black.

Parameters

color	The new color.

7.130.3.13 show()

```
void show ( ) [virtual]
```

Make the ModalWindow visible.

7.131 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.

7.131.1 Detailed Description

template<class T>
class touchgfx::MoveAnimator< T >

A MoveAnimator makes the template class T able to animate a movement from its current position to a specified end position. The movement in both the X and Y direction can be described by supplying EasingEquations. The MoveAnimator performs a callback when the animation has finished.

This mixin can be used on any Drawable.

Template Parameters

T	Specifies the type should have the move animation capability.

7.131.2 Constructor & Destructor Documentation

7.131.2.1 MoveAnimator()

```
MoveAnimator ( )  [inline]
```

Default constructor. Creates and initialize the MoveAnimator.

7.131.2.2 ~MoveAnimator()

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

Destructor. Destroys the MoveAnimator.

7.131.3 Member Function Documentation

7.131.3.1 cancelMoveAnimation()

```
void cancelMoveAnimation ( )  [inline]
```

Cancel move animation.

7.131.3.2 clearMoveAnimationEndedAction()

```
void clearMoveAnimationEndedAction ( )  [inline]
```

Clears the move animation ended action previously set by setMoveAnimationEndedAction.

See also

setMoveAnimationEndedAction

7.131.3.3 getMoveAnimationDelay()

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

Gets the current animation delay.

Returns

The current animation delay.

7.131.3.4 handleTickEvent()

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

The tick handler that handles the actual animation steps.

7.131.3.5 isMoveAnimationRunning()

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

Gets whether or not the move animation is running.

Returns

true if the move animation is running.

7.131.3.6 isRunning()

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

Gets whether or not the move animation is running.

Returns

true if the move animation is running.

7.131.3.7 nextMoveAnimationStep()

```
void nextMoveAnimationStep ( )  [inline], [protected]
```

Execute next step in move animation and stop the timer if necessary.

7.131.3.8 setMoveAnimationDelay()

```
void setMoveAnimationDelay (
            uint16_t delay )  [inline], [virtual]
```

Sets a delay on animations done by the MoveAnimator.

Parameters

delay	The delay in ticks.

7.131.3.9 setMoveAnimationEndedAction()

```
void setMoveAnimationEndedAction (
            GenericCallback< const MoveAnimator< T > & > & callback )  [inline]
```

Associates an action to be performed when the animation ends.

Parameters

callback	The callback to be executed. The callback will be given a reference to the MoveAnimator.

See also

GenericCallback

7.131.3.10 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

endX	The X position of T at animation end. Relative to the container or view that holds T.
endY	The Y position of T at animation end. Relative to the container or view that holds T.
duration	The duration of the animation measured in ticks.
xProgressionEquation	The equation that describes the development of the X position during the animation. Default = EasingEquations::linearEaseNone.
yProgressionEquation	The equation that describes the development of the Y position during the animation. Default = EasingEquations::linearEaseNone.

7.132 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

7.132.1 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

7.132.2 Constructor & Destructor Documentation

7.132.2.1 MVPApplication()

```
MVPApplication ( )  [inline]
```

Default constructor.

7.132.2.2 ~MVPApplication()

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

Destructor.

7.132.3 Member Function Documentation

7.132.3.1 evaluatePendingScreenTransition()

```
void evaluatePendingScreenTransition ( )  [inline], [protected]
```

Evaluates the pending Callback instances. If a callback is valid, it is executed and a Screen transition is executed.

7.132.3.2 handlePendingScreenTransition()

```
void handlePendingScreenTransition ( )  [inline], [virtual]
```

Delegates the work to evaluatePendingScreenTransition()

Reimplemented from Application.

7.133 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.

7.133.1 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.

7.133.2 Constructor & Destructor Documentation

7.133.2.1 MVPHeap()

```
MVPHeap (
            AbstractPartition & pres,
            AbstractPartition & scr,
            AbstractPartition & tra,
            MVPApplication & app )  [inline]
```

Constructs an MVPHeap.

Parameters

in	*pres*	A memory partition containing enough memory to hold the largest presenter.
in	*scr*	A memory partition containing enough memory to hold the largest view.
in	*tra*	A memory partition containing enough memory to hold the largest transition.
in	*app*	A reference to the MVPApplication instance.

7.133.2.2 ~MVPHeap()

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

Destructor.

7.134 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

7.134.1 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

7.134.2 Constructor & Destructor Documentation

7.134.2.1 NoDMA()

```
NoDMA ( )  [inline]
```

Constructs a NoDMA object, with a queue of 1 element.

7.134.3 Member Function Documentation

7.134.3.1 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.

7.134.3.2 getBlitCaps()

```
BlitOperations getBlitCaps ( )  [inline], [virtual]
```

No blit operations supported by this DMA implementation.

Returns

Zero (no blit ops supported).

Implements DMA_Interface.

7.134.3.3 setupDataCopy()

```
void setupDataCopy (
            const BlitOp & blitOp )  [inline], [virtual]
```

Asserts if used.

Parameters

blitOp	The blit operation to be performed by this DMA instance.

Implements DMA_Interface.

7.134.3.4 setupDataFill()

```
void setupDataFill (
            const BlitOp & blitOp )  [inline], [virtual]
```

Asserts if used.

Parameters

blitOp	The blit operation to be performed by this DMA instance.

Implements DMA_Interface.

7.134.3.5 signalDMAInterrupt()

```
void signalDMAInterrupt ( )  [inline], [virtual]
```

Does nothing.

Implements DMA_Interface.

7.135 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.

7.135.1 Detailed Description

Empty TouchController implementation which does nothing. Use this if your display does not have touch input capabilities.

7.135.2 Constructor & Destructor Documentation

7.135.2.1 ~NoTouchController()

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

Destructor.

7.135.3 Member Function Documentation

7.135.3.1 init()

```
void init ( )  [inline], [virtual]
```

Empty initialization.

Implements TouchController.

7.135.3.2 sampleTouch()

```
bool sampleTouch (
            int32_t & x,
            int32_t & y )  [inline], [virtual]
```

Parameters

out	*x*	unused.
out	*y*	unused.

Returns

false.

Implements TouchController.

7.136 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

7.136.1 Detailed Description

The most simple Transition without any visual effects.

See also

Transition

7.136.2 Constructor & Destructor Documentation

7.136.2.1 NoTransition()

`NoTransition ( ) [inline]`

Default constructor.

7.136.2.2 ~NoTransition()

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

Destructor.

7.136.3 Member Function Documentation

7.136.3.1 handleTickEvent()

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

Indicates that the transition is done after the first tick.

Reimplemented from Transition.

7.137 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.

7.137.1 Detailed Description

This class specifies OS wrappers for dealing with the frame buffer semaphore and the VSYNC signal.

7.137.2 Member Function Documentation

7.137.2.1 giveFrameBufferSemaphore()

`static void giveFrameBufferSemaphore ( ) [static]`

Release the frame buffer semaphore.

7.137.2.2 giveFrameBufferSemaphoreFromISR()

`static void giveFrameBufferSemaphoreFromISR ( ) [static]`

Release the frame buffer semaphore in a way that is safe in interrupt context. Called from ISR.

7.137.2.3 initialize()

`static void initialize ( ) [static]`

Initialize frame buffer semaphore and queue/mutex for VSYNC signal.

7.137.2.4 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.

7.137.2.5 takeFrameBufferSemaphore()

```
static void takeFrameBufferSemaphore ( )  [static]
```

Take the frame buffer semaphore. Blocks until semaphore is available.

7.137.2.6 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_SING↩ LE_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

ms	The number of milliseconds to sleep

See also

HAL::setFrameRefreshStrategy(FrameRefreshStrategy s)
HAL::registerTaskDelayFunction(void (∗delayF)(uint16_t))

7.137.2.7 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 the taken the semaphore, it only serves to make sure that the semaphore is taken by someone.

7.137.2.8 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.

7.138 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.

7.138.1 Detailed Description

An internal class that implements the main rasterization algorithm. Used in the Rasterizer. Should not be used direcly.

7.138.2 Member Typedef Documentation

7.138.2.1 OutlineFlags_t

```
unsigned int OutlineFlags_t
```

Defines an alias representing the outline flags.

7.138.3 Constructor & Destructor Documentation

7.138.3.1 Outline()

```
Outline ( )
```

Default constructor.

7.138.3.2 ~Outline()

```
~Outline ( )  [virtual]
```

Destructor.

7.138.4 Member Function Documentation

7.138.4.1 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.

7.138.4.2 getNumCells()

```
unsigned getNumCells ( ) const  [inline]
```

Gets number cells registered in the current drawn path for the Outline.

Returns

The number of cells.

7.138.4.3 lineTo()

```
void lineTo (
             int x,
             int y )
```

Create a line from the current virtual pen coordinate to the given coordinate creating an Outline.

Parameters

x	The x coordinate.
y	The y coordinate.

7.138.4.4 moveTo()

```
void moveTo (
             int x,
             int y )
```

Move a virtual pen to the specified coordinate.

Parameters

x	The x coordinate.
y	The y coordinate.

7.138.4.5 reset()

```
void reset ( )
```

Resets this object. This implies removing the current Cell objects and preparing for a new Outline.

7.138.4.6 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 of higher than the given y.

Parameters

y	The max y coordinate to render for the Outline.

7.138.4.7 wasOutlineTooComplex()

```
bool wasOutlineTooComplex ( )  [inline]
```

Determines if there was enough memory to register the entire outline, of if the outline was too complex.

Returns

false if the buffer for Outline Cell objects was too small.

7.139 PainterABGR2222 Class Reference

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

`#include <touchgfx/widgets/canvas/PainterABGR2222.hpp>`

Public Member Functions

- PainterABGR2222 (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 painterColor
  The color.
- uint8_t painterRed
  The red part of the color, scaled up to [0..255].
- uint8_t painterGreen
  The green part of the color, scaled up to [0..255].
- uint8_t painterBlue
  The blue part of the color, scaled up to [0..255].
- uint8_t painterAlpha
  The alpha value.

Additional Inherited Members

7.139.1 Detailed Description

The PainterABGR2222 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

7.139.2 Constructor & Destructor Documentation

7.139.2.1 PainterABGR2222()

```
PainterABGR2222 (
            colortype color = 0,
            uint8_t alpha = 255 )
```

Constructor.

Parameters

color	the color.
alpha	the alpha.

7.139.3 Member Function Documentation

7.139.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.139.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.139.3.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterABGR2222.

7.139.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterABGR2222.

7.139.3.5 setAlpha()

```
void setAlpha (
            uint8_t alpha )
```

Sets an alpha value for the painter.

Parameters

alpha	The alpha value to use.

7.139.3.6 setColor()

```
void setColor (
            colortype color,
            uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

7.140 PainterABGR2222Bitmap Class Reference

A Painter that will paint using a bitmap.

`#include <touchgfx/widgets/canvas/PainterABGR2222Bitmap.hpp>`

Public Member Functions

- PainterABGR2222Bitmap (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 uint8_t * bitmapABGR2222Pointer

  Pointer to the bitmap (ABGR2222)
- 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

7.140.1 Detailed Description

PainterABGR2222Bitmap 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

7.140.2 Constructor & Destructor Documentation

7.140.2.1 PainterABGR2222Bitmap()

```
PainterABGR2222Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.140.3 Member Function Documentation

7.140.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.140.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterABGR2222.

7.140.3.3 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 AbstractPainterABGR2222.

7.140.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterABGR2222.

7.140.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.140.3.6 setBitmap()

```
void setBitmap (
            const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.141 PainterARGB2222 Class Reference

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

`#include <touchgfx/widgets/canvas/PainterARGB2222.hpp>`

Public Member Functions

- PainterARGB2222 (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 painterColor

  The color.
- uint8_t painterRed

  The red part of the color, scaled up to [0..255].
- uint8_t painterGreen

  The green part of the color, scaled up to [0..255].
- uint8_t painterBlue

  The blue part of the color, scaled up to [0..255].
- uint8_t painterAlpha

  The alpha value.

Additional Inherited Members

7.141.1 Detailed Description

The PainterARGB2222 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

7.141.2 Constructor & Destructor Documentation

7.141.2.1 PainterARGB2222()

```
PainterARGB2222 (
            colortype color = 0,
            uint8_t alpha = 255 )
```

Constructor.

Parameters

color	the color.
alpha	the alpha.

7.141.3 Member Function Documentation

7.141.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.141.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.141.3.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterARGB2222.

7.141.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterARGB2222.

7.141.3.5 setAlpha()

```
void setAlpha (
            uint8_t alpha )
```

Sets an alpha value for the painter.

Parameters

alpha	The alpha value to use.

7.141.3.6 setColor()

```
void setColor (
            colortype color,
            uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

7.142 PainterARGB2222Bitmap Class Reference

A Painter that will paint using a bitmap.

`#include <touchgfx/widgets/canvas/PainterARGB2222Bitmap.hpp>`

Public Member Functions

- PainterARGB2222Bitmap (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 uint8_t * bitmapARGB2222Pointer

  Pointer to the bitmap (ARGB2222)
- 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

7.142.1 Detailed Description

PainterARGB2222Bitmap 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

7.142.2 Constructor & Destructor Documentation

7.142.2.1 PainterARGB2222Bitmap()

```
PainterARGB2222Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.142.3 Member Function Documentation

7.142.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.142.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterARGB2222.

7.142.3.3 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 AbstractPainterARGB2222.

7.142.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterARGB2222.

7.142.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.142.3.6 setBitmap()

```
void setBitmap (
             const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.143 PainterARGB8888 Class Reference

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

`#include <touchgfx/widgets/canvas/PainterARGB8888.hpp>`

Public Member Functions

- PainterARGB8888 (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.

Additional Inherited Members

7.143.1 Detailed Description

The PainterARGB8888 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

7.143.2 Constructor & Destructor Documentation

7.143.2.1 PainterARGB8888()

```
PainterARGB8888 (
            colortype color = 0,
            uint8_t alpha = 255 )
```

Constructor.

Parameters

color	the color.
alpha	the alpha.

7.143.3 Member Function Documentation

7.143.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.143.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.143.3.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterARGB8888.

7.143.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterARGB8888.

7.143.3.5 setAlpha()

```
void setAlpha (
             uint8_t alpha )
```

Sets an alpha value for the painter.

Parameters

alpha	The alpha value to use.

7.143.3.6 setColor()

```
void setColor (
```

```
colortype color,
uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

7.144 PainterARGB8888Bitmap Class Reference

A Painter that will paint using a bitmap.

`#include <touchgfx/widgets/canvas/PainterARGB8888Bitmap.hpp>`

Public Member Functions

- PainterARGB8888Bitmap (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 * 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.

Additional Inherited Members

7.144.1 Detailed Description

PainterARGB8888Bitmap 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

7.144.2 Constructor & Destructor Documentation

7.144.2.1 PainterARGB8888Bitmap()

```
PainterARGB8888Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.144.3 Member Function Documentation

7.144.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.144.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterARGB8888.

7.144.3.3 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 AbstractPainterARGB8888.

7.144.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterARGB8888.

7.144.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.144.3.6 setBitmap()

```
void setBitmap (
            const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.145 PainterARGB8888L8Bitmap Class Reference

A Painter that will paint using a bitmap.

`#include <touchgfx/widgets/canvas/PainterARGB8888L8Bitmap.hpp>`

Public Member Functions

- PainterARGB8888L8Bitmap (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 uint8_t * bitmapPointer

  Pointer to the bitmap (L8)
- const uint8_t * bitmapExtraPointer

  Pointer to the CLUT (L8)
- 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.

Additional Inherited Members

7.145.1 Detailed Description

PainterARGB8888L8Bitmap 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

7.145.2 Constructor & Destructor Documentation

7.145.2.1 PainterARGB8888L8Bitmap()

```
PainterARGB8888L8Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.145.3 Member Function Documentation

7.145.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.145.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterARGB8888.

7.145.3.3 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 AbstractPainterARGB8888.

7.145.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterARGB8888.

7.145.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.145.3.6 setBitmap()

```
void setBitmap (
            const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.146 PainterBGRA2222 Class Reference

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

```
#include <touchgfx/widgets/canvas/PainterBGRA2222.hpp>
```

Public Member Functions

- PainterBGRA2222 (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 painterColor

  The color.
- uint8_t painterRed

  The red part of the color, scaled up to [0..255].
- uint8_t painterGreen

  The green part of the color, scaled up to [0..255].
- uint8_t painterBlue

  The blue part of the color, scaled up to [0..255].
- uint8_t painterAlpha

  The alpha value.

Additional Inherited Members

7.146.1 Detailed Description

The PainterBGRA2222 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

7.146.2 Constructor & Destructor Documentation

7.146.2.1 PainterBGRA2222()

```
PainterBGRA2222 (
            colortype color = 0,
            uint8_t alpha = 255 )
```

Constructor.

Parameters

color	the color.
alpha	the alpha.

7.146.3 Member Function Documentation

7.146.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.146.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.146.3.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).

Parameters

	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterBGRA2222.

7.146.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterBGRA2222.

7.146.3.5 setAlpha()

```
void setAlpha (
             uint8_t alpha )
```

Sets an alpha value for the painter.

Parameters

alpha	The alpha value to use.

7.146.3.6 setColor()

```
void setColor (
             colortype color,
             uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

7.147 PainterBGRA2222Bitmap Class Reference

A Painter that will paint using a bitmap.

`#include <touchgfx/widgets/canvas/PainterBGRA2222Bitmap.hpp>`

Public Member Functions

- PainterBGRA2222Bitmap (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 uint8_t * bitmapBGRA2222Pointer

  Pointer to the bitmap (BGRA2222)
- 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

7.147.1 Detailed Description

PainterBGRA2222Bitmap 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

7.147.2 Constructor & Destructor Documentation

7.147.2.1 PainterBGRA2222Bitmap()

```
PainterBGRA2222Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.147.3 Member Function Documentation

7.147.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.147.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterBGRA2222.

7.147.3.3 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 AbstractPainterBGRA2222.

7.147.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterBGRA2222.

7.147.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.147.3.6 setBitmap()

```
void setBitmap (
            const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.148 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.

Additional Inherited Members

7.148.1 Detailed Description

PainterBW is used for drawing one 1bpp displays. The color is either on or off No transparency is supported.

See also

AbstractPainter

7.148.2 Member Function Documentation

7.148.2.1 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

red	The red color.
green	The green color.
blue	The blue color.

Returns

1 (white) if the brightness of the RGB color is above 50% and 0 (black) otherwise.

7.148.2.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.148.2.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterBW.

7.148.2.4 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.

7.148.2.5 setColor()

```
void setColor (
            colortype color )
```

Sets color to use when drawing the CanvasWidget.

Parameters

color	The color, 0=black, otherwise white.

7.149 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.

Additional Inherited Members

7.149.1 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

7.149.2 Constructor & Destructor Documentation

7.149.2.1 PainterBWBitmap()

```
PainterBWBitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID) )
```

Constructor.

Parameters

bmp	The bitmap.

7.149.3 Member Function Documentation

7.149.3.1 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterBW.

7.149.3.2 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.

7.149.3.3 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.

7.149.3.4 setBitmap()

```
void setBitmap (
            const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.150 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.

Additional Inherited Members

7.150.1 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

7.150.2 Constructor & Destructor Documentation

7.150.2.1 PainterGRAY2()

```
PainterGRAY2 (
             colortype color = 0,
             uint8_t alpha = 255 )
```

Constructor.

Parameters

color	the color.
alpha	the alpha.

7.150.3 Member Function Documentation

7.150.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.150.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.150.3.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterGRAY2.

7.150.3.4 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	*gray*	The gray (0-15).
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterGRAY2.

7.150.3.5 setAlpha()

```
void setAlpha (
```

```
            uint8_t alpha )
```

Sets an alpha value for the painter.

Parameters

alpha	The alpha value to use.

7.150.3.6 setColor()

```
void setColor (
            colortype color,
            uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

7.151 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.

Additional Inherited Members

7.151.1 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

7.151.2 Constructor & Destructor Documentation

7.151.2.1 PainterGRAY2Bitmap()

```
PainterGRAY2Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.151.3 Member Function Documentation

7.151.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.151.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterGRAY2.

7.151.3.3 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.

7.151.3.4 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	*gray*	The gray (0-15).
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterGRAY2.

7.151.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.151.3.6 setBitmap()

```
void setBitmap (
            const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.152 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.

Additional Inherited Members

7.152.1 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

7.152.2 Constructor & Destructor Documentation

7.152.2.1 PainterGRAY4()

```
PainterGRAY4 (
             colortype color = 0,
             uint8_t alpha = 255 )
```

Constructor.

Parameters

color	the color.
alpha	the alpha.

7.152.3 Member Function Documentation

7.152.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.152.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.152.3.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterGRAY4.

7.152.3.4 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	*gray*	The gray (0-15).
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterGRAY4.

7.152.3.5 setAlpha()

```
void setAlpha (
             uint8_t alpha )
```

Sets an alpha value for the painter.

Parameters

alpha	The alpha value to use.

7.152.3.6 setColor()

```
void setColor (
             colortype color,
             uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

7.153 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.

Additional Inherited Members

7.153.1 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

7.153.2 Constructor & Destructor Documentation

7.153.2.1 PainterGRAY4Bitmap()

```
PainterGRAY4Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.153.3 Member Function Documentation

7.153.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.153.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterGRAY4.

7.153.3.3 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.

7.153.3.4 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	*gray*	The gray (0-15).
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterGRAY4.

7.153.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.153.3.6 setBitmap()

```
void setBitmap (
             const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.154 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

7.154.1 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

7.154.2 Constructor & Destructor Documentation

7.154.2.1 PainterRGB565()

```
PainterRGB565 (
             colortype color = 0,
             uint8_t alpha = 255 )
```

Constructor.

Parameters

color	the color.
alpha	the alpha.

7.154.3 Member Function Documentation

7.154.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.154.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.154.3.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterRGB565.

7.154.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB565.

7.154.3.5 setAlpha()

```
void setAlpha (
            uint8_t alpha )
```

Sets an alpha value for the painter.

Parameters

alpha	The alpha value to use.

7.154.3.6 setColor()

```
void setColor (
            colortype color,
            uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

7.155 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

7.155.1 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

7.155.2 Constructor & Destructor Documentation

7.155.2.1 PainterRGB565Bitmap()

```
PainterRGB565Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.155.3 Member Function Documentation

7.155.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.155.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterRGB565.

7.155.3.3 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.

7.155.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB565.

7.155.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.155.3.6 setBitmap()

```
void setBitmap (
            const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.156 PainterRGB565L8Bitmap Class Reference

A Painter that will paint using a bitmap.

`#include <touchgfx/widgets/canvas/PainterRGB565L8Bitmap.hpp>`

Public Member Functions

- PainterRGB565L8Bitmap (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 uint8_t * bitmapPointer

  Pointer to the bitmap (L8)
- const uint8_t * bitmapExtraPointer

  Pointer to the bitmap alpha data for RGB565 / CLUT for L8.
- 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

7.156.1 Detailed Description

PainterRGB565L8Bitmap 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

7.156.2 Constructor & Destructor Documentation

7.156.2.1 PainterRGB565L8Bitmap()

```
PainterRGB565L8Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.156.3 Member Function Documentation

7.156.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.156.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterRGB565.

7.156.3.3 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.

7.156.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB565.

7.156.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.156.3.6 setBitmap()

```
void setBitmap (
             const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.157 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.

Additional Inherited Members

7.157.1 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

7.157.2 Constructor & Destructor Documentation

7.157.2.1 PainterRGB888()

```
PainterRGB888 (
            colortype color = 0,
            uint8_t alpha = 255 )
```

Constructor.

Parameters

color	the color.
alpha	the alpha.

7.157.3 Member Function Documentation

7.157.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.157.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.157.3.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterRGB888.

7.157.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB888.

7.157.3.5 setAlpha()

```
void setAlpha (
            uint8_t alpha )
```

Sets an alpha value for the painter.

Parameters

alpha	The alpha value to use.

7.157.3.6 setColor()

```
void setColor (
            colortype color,
            uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

7.158 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.

Additional Inherited Members

7.158.1 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

7.158.2 Constructor & Destructor Documentation

7.158.2.1 PainterRGB888Bitmap()

```
PainterRGB888Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.158.3 Member Function Documentation

7.158.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.158.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterRGB888.

7.158.3.3 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.

7.158.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB888.

7.158.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.158.3.6 setBitmap()

```
void setBitmap (
            const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.159 PainterRGB888L8Bitmap Class Reference

A Painter that will paint using a bitmap.

`#include <touchgfx/widgets/canvas/PainterRGB888L8Bitmap.hpp>`

Public Member Functions

- PainterRGB888L8Bitmap (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 uint8_t * bitmapPointer

  Pointer to the bitmap (L8)

- const uint8_t * bitmapExtraPointer

  Pointer to the CLUT (L8)

- 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.

Additional Inherited Members

7.159.1 Detailed Description

PainterRGB888L8Bitmap 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

7.159.2 Constructor & Destructor Documentation

7.159.2.1 PainterRGB888L8Bitmap()

```
PainterRGB888L8Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.159.3 Member Function Documentation

7.159.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.159.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterRGB888.

7.159.3.3 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.

7.159.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGB888.

7.159.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.159.3.6 setBitmap()

```
void setBitmap (
             const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.160 PainterRGBA2222 Class Reference

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

`#include <touchgfx/widgets/canvas/PainterRGBA2222.hpp>`

Public Member Functions

- PainterRGBA2222 (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 painterColor

  The color.
- uint8_t painterRed

  The red part of the color, scaled up to [0..255].
- uint8_t painterGreen

  The green part of the color, scaled up to [0..255].
- uint8_t painterBlue

  The blue part of the color, scaled up to [0..255].
- uint8_t painterAlpha

  The alpha value.

Additional Inherited Members

7.160.1 Detailed Description

The PainterRGBA2222 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

7.160.2 Constructor & Destructor Documentation

7.160.2.1 PainterRGBA2222()

```
PainterRGBA2222 (
            colortype color = 0,
            uint8_t alpha = 255 )
```

Constructor.

Parameters

color	the color.
alpha	the alpha.

7.160.3 Member Function Documentation

7.160.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.160.3.2 getColor()

```
colortype getColor ( ) const
```

Gets the current color.

Returns

The color.

7.160.3.3 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterRGBA2222.

7.160.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGBA2222.

7.160.3.5 setAlpha()

```
void setAlpha (
            uint8_t alpha )
```

Sets an alpha value for the painter.

Parameters

alpha	The alpha value to use.

7.160.3.6 setColor()

```
void setColor (
            colortype color,
            uint8_t alpha = 255 )
```

Sets color and alpha to use when drawing the CanvasWidget.

Parameters

color	The color.
alpha	The alpha.

7.161 PainterRGBA2222Bitmap Class Reference

A Painter that will paint using a bitmap.

`#include <touchgfx/widgets/canvas/PainterRGBA2222Bitmap.hpp>`

Public Member Functions

- PainterRGBA2222Bitmap (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 uint8_t * bitmapRGBA2222Pointer

  Pointer to the bitmap (RGBA2222)
- 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

7.161.1 Detailed Description

PainterRGBA2222Bitmap 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

7.161.2 Constructor & Destructor Documentation

7.161.2.1 PainterRGBA2222Bitmap()

```
PainterRGBA2222Bitmap (
            const Bitmap & bmp = Bitmap(BITMAP_INVALID),
            uint8_t alpha = 255 )
```

Constructor.

Parameters

bmp	The bitmap.
alpha	the alpha.

7.161.3 Member Function Documentation

7.161.3.1 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.161.3.2 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	*ptr*	Pointer to the row in the RenderingBuffer.
	x	The x coordinate.
	xAdjust	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).
	y	The y coordinate.
	count	Number of pixels to fill.
	covers	The coverage in of each pixel.

Reimplemented from AbstractPainterRGBA2222.

7.161.3.3 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 AbstractPainterRGBA2222.

7.161.3.4 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	*red*	The red.
out	*green*	The green.
out	*blue*	The blue.
out	*alpha*	The alpha.

Returns

true if the pixel should be painted, false otherwise.

Implements AbstractPainterRGBA2222.

7.161.3.5 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

alpha	The alpha value to use if there is no alpha channel in the bitmap.

7.161.3.6 setBitmap()

```
void setBitmap (
            const Bitmap & bmp )
```

Sets a bitmap to be used when drawing the CanvasWidget.

Parameters

bmp	The bitmap.

7.162 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.

7.162.1 Detailed Description

template<class T1, class T2>
struct touchgfx::Pair< T1, T2 >

A simple struct for holding pairs of data.

Template Parameters

T1	The type of the first element.
T2	The type of the second element.

7.162.2 Constructor & Destructor Documentation

7.162.2.1 Pair() [1/3]

```
Pair ( )  [inline]
```

Constructor initializing the elements it holds using their default constructors.

7.162.2.2 Pair() [2/3]

```
Pair (
            const T1 & x,
            const T2 & y )  [inline]
```

Constructor initializing the elements it holds using their copy constructor.

Parameters

x	Reference to the first element.
y	Reference to the second element.

7.162.2.3 Pair() [3/3]

```
Pair (
            const Pair< U, V > & p )  [inline]
```

Copy constructor.

Template Parameters

U	Generic type parameter.
V	Generic type parameter.

Parameters

p	The pair to copy from.

7.163 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.

7.163.1 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

ListOfTypes	Type of the list of types.
NUMBER_OF_ELEMENTS	Type of the number of elements.

See also

AbstractPartition

7.163.2 Member Typedef Documentation

7.163.2.1 SupportedTypesList

```
ListOfTypes SupportedTypesList
```

Provides a generic public type containing the list of supported types.

7.163.3 Member Enumeration Documentation

7.163.3.1 anonymous enum

```
anonymous enum
```

Compile-time generated constants specifying the "element" or "slot" size used by this partition

7.163.4 Constructor & Destructor Documentation

7.163.4.1 Partition()

```
Partition ( )  [inline]
```

Constructs an empty Partition.

7.163.5 Member Function Documentation

7.163.5.1 capacity()

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

Specialization of AbstractPartition::capacity().

Returns

An uint16_t.

See also

touchgfx::AbstractPartition::capacity()

Implements AbstractPartition.

7.163.5.2 element() [1/2]

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

Parameters

index	Zero-based index of the.

Returns

null if it fails, else a void*.

See also

touchgfx::AbstractPartition::element()

Implements AbstractPartition.

7.163.5.3 element() [2/2]

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

Parameters

index	Zero-based index of the.

Returns

null if it fails, else a void*.

See also

touchgfx::AbstractPartition::element()

Implements AbstractPartition.

7.163.5.4 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.

7.164 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

7.164.1 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

7.164.2 Constructor & Destructor Documentation

7.164.2.1 PixelDataWidget()

`PixelDataWidget ( )`

Default constructor.

7.164.3 Member Function Documentation

7.164.3.1 draw()

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

Draw the part of the RAM buffer that is inside the invalidated area.

Parameters

invalidatedArea	The region of this drawable that needs to be redrawn.

See also

touchgfx::Drawable

Implements Drawable.

7.164.3.2 getAlpha()

```
uint8_t getAlpha ( ) const
```

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.164.3.3 getSolidRect()

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

Report this widget as being completely solid.

Returns

The solid rect.

Implements Drawable.

7.164.3.4 setAlpha()

```
void setAlpha (
            uint8_t a )
```

Sets the alpha channel for the image.

Parameters

a	The alpha value. 255 = completely solid.

7.164.3.5 setBitmapFormat()

```
void setBitmapFormat (
             Bitmap::BitmapFormat format )
```

Set the format of the pixel data.

Parameters

format	Describes the format to use.

7.164.3.6 setPixelData()

```
void setPixelData (
             uint8_t *const data )
```

Set the pixel data to display.

Parameters

in,out	*data*	If non-null, the data.

7.165 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.

7.165.1 Member Function Documentation

7.165.1.1 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.

7.166 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.

7.167 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

7.167.1 Detailed Description

This class represents a homogeneous 3D point.

See also

quadruple

7.167.2 Constructor & Destructor Documentation

7.167.2.1 Point4() [1/2]

```
Point4 ( )  [inline]
```

Default constructor.

7.167.2.2 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.

7.168 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.

7.168.1 Detailed Description

template<class T>
class touchgfx::PreRenderable< T >

This mixin can be used on any Drawable. It provides a preRender function, which will cache the current visual appearance of the Drawable to be cache in a memory region. Subsequent calls to draw() on this Drawable will result in a simple memcpy of the cached memory instead of the normal draw operation. This mixin can therefore be used on Drawables whose visual appearance is static and the normal draw operation takes a long time to compute.

Note

The actual uses of this mixin are rare, and the class is mainly provided for example purposes.

Template Parameters

T	The type of Drawable to add this functionality to.

7.168.2 Constructor & Destructor Documentation

7.168.2.1 PreRenderable()

```
PreRenderable ( )  [inline]
```

Default constructor. Initializes the PreRenderable.

7.168.3 Member Function Documentation

7.168.3.1 draw()

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

Overrides the draw function. If preRender has been called, perform a memcpy of the cached version. If not, just call the base class version of draw.

Parameters

invalidatedArea	The subregion of this Drawable which needs to be redrawn.

7.168.3.2 isPreRendered()

```
bool isPreRendered ( ) const  [inline]
```

Returns

Is the widget rendered.

7.168.3.3 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.

7.168.3.4 setupDrawChain()

```
void setupDrawChain (
            const Rect & invalidatedArea,
            Drawable ** nextPreviousElement )  [inline], [virtual]
```

Note

For TouchGFX internal use only.

Parameters

	invalidatedArea	Include drawables that intersect with this area only.
in,out	*nextPreviousElement*	Modifiable element in linked list.

7.169 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.

7.169.1 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.

7.169.2 Constructor & Destructor Documentation

7.169.2.1 ~Presenter()

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

Destructor.

7.169.2.2 Presenter()

```
Presenter ( )  [inline], [protected]
```

Default constructor.

7.169.3 Member Function Documentation

7.169.3.1 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.

7.169.3.2 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.

7.170 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.

- template<typename T >
  T to () const

  Converts the Q10 value to an int or a float.

7.170.1 Detailed Description

Defines a number with 10 bits reserved for the fractional part of the decimal number. Q10 implements some simple arithmetic operations, most yielding a Q10 number and some yielding a Q5 number or a Q15 number as a result.

Q5∗Q5=Q10, Q10/Q5=Q5, ...

See also

Q5
Q15
http://en.wikipedia.org/wiki/Q_%28number_format%29
http://en.wikipedia.org/wiki/Fixed-point_arithmetic

7.170.2 Constructor & Destructor Documentation

7.170.2.1 Q10() [1/2]

```
Q10 ( )  [inline]
```

Default constructor.

7.170.2.2 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	int pre-formattet in Q10 format.

7.170.3 Member Function Documentation

7.170.3.1 operator int()

```
operator int ( ) const  [inline]
```

Gets the Q10 as an integer without conversion.

Returns

The unconverted Q10 value.

7.170.3.2 operator∗()

```
Q15 operator* (
            const Q5 & q5 ) const  [inline]
```

Multiplication operator. The result is a Q15, not a Q10, for increased precision.

Parameters

q5	The Q5 to multiply this with.

Returns

The result of the operation.

7.170.3.3 operator+()

```
Q10 operator+ (
            const Q10 & q10 ) const  [inline]
```

Addition operator.

Parameters

q10	The Q10 to add to this.

Returns

The result of the operation.

7.170.3.4 operator-()

```
Q10 operator- ( ) const  [inline]
```

Negation operator.

Returns

The negative value of this.

7.170.3.5 operator/()

```
Q5 operator/ (
            const Q5 & q5 ) const  [inline]
```

Division operator.

Parameters

q5	The Q5 to divide this by.

Returns

The result of the operation.

7.170.3.6 to()

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

Convert the Q10 value to an integer by removing the 10 bits used for the fraction, or to a floating point value by dividing by 32 ∗ 32, depending on the type specified as T.

Template Parameters

T	Either int or float.

Returns

Q10 value as a type T.

7.171 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.

7.171.1 Detailed Description

Defines a number with 15 bits reserved for the fractional part of the decimal number. Q15 is only used for sine/cosine and for intermediate calculations when multiplying.

Q5∗Q5=Q10, Q10/Q5=Q5, ...

See also

Q5
Q10
http://en.wikipedia.org/wiki/Q_%28number_format%29
http://en.wikipedia.org/wiki/Fixed-point_arithmetic

7.171.2 Constructor & Destructor Documentation

7.171.2.1 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	int pre-formattet in Q15 format.

7.171.3 Member Function Documentation

7.171.3.1 operator int()

```
operator int ( ) const  [inline]
```

Gets the Q15 as an integer without conversion.

Returns

The unconverted Q15 value.

7.171.3.2 operator+()

```
Q15 operator+ (
            const Q15 & q15 ) const  [inline]
```

Addition operator.

Parameters

q15	The Q10 to add to this.

Returns

The result of the operation.

7.171.3.3 operator-()

```
Q15 operator- ( ) const  [inline]
```

Negation operator.

Returns

The negative value of this.

7.171.3.4 operator/()

```
Q10 operator/ (
            const Q5 & q5 ) const  [inline]
```

Q5 / Q5 which requires the result of a Q15 / Q5 to be calculated.

Parameters

q5	The Q5 to divide this by.

Returns

The result of the operation.

7.172 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.

7.172.1 Detailed Description

Defines a number with 5 bits reserved for the fractional part of the decimal number. Q5 implements some simple arithmetic operations, most yielding a Q5 number and some yielding a Q10 number as a result. Other operations also work with Q15 numbers.

See also

Q10
Q15
http://en.wikipedia.org/wiki/Q_%28number_format%29
http://en.wikipedia.org/wiki/Fixed-point_arithmetic

7.172.2 Constructor & Destructor Documentation

7.172.2.1 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	Integer pre-formattet in Q5 format.

7.172.2.2 Q5() [2/2]

```
Q5 (
            const Q10 q10 )  [inline]
```

Constructor from Q10. The Q10 is shifted down to convert it to Q5.

Parameters

q10	The Q10 value to convert to a Q5 value.

See also

Q10

7.172.3 Member Function Documentation

7.172.3.1 operator int()

```
operator int ( ) const  [inline]
```

Gets the Q5 as an integer without conversion.

Returns

The unconverted Q5 value.

7.172.3.2 operator∗() [1/3]

```
Q10 operator* (
            const Q5 & q5 ) const  [inline]
```

Multiplication operator. The result is a Q10, not a Q5, for increased precision.

Parameters

q5	The Q5 to multiply this with.

Returns

The result of the operation.

See also

Q10

7.172.3.3 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

q15	The Q15 to multiply this with.

Returns

The result of the operation.

See also

Q15

7.172.3.4 operator∗() [3/3]

```
Q5 operator* (
            const int i ) const  [inline]
```

Multiplication operator.

Parameters

i	The integer to multiply this with.

Returns

The result of the operation.

7.172.3.5 operator+()

```
Q5 operator+ (
            const Q5 & q5 ) const  [inline]
```

Addition operator.

Parameters

q5	The Q5 to add to this.

Returns

The result of the operation.

7.172.3.6 operator-() [1/2]

```
Q5 operator- ( ) const  [inline]
```

Negation operator.

Returns

The negative value of this.

7.172.3.7 operator-() [2/2]

```
Q5 operator- (
            const Q5 & q5 ) const  [inline]
```

Subtraction operator.

Parameters

q5	The Q5 to subtract from this.

Returns

The result of the operation.

7.172.3.8 operator/() [1/2]

```
Q5 operator/ (
             const int i ) const  [inline]
```

Division operator.

Parameters

i	The integer to divide this by.

Returns

The result of the operation.

7.172.3.9 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

q5	The Q5 to divide this by.

Returns

The result of the operation.

See also

Q10

7.172.3.10 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

T	Either int or float.

Returns

Q5 value as a type T.

7.173 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].

7.173.1 Detailed Description

Base class for homogeneous vectors and points.

7.173.2 Constructor & Destructor Documentation

7.173.2.1 Quadruple() [1/2]

```
Quadruple ( )  [inline], [protected]
```

Default constructor. < The elements[3]

7.173.2.2 Quadruple() [2/2]

```
Quadruple (
            float x,
            float y,
            float z,
            float w )  [inline], [protected]
```

Constructor.

Parameters

x	The x value.
y	The y value.
z	The z value.
w	The w value.

7.173.3 Member Function Documentation

7.173.3.1 getElement()

```
float getElement (
            int row ) const  [inline]
```

Gets an element.

Parameters

row	The row.

Returns

The element.

7.173.3.2 getW()

```
float getW ( ) const  [inline]
```

Gets the w.

Returns

The w.

7.173.3.3 getX()

```
float getX ( ) const  [inline]
```

Get x coordinate.

Returns

The x coordinate.

7.173.3.4 getY()

```
float getY ( ) const  [inline]
```

Get y coordinate.

Returns

The y coordinate.

7.173.3.5 getZ()

```
float getZ ( ) const  [inline]
```

Get z coordinate.

Returns

The z coordinate.

7.173.3.6 setElement()

```
void setElement (
            int row,
            float value )  [inline]
```

Sets an element.

Parameters

row	The row.
value	The value.

7.173.3.7 setW()

```
void setW (
             float value )  [inline]
```

Sets a w.

Parameters

value	The value.

7.173.3.8 setX()

```
void setX (
             float value )  [inline]
```

Sets an x coordinate.

Parameters

value	The value.

7.173.3.9 setY()

```
void setY (
             float value )  [inline]
```

Sets a y coordinate.

Parameters

value	The value.

7.173.3.10 setZ()

```
void setZ (
             float value )  [inline]
```

Sets a z coordinate.

Parameters

value	The value.

7.174 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

7.174.1 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

7.174.2 Constructor & Destructor Documentation

7.174.2.1 RadioButton()

```
RadioButton ( )  [inline]
```

Default constructor.

7.174.2.2 ~RadioButton()

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

Destructor.

7.174.3 Member Function Documentation

7.174.3.1 draw()

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

Draws the given invalidated area.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Implements Drawable.

7.174.3.2 getAlpha()

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

Gets the current alpha value.

Returns

The current alpha value.

7.174.3.3 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.

7.174.3.4 getDeselectionEnabled()

```
bool getDeselectionEnabled ( ) const  [inline]
```

Gets the current deselectionEnabled state.

Returns

The current deselectionEnabled state.

7.174.3.5 getSelected()

```
bool getSelected ( ) const  [inline]
```

Gets the current selected state.

Returns

The current selected state.

7.174.3.6 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.

7.174.3.7 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_RADIOBUTTON.

Reimplemented from AbstractButton.

7.174.3.8 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

event	Information about the click.

See also

Drawable::handleClickEvent()

Reimplemented from AbstractButton.

7.174.3.9 setAlpha()

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

Sets the alpha channel for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

7.174.3.10 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

bmpUnselected	Bitmap to use when button is unselected.
bmpUnselectedPressed	Bitmap to use when button is unselected and pressed.
bmpSelected	Bitmap to use when button is selected.
bmpSelectedPressed	Bitmap to use when button is selected and pressed.

7.174.3.11 setDeselectedAction()

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

Associates an action to be performed when the AbstractButton is deselected.

Parameters

callback	The callback to be executed. The callback will be given a reference to the AbstractButton.

See also

GenericCallback

7.174.3.12 setDeselectionEnabled()

```
void setDeselectionEnabled (
            bool state )  [inline]
```

States whether or not it is possible to de-select the RadioButton by clicking it.

Parameters

state	true if it should be possible to de-select by click.

7.174.3.13 setSelected()

```
void setSelected (
            bool newSelected )
```

Sets the radio buttons selected state.

Parameters

newSelected	the new selected state.

7.175 RadioButtonGroup< CAPACITY > Class Template Reference

Class for handling a collection of RadioButtons.

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

Public Member Functions

- RadioButtonGroup ()

  Default constructor.
- virtual ~RadioButtonGroup ()

  Destructor.
- virtual void add (RadioButton &radioButton)

  Add the RadioButton to the RadioButtonGroup.
- virtual RadioButton * getRadioButton (uint16_t index) const

  Gets the RadioButton at the specified index.
- virtual int32_t getSelectedRadioButtonIndex () const

  Gets the index of the selected RadioButton.
- virtual RadioButton * getSelectedRadioButton () const

  Gets the selected RadioButton.
- virtual void setSelected (RadioButton &radioButton)

  Sets the specified RadioButton to be selected.
- virtual void setDeselectionEnabled (bool deselectionEnabled)

  Sets whether or not it is possible to deselect RadioButtons by clicking them when they are selected.
- virtual bool getDeselectionEnabled () const

  Gets the current deselectionEnabled state.
- void setRadioButtonSelectedHandler (GenericCallback< const AbstractButton & > &callback)

  Associate an action with a radio button.
- void setRadioButtonDeselectedHandler (GenericCallback< const AbstractButton & > &callback)

  Associate an action with a radio button.

Protected Member Functions

- virtual void radioButtonClickedHandler (const AbstractButton &radioButton)

  Handles the event that a RadioButton has been selected.
- virtual void radioButtonDeselectedHandler (const AbstractButton &radioButton)

  Handles the event that a RadioButton has been deselected.

Protected Attributes

- RadioButton * radioButtons [CAPACITY]

  The list of added RadioButtons.
- uint16_t size

  The current number of added RadioButtons.
- Callback< RadioButtonGroup, const AbstractButton &> radioButtonClicked

  Callback that is attached to the RadioButtons.
- Callback< RadioButtonGroup, const AbstractButton &> radioButtonUnselected

  Callback that is attached to the RadioButtons.
- GenericCallback< const AbstractButton &> * radioButtonSelectedCallback

  The callback to be executed when a radio button belonging to this group is selected.
- GenericCallback< const AbstractButton &> * radioButtonDeselectedCallback

  The callback to be executed when a radio button belonging to this group is deselected.

7.175.1 Detailed Description

template<uint16_t CAPACITY>
class touchgfx::RadioButtonGroup< CAPACITY >

Class for handling a collection of RadioButtons. The RadioButtonGroup handles the de-selection of radio buttons when a new selection occurs. A callback is executed when a new selection occurs reporting the newly selected RadioButton.

Template class: specify a CAPACITY, that is the number of RadioButtons to store.

Template Parameters

CAPACITY	Type of the capacity.

See also

RadioButton

7.175.2 Constructor & Destructor Documentation

7.175.2.1 RadioButtonGroup()

```
RadioButtonGroup ( )  [inline]
```

Default constructor.

7.175.2.2 ~RadioButtonGroup()

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

Destructor.

7.175.3 Member Function Documentation

7.175.3.1 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	*radioButton*	the RadioButton that is to be added.

7.175.3.2 getDeselectionEnabled()

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

Gets the current deselectionEnabled state.

Returns

The current deselectionEnabled state.

7.175.3.3 getRadioButton()

```
RadioButton * getRadioButton (
            uint16_t index ) const  [inline], [virtual]
```

Gets the RadioButton at the specified index.

Parameters

index	the index of the RadioButton to return.

Returns

the RadioButton at the specified index. Returns 0 if illegal index.

7.175.3.4 getSelectedRadioButton()

```
RadioButton * getSelectedRadioButton ( ) const  [inline], [virtual]
```

Gets the selected RadioButton.

Returns

a pointer to the selected RadioButton. Returns 0 if no RadioButton is selected.

7.175.3.5 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.

7.175.3.6 radioButtonClickedHandler()

```
void radioButtonClickedHandler (
            const AbstractButton & radioButton )  [inline], [protected], [virtual]
```

Handles the event that a RadioButton has been selected. deselects all other RadioButtons.

Parameters

radioButton	the RadioButton that has been selected.

7.175.3.7 radioButtonDeselectedHandler()

```
void radioButtonDeselectedHandler (
            const AbstractButton & radioButton )  [inline], [protected], [virtual]
```

Handles the event that a RadioButton has been deselected.

Parameters

radioButton	the RadioButton that has been deselected.

7.175.3.8 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

deselectionEnabled	true if it should be possible to deselect by click.

7.175.3.9 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

callback	The callback to be executed. The callback will be given a reference to the RadioButton that was selected.

See also

GenericCallback

7.175.3.10 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

callback	The callback to be executed. The callback will be given a reference to the RadioButton that was selected.

See also

GenericCallback

7.175.3.11 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	*radioButton*	the RadioButton to be selected.

7.176 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.

7.176.1 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".

7.176.2 Member Enumeration Documentation

7.176.2.1 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.

7.176.2.2 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.

7.176.2.3 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.

7.176.3 Constructor & Destructor Documentation

7.176.3.1 Rasterizer()

```
Rasterizer ( )  [inline]
```

Default constructor.

7.176.4 Member Function Documentation

7.176.4.1 calculateAlpha()

```
unsigned calculateAlpha (
             int area ) const  [inline]
```

Calculates the alpha.

Parameters

area	The area.

Returns

The calculated alpha.

7.176.4.2 lineTo()

```
void lineTo (
             int x,
             int y )  [inline]
```

Line to.

Parameters

x	The x coordinate.
y	The y coordinate.

7.176.4.3 moveTo()

```
void moveTo (
             int x,
             int y )  [inline]
```

Move to.

Parameters

x	The x coordinate.
y	The y coordinate.

7.176.4.4 render()

```
template< class Renderer > bool render (
             Renderer & r )  [inline]
```

Renders this object.

Template Parameters

Renderer	Type of the renderer.

Parameters

in	*r*	The Renderer to process.

Returns

true there was enough memory available to draw the outline and render the graphics, false if there was insufficient memory and nothing was drawn.

7.176.4.5 reset()

```
void reset ( )  [inline]
```

Resets this object. Basically this is done by resetting the the Outline.

7.176.4.6 setFillingRule()

```
void setFillingRule (
            FillingRule fillingRule )  [inline]
```

Sets the filling rule to be used when rendering the outline.

Parameters

fillingRule	The filling rule.

7.176.4.7 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 of higher than the given y.

Parameters

y	The max y coordinate to render for the Outline.

7.176.4.8 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.

7.177 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.

7.177.1 Detailed Description

Class representing a Rectangle with a few convenient methods. Size: 8 bytes.

7.177.2 Constructor & Destructor Documentation

7.177.2.1 Rect() [1/2]

`Rect ( ) [inline]`

Default constructor. Resulting in an empty Rect with coordinates 0,0.

7.177.2.2 Rect() [2/2]

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

Constructor.

Parameters

x	The x coordinate.
y	The y coordinate.
width	The width.
height	The height.

7.177.3 Member Function Documentation

7.177.3.1 area()

```
uint32_t area ( ) const  [inline]
```

Calculate the area of the rectangle.

Returns

area of the rectangle.

7.177.3.2 bottom()

```
int16_t bottom ( ) const  [inline]
```

Gets the y coordinate of the bottom edge of the Rect.

Returns

y coordinate of the buttom edge.

7.177.3.3 expandToFit()

```
void expandToFit (
            const Rect & other )  [inline]
```

Increases the area covered by this rectangle to encompass the area covered by supplied rectangle.

Parameters

other	The other rectangle.

7.177.3.4 includes()

```
bool includes (
            const Rect & other ) const  [inline]
```

Determines whether the specified rectangle is completely included in this rectangle.

Parameters

other	The other rectangle.

Returns

true if the specified rectangle is completely included.

7.177.3.5 intersect() [1/2]

```
bool intersect (
            int16_t otherX,
            int16_t otherY ) const  [inline]
```

Determines whether specified point lies inside this rectangle.

Parameters

otherX	The x coordinate of the point.
otherY	The y coordinate of the point.

Returns

true if point lies inside rectangle.

7.177.3.6 intersect() [2/2]

```
bool intersect (
            const Rect & other ) const  [inline]
```

Determines whether specified rectangle intersects with this rectangle.

Parameters

other	The other rectangle.

Returns

true if the two rectangles intersect.

7.177.3.7 isEmpty()

```
bool isEmpty ( ) const  [inline]
```

Query if this object is empty.

Returns

true if any of the dimensions are 0.

7.177.3.8 operator"!=()

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

Opposite of the == operator.

Parameters

other	The Rect to compare with.

Returns

true if the compared Rect differ in dimensions or coordinates.

7.177.3.9 operator&()

```
Rect operator & (
            const Rect & other ) const  [inline]
```

Gets a rectangle describing the intersecting area between this rectangle and the supplied rectangle.

Parameters

other	The other rectangle.

Returns

Intersecting rectangle or Rect(0, 0, 0, 0) in case of no intersection.

7.177.3.10 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

other	The rect to intersect with.

7.177.3.11 operator==()

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

Compares equality of two Rect by the dimensions and position of these.

Parameters

other	The Rect to compare with.

Returns

true if the compared Rect have the same dimensions and coordinates.

7.177.3.12 right()

```
int16_t right ( ) const  [inline]
```

Gets the x coordinate of the right edge of the Rect.

Returns

x coordinate of the right edge.

7.178 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.

7.178.1 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.

7.178.2 Constructor & Destructor Documentation

7.178.2.1 Renderer() [1/2]

```
Renderer ( )  [inline]
```

Default constructor. Function setRenderingBuffer() should be called to specify where the polygon should be rendered.

7.178.2.2 Renderer() [2/2]

```
Renderer (
            RenderingBuffer & renderingBuffer,
            AbstractPainter & painter )  [inline]
```

Constructor.

Parameters

in	*renderingBuffer*	The screen buffer to render the polygon in.
in	*painter*	The painter to use for drawing individual pixels in a scanline.

7.178.3 Member Function Documentation

7.178.3.1 getRenderingBuffer()

```
RenderingBuffer & getRenderingBuffer ( )  [inline]
```

Gets the getRenderingBuffer.

Returns

A RenderingBuffer&

7.178.3.2 render()

```
void render (
            const Scanline & scanline )  [inline]
```

Render the given Scanline in the given color.

Parameters

scanline	The Scanline.

7.178.3.3 setRenderingBuffer()

```
void setRenderingBuffer (
            RenderingBuffer & renderingBuffer )  [inline]
```

Sets rendering buffer.

Parameters

in	*renderingBuffer*	The screen buffer to render the polygon in.

7.179 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.

7.179.1 Detailed Description

Rendering buffer wrapper. This class does not know anything about memory organizations, all it does it keeps an array of pointers to each pixel row. The general rules of rendering are as follows.

1. Allocate or create somehow a rendering buffer itself. Since the library does not depend on any particular platform or architecture it was decided that it's your responsibility to create and destroy rendering buffers properly. You can use any available mechanism to create it - you can use a system API function, simple memory allocation, or even statically defined array. You also should know the memory organization (or possible variants) in your system. For example, there's an R,G,B or B,G,R organizations with one byte per component (three bytes per pixel) is used very often. So, if you intend to use class render_bgr24, for example, you should allocate at least width*height*3 bytes of memory.

2. Create a RenderingBuffer object and then call method attach(). It requires a pointer to the buffer itself, width and height of the buffer in pixels, and the length of the row in bytes. All these values must properly correspond to the memory organization. The argument stride is used because in reality the row length in bytes does not obligatory correspond with the width of the image in pixels, i.e. it cannot be simply calculated as width_in_↩ pixels ∗ bytes_per_pixel. For example, it must be aligned to 4 bytes in Windows bitmaps. Method attach() can be called more than once. The execution time of it is very little, still it allocates memory of heigh ∗ sizeof(char∗) bytes and has a loop while (height–) {...}, so it's unreasonable to call it every time before drawing any single pixel :-)
3. Create an object (or a number of objects) of a rendering class, such as renderer_bgr24_solid, renderer↩ _bgr24_image and so on. These classes require a pointer to the RenderingBuffer object, but they do not perform any considerable operations except storing this pointer. So, rendering objects can be created on demand almost any time. These objects know about concrete memory organization (this knowledge is hard coded), so actually, the memory you allocated or created in clause 1 should actually be in correspondence to the needs of the rendering class.
4. Render your image using rendering classes, for example, Rasterizer
5. Display the result, or store it, or whatever. It's also your responsibility and depends on the platform.

7.179.2 Constructor & Destructor Documentation

7.179.2.1 RenderingBuffer() [1/2]

```
RenderingBuffer ( )
```

Default constructor.

7.179.2.2 ∼RenderingBuffer()

```
∼RenderingBuffer ( )
```

Destructor.

7.179.2.3 RenderingBuffer() [2/2]

```
RenderingBuffer (
            unsigned char * buf_,
            unsigned char xAdjust_,
            unsigned width_,
            unsigned height_,
            int stride_ )
```

Constructor.

Parameters

in	*buf_*	Pointer to the frame buffer where the image is rendered.
	x↩ Adjust↩ _	Horizontal adjustment of the x coordinate, used when bits per pixel is less than eight which implies that a uint8_t pointer cannot precisely address the start of the frame buffer.
	*width_*	The width of the frame buffer to write.
	height↩ _	The height of the frame buffer to write.
	stride↩ _	How much to add the a pointer inside the frame buffer to advance to the next line in the frame buffer.

7.179.3 Member Function Documentation

7.179.3.1 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	*buf_*	Pointer to the frame buffer where the image is rendered.
	*xAdjust_*	Horizontal adjustment of the x coordinate, used when bits per pixel is less than eight which implies that a uint8_t pointer cannot precisely address the start of the frame buffer.
	*width_*	The width of the frame buffer to write.
	*height_*	The height of the frame buffer to write.
	*stride_*	How much to add the a pointer inside the frame buffer to advance to the next line in the frame buffer.

7.179.3.2 getHeight()

```
unsigned getHeight ( ) const  [inline]
```

Gets the height.

Returns

The height.

7.179.3.3 getWidth()

```
unsigned getWidth ( ) const  [inline]
```

Gets the width.

Returns

The width.

7.179.3.4 getXAdjust()

```
unsigned char getXAdjust ( ) const  [inline]
```

Gets x coordinate adjust.

Returns

The x coordinate adjust.

7.179.3.5 inbox()

```
bool inbox (
            int x,
            int y ) const  [inline]
```

Tests if a given coordinate is inside the RenderingBuffer.

Parameters

x	The x coordinate.
y	The y coordinate.

Returns

true if (x,y) is inside the RenderingBuffer, false otherwise.

7.179.3.6 row() [1/2]

```
unsigned char * row (
            unsigned y )  [inline]
```

Gets a pointer to the given row in the RenderingBuffer.

Parameters

y	The line number, ie the row.

Returns

The pointer to the start of the given line in the RenderingBuffer.

7.179.3.7 row() [2/2]

```
const unsigned char * row (
            unsigned y ) const  [inline]
```

Gets a pointer to the given row in the RenderingBuffer.

Parameters

y	The line number, ie the row.

Returns

The pointer to the start of the given line in the RenderingBuffer.

7.180 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

7.180.1 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, the after a given delay and then repeatedly after an interval.

See also

Button

7.180.2 Constructor & Destructor Documentation

7.180.2.1 RepeatButton()

`RepeatButton ( )`

Default constructor. Sets delay to 10 ticks and interval to 5 ticks.

See also

setDelay
setInterval

7.180.3 Member Function Documentation

7.180.3.1 getDelay()

```
int getDelay ( )  [virtual]
```

Gets the delay in ticks.

Returns

The delay.

See also

setDelay

7.180.3.2 getInterval()

```
int getInterval ( )  [virtual]
```

Gets the interval in ticks.

Returns

The interval.

7.180.3.3 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

event	The event.

Reimplemented from AbstractButton.

7.180.3.4 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.

7.180.3.5 setDelay()

```
void setDelay (
            int delay )  [virtual]
```

Sets the number of ticks from the first button activation until the next time it gets activated.

Parameters

delay	The delay.

See also

setInterval
getDelay

7.180.3.6 setInterval()

```
void setInterval (
            int interval )  [virtual]
```

Sets the interval in number of ticks between each each activation of the pressed button.

Parameters

interval	The interval.

See also

setDelay
getInterval

7.181 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

7.181.1 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.

7.181.2 Member Function Documentation

7.181.2.1 getDelay()

```
int getDelay ( )  [inline]
```

Returns

The delay.

7.181.2.2 getInterval()

```
int getInterval ( )  [inline]
```

Returns

The interval.

7.181.2.3 handleClickEvent()

```
void handleClickEvent (
             const touchgfx::ClickEvent & event )  [inline], [virtual]
```

Parameters

event	The event.

Reimplemented from Drawable.

7.181.2.4 setDelay()

```
void setDelay (
             int delay )  [inline]
```

Parameters

delay	The delay.

7.181.2.5 setInterval()

```
void setInterval (
            int interval )  [inline]
```

Parameters

interval	The interval.

7.182 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

7.182.1 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

7.182.2 Member Enumeration Documentation

7.182.2.1 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.

7.182.3 Constructor & Destructor Documentation

7.182.3.1 ScalableImage()

```
ScalableImage ( )
```

Default constructor.

7.182.3.2 ~ScalableImage()

```
~ScalableImage ( )  [virtual]
```

Destructor.

7.182.4 Member Function Documentation

7.182.4.1 draw()

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

Draws the given invalidated area.

Parameters

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Implements Drawable.

7.182.4.2 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

	invalidatedArea	The invalidated area.
in,out	*fb*	If non-null, the fb.
	triangleXs	The triangle xs.
	triangleYs	The triangle ys.
	triangleZs	The triangle zs.
	triangleUs	The triangle us.
	triangleVs	The triangle vs.

7.182.4.3 getAlpha()

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

Gets the current alpha value.

Returns

The current alpha value.

7.182.4.4 getBitmap()

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

Gets the bitmap for the image.

Returns

the small bitmap.

7.182.4.5 getScalingAlgorithm()

```
ScalingAlgorithm getScalingAlgorithm ( )  [virtual]
```

Gets the algorithm used when rendering.

Returns

The algorithm used when rendering.

7.182.4.6 getSolidRect()

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

Gets solid rectangle.

Returns

largest possible solid rect.

See also

Drawable::getSolidRect()

Implements Drawable.

7.182.4.7 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_SCALABLEIMAGE.

Reimplemented from Widget.

7.182.4.8 lookupRenderVariant()

```
RenderingVariant lookupRenderVariant ( ) const  [protected]
```

Looks up the appropriate render variant based on the bitmap format and scaling algorithm.

Returns

A RenderingVariant.

7.182.4.9 setAlpha()

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

Sets the alpha channel for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

7.182.4.10 setBitmap()

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

Sets the bitmap for the image.

Parameters

bmp	The bitmap to be used by the widget.

7.182.4.11 setScalingAlgorithm()

```
void setScalingAlgorithm (
            ScalingAlgorithm algorithm )  [virtual]
```

Sets the algorithm to be used.

Parameters

algorithm	The algorithm to use when rendering.

7.183 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.

7.183.1 Detailed Description

This class is used to transfer data from class Outline (or a similar one) to the rendering buffer. It's organized very simple. The class stores information of horizontal spans to render it into a pixel-map buffer. Each span has initial X, length, and an array of bytes that determine the alpha values for each pixel. So, the restriction of using this class is 256 levels of Anti-Aliasing, which is quite enough for any practical purpose. Before using this class you should know the minimal and maximal pixel coordinates of your scanline. The protocol of using is: 1. reset()

1. addCell() / addSpan() - accumulate scanline. You pass y coordinate into these functions in order to make scanline know the last Y. Before calling addCell() / addSpan() you should check with method isReady(y) if the last Y has changed. It also checks if the scanline is not empty. When forming one scanline the next x coordinate must be always greater than the last stored one, i.e. it works only with ordered coordinates.

2. If the current scanline isReady() you should render it and then call resetSpans() before adding new cells/spans.

3. Rendering:

Scanline provides an iterator class that allows you to extract the spans and the cover values for each pixel. Be aware that clipping has not been done yet, so you

should perform it yourself. Use Scanline::iterator to render spans:

int baseX = scanline.getBaseX(); // base X. Should be added to the span's X // "scanline" is a const reference to the // scanline passed in.

int y = scanline.y(); // y coordinate of the scanline

...Perform vertical clipping here...

Scanline::iterator span(scanline);

unsigned char* row = renderingBuffer->row(y); // The the address of the beginning // of the current row

unsigned num_spans = scanline.getNumSpans(); // Number of spans. It's guaranteed that // numSpans is always greater than 0.

do { int x = span.next() + baseX; // The beginning X of the span

const int8u covers* = span.getCovers(); // The array of the cover values

int numPix = span.getNumPix(); // Number of pixels of the span. // Always greater than 0, still we // should use "int" instead of // "unsigned" because it's more // convenient for clipping

...Perform horizontal clipping here... ...you have x, covers, and pix_Fromcount...

unsigned char* dst = row + x; // Calculate the start address of the row. // In this case we assume a simple // grayscale image 1-byte per pixel. do { *dst++ = *covers++; // Hypotetical rendering. } while (–numPix); } while (–numSpans); // numSpans cannot be 0, so this loop is quite safe

The question is: why should we accumulate the whole scanline when we could render just separate spans when they're ready? That's because using the scanline is in general faster. When is consists of more than one span the conditions for the processor cash system are better, because switching between two different areas of memory (that can be large ones) occurs less frequently.

7.183.2 Constructor & Destructor Documentation

7.183.2.1 Scanline()

```
Scanline ( )
```

Default constructor. Initiate a Scanline by setting up pointers to store covers, and counts.

7.183.2.2 ~Scanline()

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

Destructor.

7.183.3 Member Function Documentation

7.183.3.1 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

x	The x coordinate.
y	The y coordinate.
cover	The cover.

7.183.3.2 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

x	The x coordinate.
y	The y coordinate.
len	The length.
cover	The cover.

7.183.3.3 getNumSpans()

```
unsigned getNumSpans ( ) const  [inline]
```

Gets number spans in the Scanline.

Returns

The number spans.

7.183.3.4 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.

7.183.3.5 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

y	The y coordinate.

Returns

True if the given y coordinate differs from the y coordinate for the cells in the current Scanline.

7.183.3.6 reset()

```
void reset ( )
```

Resets the Scanline object in preparation for the handling the next Scanline.

7.183.3.7 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().

7.184 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 (const 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).

7.184.1 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.

7.184.2 Constructor & Destructor Documentation

7.184.2.1 Screen()

```
Screen ( )
```

Default constructor.

7.184.2.2 ~Screen()

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

Destructor.

7.184.3 Member Function Documentation

7.184.3.1 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	*d*	The Drawable to add.

7.184.3.2 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()

7.184.3.3 bindTransition()

```
void bindTransition (
            Transition & trans )
```

Enables the transition to access the containers.

Parameters

in	*trans*	The transition to bind.

7.184.3.4 draw() [1/2]

```
void draw ( )
```

Tells the screen to draw its entire area.

Note

The more specific draw(Rect&) version is preferred when possible.

7.184.3.5 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	*rect*	The area in absolute coordinates.

7.184.3.6 getRootContainer()

```
Container & getRootContainer ( )  [inline]
```

Obtain a reference to the root container of this screen.

Returns

The root container.

7.184.3.7 handleClickEvent()

```
void handleClickEvent (
            const ClickEvent & evt )  [virtual]
```

Traverse the drawables in reverse z-order and notify them of a click event.

Parameters

evt	The event to handle.

7.184.3.8 handleDragEvent()

```
void handleDragEvent (
            const DragEvent & evt )  [virtual]
```

Traverse the drawables in reverse z-order and notify them of a drag event.

Parameters

evt	The event to handle.

7.184.3.9 handleGestureEvent()

```
void handleGestureEvent (
            const GestureEvent & evt )  [virtual]
```

Handle gestures. Traverses drawables in reverse-z and notifies them of the gesture.

Parameters

evt	The event to handle.

7.184.3.10 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

key	The key to handle.

7.184.3.11 handleTickEvent()

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

Called by the Application on the current screen with a frequency of Application::TICK_INTERVAL_MS.

7.184.3.12 JSMOC()

```
void JSMOC (
            const Rect & invalidatedArea,
            Drawable * widgetToDraw )
```

Recursive JSMOC function. This is the actual occlusion culling implementation.

Parameters

in	*invalidatedArea*	The area to redraw, expressed in absolute coordinates.
in	*widgetToDraw*	Widget currently being drawn.

7.184.3.13 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	*d*	The Drawable to remove.

7.184.3.14 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()

7.184.3.15 startSMOC()

```
void startSMOC (
            const Rect & invalidatedArea )
```

Starts a JSMOC run, analyzing what parts of what widgets should be redrawn.

Parameters

in	*invalidatedArea*	The area to redraw, expressed in absolute coordinates.

7.184.3.16 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()

7.184.3.17 useSMOCDrawing()

```
void useSMOCDrawing (
            bool enabled )  [protected]
```

Determines whether to use JSMOC or painter's algorithm for drawing.

Parameters

enabled	true if JSMOC should be enabled, false if disabled (meaning painter's algorithm is employed instead).

7.184.3.18 usingSMOC()

```
bool usingSMOC ( ) const  [inline]
```

Returns

true if this screen uses the JSMOC drawing algorithm.

7.185 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 alway 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

7.185.1 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

7.185.2 Constructor & Destructor Documentation

7.185.2.1 ScrollableContainer()

```
ScrollableContainer ( )
```

Default constructor.

7.185.2.2 ~ScrollableContainer()

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

Destructor.

7.185.3 Member Function Documentation

7.185.3.1 add()

```
void add (
            Drawable & d )  [virtual]
```

Adds a Drawable instance as child to this ScrollableContainer.

Parameters

in	*d*	The drawable.

Reimplemented from Container.

7.185.3.2 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.

7.185.3.3 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

deltaX	The horizontal amount to scroll.
deltaY	The vertical amount to scroll.

Returns

did the container actually scroll. The call doScroll(0,0) will always return false.

7.185.3.4 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

enable	If true (default), horizontal scrolling is enabled. If false, scrolling is disabled.

7.185.3.5 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

enable	If true (default), vertical scrolling is enabled. If false, scrolling is disabled.

7.185.3.6 getContainedArea()

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

Gets contained area.

Returns

The contained area.

Reimplemented from Container.

7.185.3.7 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

	x	The x coordinate of the (click) event.
	y	The y coordinate of the (click) event.
out	*last*	The last child intersecting x,y. ScrollableContainer intercepts these, so returns it self.

Reimplemented from Container.

7.185.3.8 getScrolledX()

```
int16_t getScrolledX ( ) const
```

Gets the distance scrolled for the x-axis.

Returns

the distance scrolled for the x-axis.

7.185.3.9 getScrolledY()

```
int16_t getScrolledY ( ) const
```

Gets the distance scrolled for the y-axis.

Returns

the distance scrolled for the y-axis.

7.185.3.10 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_SCROLLABLECONTAINER.

Reimplemented from Container.

7.185.3.11 getXBorder()

```
Rect getXBorder (
            const Rect & xBar,
            const Rect & yBar ) const  [protected]
```

Gets the area where the horizontal scrollbar can move.

Parameters

xBar	The current horizontal scrollbar, supplied for caching reasons.
yBar	The current vertical scrollbar, supplied for caching reasons.

Returns

The area.

7.185.3.12 getXScrollbar()

```
Rect getXScrollbar ( ) const  [protected]
```

Gets x coordinate of the scrollbar.

Returns

The horizontal scrollbar area.

7.185.3.13 getYBorder()

```
Rect getYBorder (
            const Rect & xBar,
            const Rect & yBar ) const  [protected]
```

Gets the area where the vertical scrollbar can move.

Parameters

xBar	The current horizontal scrollbar, supplied for caching reasons.
yBar	The current vertical scrollbar, supplied for caching reasons.

Returns

The area.

7.185.3.14 getYScrollbar()

```
Rect getYScrollbar ( ) const  [protected]
```

Gets y coordinate of the scrollbar.

Returns

The vertical scrollbar area.

7.185.3.15 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

evt	The ClickEvent.

Reimplemented from Drawable.

7.185.3.16 handleDragEvent()

```
void handleDragEvent (
            const DragEvent & evt )  [virtual]
```

Handle the drag event. Initiate a scrolling of the container. Update scrollbars.

Parameters

evt	The DragEvent.

Reimplemented from Drawable.

7.185.3.17 handleGestureEvent()

```
void handleGestureEvent (
            const GestureEvent & evt )  [virtual]
```

Gestures generate a scroll animation so these are intercepted in the same manner as drag events.

Parameters

evt	The GestureEvent.

Reimplemented from Drawable.

7.185.3.18 handleTickEvent()

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

Handle tick events. Used in updating the animation of the scroll.

Reimplemented from Drawable.

7.185.3.19 invalidateScrollbars()

```
void invalidateScrollbars ( )  [protected]
```

Invalidate the scrollbars.

7.185.3.20 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

in,out	*scrollX*	Is the container able to scroll horizontally.
in,out	*scrollY*	Is the container able to scroll vertically.

7.185.3.21 moveChildrenRelative()

```
void moveChildrenRelative (
            int16_t deltaX,
            int16_t deltaY )  [virtual]
```

Moves the scrollable contents relatively.

Parameters

deltaX	Horizontal displacement.
deltaY	Vertical displacement.

Reimplemented from Container.

7.185.3.22 reset()

```
void reset ( )
```

Resets the x/y coordinates of childrento the position they were in before the first drag event was received or to the position they were in the last time reset() was invoked.

7.185.3.23 setMaxVelocity()

```
void setMaxVelocity (
            uint16_t max )  [inline]
```

Sets the maximum velocity of a scroll due to a swipe.

Parameters

max	The maximum velocity of the scroll.

7.185.3.24 setScrollbarPadding()

```
void setScrollbarPadding (
            uint8_t padding )
```

Sets the amount of space the scrollbar has to its borders.

Parameters

padding	The padding.

7.185.3.25 setScrollbarsAlpha()

```
void setScrollbarsAlpha (
            uint8_t alpha )
```

Sets the alpha value for the scroll bars.

Parameters

alpha	The alpha value. 255 = completely solid.

7.185.3.26 setScrollbarsColor()

```
void setScrollbarsColor (
             colortype color )
```

Sets the color of the scroll bars.

Parameters

color	The color of the box.

7.185.3.27 setScrollbarsPermanentlyVisible()

```
void setScrollbarsPermanentlyVisible ( )
```

sets the visibility for the scrollbars to be permanent.

7.185.3.28 setScrollbarsVisible()

```
void setScrollbarsVisible (
             bool newVisible )
```

Sets the visibility of the scrollbars, when the scrollable area is pressed.

Parameters

newVisible	If true (default), the scrollbars are visible when scrollable area is pressed. If false, scrollbars are always hidden.

7.185.3.29 setScrollbarWidth()

```
void setScrollbarWidth (
             uint8_t width )
```

Sets the width of the scrollbar.

Parameters

width	The width of the scrollbar.

7.185.3.30 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

t	The new threshold value.

7.186 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

7.186.1 Detailed Description

A scroll base class with a list of drawables (DrawableList).

See also

ScrollWheelBase
ScrollList
DrawableList

7.186.2 Member Enumeration Documentation

7.186.2.1 AnimationState

```
enum AnimationState  [protected]
```

Enumerator

NO_ANIMATION	No animation.
ANIMATING_GESTURE	Animating a gesture.
ANIMATING_DRAG	Animating a click+drag.

7.186.3 Constructor & Destructor Documentation

7.186.3.1 ScrollBase()

```
ScrollBase ( )
```

Default constructor.

7.186.3.2 ~ScrollBase()

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

Destructor.

7.186.4 Member Function Documentation

7.186.4.1 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

enable	If true (default), horizontal scrolling is enabled. If false, scrolling is disabled.

7.186.4.2 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

enable	If true (default), vertical scrolling is enabled. If false, scrolling is disabled.

7.186.4.3 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

itemIndex	Zero-based index of the item.
animationSteps	(Optional) The steps to use for the animation. 0 means no animation. If omitted, default animation steps are used.

See also

setAnimationSteps

7.186.4.4 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

position	The new position.
steps	The steps.

Reimplemented in ScrollWheelBase.

7.186.4.5 getAnimationSteps()

```
uint16_t getAnimationSteps ( ) const
```

Gets animation steps as set in setAnimationSteps.

Returns

The animation steps.

See also

setAnimationSteps
setEasingEquation

7.186.4.6 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

7.186.4.7 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

7.186.4.8 getDrawableMargin()

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

Gets drawable margin as set through the argument in setDrawables().

Returns

The drawable margin.

7.186.4.9 getDrawableSize()

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

Gets drawable size as set through the first argument in setDrawables().

Returns

The drawable size.

See also

setDrawables

7.186.4.10 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

7.186.4.11 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

7.186.4.12 getNearestAlignedOffset()

```
int32_t getNearestAlignedOffset (
            int32_t offset ) const  [protected], [virtual]
```

Gets nearest offset aligned to a multiple of itemSize.

Parameters

offset	The offset.

Returns

The nearest aligned offset.

Reimplemented in ScrollList.

7.186.4.13 getNormalizedOffset()

```
int getNormalizedOffset (
            int offset ) const  [protected]
```

Gets normalized offset from a given offset from 0 down to -numItems∗itemSize.

Parameters

offset	The offset.

Returns

The normalized offset.

7.186.4.14 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

7.186.4.15 getOffset()

```
int32_t getOffset ( ) const  [protected], [virtual]
```

Gets display offset of first item.

Returns

The offset.

7.186.4.16 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

itemIndex	Zero-based index of the item.

Returns

The position for item.

Implemented in ScrollList, and ScrollWheelBase.

7.186.4.17 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

7.186.4.18 handleDragEvent()

```
void handleDragEvent (
             const DragEvent & evt )  [virtual]
```

Handles the drag event described by evt.

Parameters

evt	The event.

Reimplemented from Drawable.

Reimplemented in ScrollWheelBase.

7.186.4.19 handleGestureEvent()

```
void handleGestureEvent (
             const GestureEvent & evt )  [virtual]
```

Handles the gesture event described by evt.

Parameters

evt	The event.

Reimplemented from Drawable.

Reimplemented in ScrollWheelBase.

7.186.4.20 handleTickEvent()

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

Handles the tick event.

Reimplemented from Drawable.

7.186.4.21 initialize()

```
void initialize ( )  [inline], [virtual]
```

Initializes the contents of all drawables.

Reimplemented in ScrollWheelWithSelectionStyle.

7.186.4.22 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.

7.186.4.23 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

itemIndex	Zero-based index of the changed item.

Reimplemented in ScrollWheelWithSelectionStyle.

7.186.4.24 keepOffsetInsideLimits()

```
int32_t keepOffsetInsideLimits (
            int32_t newOffset,
            int16_t overShoot ) const  [protected], [pure virtual]
```

Parameters

newOffset	The new offset.
overShoot	The over shoot.

Returns

An int32_t.

Implemented in ScrollList, and ScrollWheelBase.

7.186.4.25 setAnimationEndedCallback()

```
void setAnimationEndedCallback (
            GenericCallback<> & callback )
```

Callback, called when the set animation ended.

Parameters

in,out	*callback*	The ended callback.

7.186.4.26 setAnimationSteps()

```
void setAnimationSteps (
            int16_t steps )
```

Sets animation steps when moving to a new selected item. The default value is 30.

Parameters

steps	The animation steps.

See also

setEasingEquation
getAnimationSteps

7.186.4.27 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

circular	True if the list should be circular, false if the list should not be circular.

See also

DrawableList::setCircular
getCircular

Reimplemented in ScrollWheelWithSelectionStyle.

7.186.4.28 setDragAcceleration()

```
void setDragAcceleration (
            uint16_t acceleration )
```

Sets drag acceleration times 10, so "10" means "1", "15" means "1.5".

Parameters

acceleration	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

7.186.4.29 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

drawableSize	The size of the drawable.
drawableMargin	The margin around drawables (margin before and margin after).

7.186.4.30 setEasingEquation()

```
void setEasingEquation (
            EasingEquation equation )
```

Sets easing equation when changing the selected item, for example via swipe or AnimateTo.

Parameters

equation	The equation.

See also

setAnimationSteps
getAnimationSteps

7.186.4.31 setHeight()

```
void setHeight (
            int16_t height ) [virtual]
```

Sets height of the ScrollBase.

Parameters

height	The height. The height is propagated to the child(ren).

Reimplemented from Drawable.

Reimplemented in ScrollWheelWithSelectionStyle.

7.186.4.32 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

horizontal	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.

7.186.4.33 setItemPressedCallback()

```
void setItemPressedCallback (
            GenericCallback< int16_t > & callback )
```

Set Callback which will be called when a item is pressed.

Parameters

in	*callback*	The callback.

7.186.4.34 setItemSelectedCallback()

```
void setItemSelectedCallback (
            GenericCallback< int16_t > & callback )
```

Sets Callback which will be called when the selected item is clicked.

Parameters

in	*callback*	The callback.

7.186.4.35 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

maxItems	The maximum items, 0 means "no limit".

See also

getMaxSwipeItems

7.186.4.36 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

items	Number of items.

Note

The DrawableList is refreshed to reflect the change.

Reimplemented in ScrollWheelWithSelectionStyle.

7.186.4.37 setOffset()

```
void setOffset (
            int32_t offset )  [protected], [virtual]
```

Sets display offset of first item.

Parameters

offset	The offset.

Reimplemented in ScrollWheelWithSelectionStyle.

7.186.4.38 setSwipeAcceleration()

```
void setSwipeAcceleration (
            uint16_t acceleration )
```

Sets swipe acceleration (times 10).

Parameters

acceleration	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

7.186.4.39 setWidth()

```
void setWidth (
              int16_t width ) [virtual]
```

Sets width of the ScrollBase. The width is propagated to the child(ren).

Parameters

width	The width.

Reimplemented from Drawable.

Reimplemented in ScrollWheelWithSelectionStyle.

7.186.4.40 stopAnimation()

```
void stopAnimation ( )
```

Stops an animation if one is ongoing.

7.187 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 stetting.
- 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

7.187.1 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.

7.187.2 Constructor & Destructor Documentation

7.187.2.1 ScrollList()

```
ScrollList ( )
```

Default constructor.

7.187.2.2 ~ScrollList()

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

Destructor.

7.187.3 Member Function Documentation

7.187.3.1 getItem()

```
int16_t getItem (
            int16_t drawableIndex )  [inline]
```

Gets an item.

Parameters

drawableIndex	Zero-based index of the drawable.

Returns

The item.

7.187.3.2 getNearestAlignedOffset()

```
int32_t getNearestAlignedOffset (
            int32_t offset ) const  [protected], [virtual]
```

Parameters

offset	The offset.

Returns

The nearest aligned offset.

Reimplemented from ScrollBase.

7.187.3.3 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

7.187.3.4 getPaddingBefore()

```
int16_t getPaddingBefore ( ) const
```

Gets distance before first drawable in ScrollList.

Returns

The distance before.

See also

setPadding
getPaddingAfter

7.187.3.5 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

itemIndex	Zero-based index of the item.

Returns

The position for item.

Implements ScrollBase.

7.187.3.6 getSnapping()

```
bool getSnapping ( ) const
```

Gets the current snap stetting.

Returns

true if snapping is set, false otherwise.

7.187.3.7 handleClickEvent()

```
void handleClickEvent (
            const ClickEvent & evt )  [virtual]
```

Parameters

evt	The event.

Reimplemented from Drawable.

7.187.3.8 keepOffsetInsideLimits()

```
int32_t keepOffsetInsideLimits (
            int32_t newOffset,
            int16_t overShoot ) const  [protected], [virtual]
```

Parameters

newOffset	The new offset.
overShoot	The over shoot.

Returns

An int32_t.

Implements ScrollBase.

7.187.3.9 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	*drawableListItems*	Number of drawables allocated.
in	*updateDrawableCallback*	A callback to update the contents of a drawable.

See also

DrawableList::setDrawables

7.187.3.10 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

distanceBefore	The distance before the first drawable in the ScrollList.
distanceAfter	The distance after the last drawable in the ScrollList.

See also

getPaddingBefore
getPaddingAfter

7.187.3.11 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

snap	true to snap.

7.187.3.12 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

items	The items.

Note

This only applies to non-circular lists.

7.188 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

7.188.1 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
ScrollWheelWithSelectionStyle

7.188.2 Constructor & Destructor Documentation

7.188.2.1 ScrollWheel()

```
ScrollWheel ( )
```

Default constructor.

7.188.2.2 ~ScrollWheel()

```
~ScrollWheel ( )  [virtual]
```

Destructor.

7.188.3 Member Function Documentation

7.188.3.1 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	*drawableListItems*	Number of drawables.
in,out	*updateDrawableCallback*	The update drawable callback.

7.189 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

7.189.1 Detailed Description

A scroll wheel base class. Used by ScrollWheel and ScrollWheelWithHighlight.

See also

ScrollWheel
ScrollWheelWithHighlight

7.189.2 Constructor & Destructor Documentation

7.189.2.1 ScrollWheelBase()

```
ScrollWheelBase ( )
```

Default constructor.

7.189.2.2 ~ScrollWheelBase()

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

Destructor.

7.189.3 Member Function Documentation

7.189.3.1 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

position	The new position.
steps	(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.

7.189.3.2 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

itemIndex	Zero-based index of the item.

Returns

The position for item.

Implements ScrollBase.

7.189.3.3 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.

7.189.3.4 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

7.189.3.5 handleClickEvent()

```
void handleClickEvent (
             const ClickEvent & evt )  [virtual]
```

Handles the click event described by evt.

Parameters

evt	The event.

Reimplemented from Drawable.

7.189.3.6 handleDragEvent()

```
void handleDragEvent (
             const DragEvent & evt )  [virtual]
```

Handles the drag event described by evt.

Parameters

evt	The event.

Reimplemented from ScrollBase.

7.189.3.7 handleGestureEvent()

```
void handleGestureEvent (
            const GestureEvent & evt )  [virtual]
```

Handles the gesture event described by evt.

Parameters

evt	The event.

Reimplemented from ScrollBase.

7.189.3.8 keepOffsetInsideLimits()

```
int32_t keepOffsetInsideLimits (
            int32_t newOffset,
            int16_t overShoot ) const  [virtual]
```

Parameters

newOffset	The new offset.
overShoot	The over shoot.

Returns

An int32_t.

Implements ScrollBase.

7.189.3.9 setAnimateToCallback()

```
void setAnimateToCallback (
            GenericCallback< int16_t > & callback )
```

Sets Callback which will be called when the ScrollWheel animates to a new item.

Parameters

in	*callback*	The callback.

7.189.3.10 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

offset	The offset.

Reimplemented in ScrollWheelWithSelectionStyle.

7.190 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 > * originalUpdateCenterDrawable↩ Callback

  The original update center drawable callback.

Additional Inherited Members

7.190.1 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

7.190.2 Constructor & Destructor Documentation

7.190.2.1 ScrollWheelWithSelectionStyle()

```
ScrollWheelWithSelectionStyle ( )
```

Default constructor.

7.190.2.2 ~ScrollWheelWithSelectionStyle()

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

Destructor.

7.190.3 Member Function Documentation

7.190.3.1 getSelectedItemExtraSizeAfter()

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

Gets selected item extra size after.

Returns

The selected item extra size after.

See also

setSelectedItemExtraSize

7.190.3.2 getSelectedItemExtraSizeBefore()

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

Gets selected item extra size before.

Returns

The selected item extra size before.

See also

setSelectedItemExtraSize

7.190.3.3 getSelectedItemMarginAfter()

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

Gets selected item margin after.

Returns

The selected item margin after.

See also

setSelectedItemMargin

7.190.3.4 getSelectedItemMarginBefore()

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

Gets selected item margin before.

Returns

The selected item margin before.

See also

setSelectedItemMargin

7.190.3.5 initialize()

```
void initialize ( )  [inline], [virtual]
```

Initializes the contents of all drawables.

Reimplemented from ScrollBase.

7.190.3.6 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

itemIndex	Zero-based index of the changed item.

Reimplemented from ScrollBase.

7.190.3.7 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 ScrollWheelWithSelectionStyle has been resized or the size of the selected item is changed.

7.190.3.8 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

circular	True to make the scroll wheel circular.

See also

getCircular

Reimplemented from ScrollBase.

7.190.3.9 setDrawables()

```
void setDrawables (
            DrawableListItemsInterface & drawableListItems,
            GenericCallback< DrawableListItemsInterface *, int16_t, int16_t > & updateDrawableCallback,
            DrawableListItemsInterface & centerDrawableListItems,
            GenericCallback< DrawableListItemsInterface *, int16_t, int16_t > & updateCenterDrawableCallback )  [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	*drawableListItems*	Number of drawables in outer array.
in	*updateDrawableCallback*	The callback to update a drawable.
in,out	*centerDrawableListItems*	Number of drawables in center array.
in	*updateCenterDrawableCallback*	The callback to update a center drawable.

7.190.3.10 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

drawableSize	Size of the drawable.
drawableMargin	The drawable margin.

See also

getDrawableSize
getDrawableMargin
getItemSize

7.190.3.11 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

height	The height.

Reimplemented from ScrollBase.

7.190.3.12 setHorizontal()

```
void setHorizontal (
            bool horizontal )  [virtual]
```

Sets whether the scroll wheel is horizontal or vertical. If the scroll wheel is hortizontal, the items are arranged side by side, otherwise they are arranged above and below each other.

Parameters

horizontal	True means horizontal, false means vertical.

See also

getHorizontal

Reimplemented from ScrollBase.

7.190.3.13 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

numberOfItems	Number of items.

See also

getNumberOfItems

Reimplemented from ScrollBase.

7.190.3.14 setOffset()

```
void setOffset (
            int32_t offset )  [protected], [virtual]
```

Sets offset of item 0 relative to the selected item's position.

Parameters

offset	The offset.

See also

getOffset

Reimplemented from ScrollBase.

7.190.3.15 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

extraSizeBefore	The extra size before.
extraSizeAfter	The extra size after.

See also

setSelectedItemOffset

7.190.3.16 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

marginBefore	The margin before.
marginAfter	The margin after.

See also

setSelectedItemOffset
setSelectedItemExtraSize

7.190.3.17 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

offset	The offset.

Reimplemented from ScrollWheelBase.

7.190.3.18 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

offset	The offset of the selected item.
extraSizeBefore	The extra size before the selected item.
extraSizeAfter	The extra size after the selected item.
marginBefore	The margin before the selected item.
marginAfter	The margin after the selected item.

See also

setSelectedItemOffset
setSelectedItemExtraSize
setSelectedItemMargin

7.190.3.19 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

width	The width.

Reimplemented from ScrollBase.

7.191 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.

7.191.1 Detailed Description

TouchController for the simulator.

See also

TouchController

7.191.2 Member Function Documentation

7.191.2.1 init()

```
void init ( ) [virtual]
```

Initializes touch controller.

Implements TouchController.

7.191.2.2 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.

7.192 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.

7.192.1 Detailed Description

TouchController for the simulator.

See also

TouchController

7.192.2 Member Function Documentation

7.192.2.1 init()

```
void init ( )  [virtual]
```

Initializes touch controller.

Implements TouchController.

7.192.2.2 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.

7.193 Shape< POINTS > Class Template Reference

Simple widget capable of drawing a shape.

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

Public Member Functions

- virtual ~Shape ()

  Virtual Destructor.
- virtual int getNumPoints () const

  Gets number points used to make up the shape.
- virtual void setCorner (int i, CWRUtil::Q5 x, CWRUtil::Q5 y)

  Sets a corner of the shape in Q5 format.
- virtual CWRUtil::Q5 getCornerX (int i) const

  Gets the x coordinate of a corner.
- virtual CWRUtil::Q5 getCornerY (int i) const

  Gets the y coordinate of a corner.

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

7.193.1 Detailed Description

template< uint16_t POINTS >
class touchgfx::Shape< POINTS >

Simple widget capable of drawing a shape. The shape can be scaled and rotated around 0,0. Note that the y axis goes down, so a shape that goes up must be given negative coordinates.

The Shape class allows the user to draw any shape and allows the defined shape to be scaled, rotated and moved freely. Example uses could be the hands of a clock (see the touchgfx_demo2014 for an actual implementation).

Template Parameters

POINTS	The number of points in the given shape.
T	The type of the points used for the shape. Must be int or float.

See also

CanvasWidget

7.193.2 Constructor & Destructor Documentation

7.193.2.1 ~Shape()

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

Virtual Destructor.

7.193.3 Member Function Documentation

7.193.3.1 getCacheX()

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

Gets cached x coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The cached x coordinate.

Implements AbstractShape.

7.193.3.2 getCacheY()

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

Gets cached y coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The cached y coordinate.

Implements AbstractShape.

7.193.3.3 getCornerX()

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

Gets the x coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The corner x coordinate.

Implements AbstractShape.

7.193.3.4 getCornerY()

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

Gets the y coordinate of a corner.

Parameters

i	Zero-based index of the corner.

Returns

The corner y coordinate.

Implements AbstractShape.

7.193.3.5 getNumPoints()

```
int getNumPoints ( ) const  [inline], [virtual]
```

Gets number points used to make up the shape.

Returns

The number points.

Implements AbstractShape.

7.193.3.6 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	Zero-based index of the corner.
x	The x coordinate.
y	The y coordinate.

Implements AbstractShape.

7.193.3.7 setCorner()

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

Sets a corner of the shape in Q5 format.

Parameters

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

Implements AbstractShape.

7.194 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.

7.194.1 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

T	Generic type parameter, either int or float.

See also

setAbstractShape()

7.195 SingleBlockAllocator< block_size, bytes_pr_pixel > Class Template Reference

This class is partial framebuffer allocator using just one block.

`#include <touchgfx/hal/FrameBufferAllocator.hpp>`

Public Member Functions

- virtual uint16_t allocateBlock (const uint16_t x, const uint16_t y, const uint16_t width, const uint16_t height, uint8_t **block)

  Allocates a framebuffer block.
- virtual void markBlockReadyForTransfer ()

  Marks a previously allocated block as ready to be transferred to the LCD.
- virtual bool hasBlockReadyForTransfer ()

  Check if a block is ready for transfer to the LCD.
- virtual const uint8_t * getBlockForTransfer (Rect &rect)

  Get the block ready for transfer.
- virtual void freeBlockAfterTransfer ()

  Free a block after transfer to the LCD.

7.195.1 Detailed Description

template<uint16_t block_size, uint32_t bytes_pr_pixel>
class touchgfx::SingleBlockAllocator< block_size, bytes_pr_pixel >

This class is partial framebuffer allocator using just one block. No new buffer can be allocated until the block has been transferred to LCD.

See also

ManyBlockAllocator

7.195.2 Member Function Documentation

7.195.2.1 allocateBlock()

```
uint16_t allocateBlock (
            const uint16_t x,
            const uint16_t y,
            const uint16_t width,
            const uint16_t height,
            uint8_t ** block )  [inline], [virtual]
```

Allocates a framebuffer block. The block will have at least the width requested. The height of the allocated block can be lower than requested if not enough memory is available. This class calls FrameBufferAllocatorWaitOnTransfer() if no block is available.

Parameters

x	The absolute x coordinate of the block on the screen.
y	The absolute y coordinate of the block on the screen.
width	The width of the block.
height	The height of the block.
block	Pointer to pointer to return the block address in.

Returns

The height of the allocated block.

Implements FrameBufferAllocator.

7.195.2.2 freeBlockAfterTransfer()

```
void freeBlockAfterTransfer ( )  [inline], [virtual]
```

Marks a previously allocated block as transferred and ready to reuse.

Implements FrameBufferAllocator.

7.195.2.3 getBlockForTransfer()

```
const uint8_t * getBlockForTransfer (
            Rect & rect )  [inline], [virtual]
```

Get the block ready for transfer.

Parameters

rect	Reference to rect to write block x, y, width, and height.

Returns

Returns the address of the block ready for transfer.

Implements FrameBufferAllocator.

7.195.2.4 hasBlockReadyForTransfer()

```
bool hasBlockReadyForTransfer ( )  [inline], [virtual]
```

Check if a block is ready for transfer to the LCD.

Returns

True if a block is ready for transfer.

Implements FrameBufferAllocator.

7.195.2.5 markBlockReadyForTransfer()

```
void markBlockReadyForTransfer ( )  [inline], [virtual]
```

Marks a previously allocated block as ready to be transferred to the LCD.

Implements FrameBufferAllocator.

7.196 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 getVisiblePixelsWhenCollapsed () 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

7.196.1 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.

7.196.2 Member Function Documentation

7.196.2.1 add()

```
void add (
            Drawable & d )  [virtual]
```

Parameters

d	The drawable to add.

Reimplemented from Container.

7.196.2.2 animateToState()

```
void animateToState (
            SlideMenu::State newState )  [virtual]
```

Parameters

newState	The new state of the SlideMenu.

7.196.2.3 animationEndedHandler()

```
void animationEndedHandler (
            const MoveAnimator< Container > & container )  [protected]
```

Parameters

container	The menuContainer.

7.196.2.4 getAnimationDuration()

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

Returns

The animation duration.

7.196.2.5 getAnimationEasingEquation()

```
EasingEquation getAnimationEasingEquation ( ) const  [virtual]
```

Returns

The animation easing equation.

7.196.2.6 getBackgroundX()

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

Returns

The background x coordinate.

7.196.2.7 getBackgroundY()

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

Returns

The background y coordinate.

7.196.2.8 getCollapsedXCoordinate()

```
int16_t getCollapsedXCoordinate ( )  [protected], [virtual]
```

Returns

The collapsed x coordinate.

7.196.2.9 getCollapsedYCoordinate()

```
int16_t getCollapsedYCoordinate ( )  [protected], [virtual]
```

Returns

The collapsed y coordinate.

7.196.2.10 getExpandDirection()

```
SlideMenu::ExpandDirection getExpandDirection ( ) const  [virtual]
```

Returns

The expand direction.

7.196.2.11 getExpandedStateTimeout()

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

Returns

The expanded state timeout.

7.196.2.12 getExpandedStateTimer()

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

Returns

The expanded state timer.

7.196.2.13 getExpandedXCoordinate()

```
int16_t getExpandedXCoordinate ( )  [protected], [virtual]
```

Returns

The expanded x coordinate.

7.196.2.14 getExpandedYCoordinate()

```
int16_t getExpandedYCoordinate ( )  [protected], [virtual]
```

Returns

The expanded y coordinate.

7.196.2.15 getHiddenPixelsWhenExpanded()

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

Returns

The hidden pixels when expanded.

7.196.2.16 getState()

```
SlideMenu::State getState ( )  [virtual]
```

Returns

The current state.

7.196.2.17 getStateChangeButtonX()

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

Returns

The state change button x coordinate.

7.196.2.18 getStateChangeButtonY()

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

Returns

The state change button y coordinate.

7.196.2.19 getVisiblePixelsWhenCollapsed()

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

Returns

The visible pixels when collapsed.

7.196.2.20 remove()

```
void remove (
            Drawable & d )  [virtual]
```

Parameters

d	The drawable to remove.

Reimplemented from Container.

7.196.2.21 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.

7.196.2.22 setAnimationDuration()

```
void setAnimationDuration (
            uint16_t duration )  [virtual]
```

Parameters

duration	The animation duration.

7.196.2.23 setAnimationEasingEquation()

```
void setAnimationEasingEquation (
            EasingEquation animationEasingEquation )  [virtual]
```

Parameters

animationEasingEquation	The animation easing equation.

7.196.2.24 setExpandDirection()

```
void setExpandDirection (
            SlideMenu::ExpandDirection newExpandDirection )  [virtual]
```

Parameters

newExpandDirection	The new expand direction.

7.196.2.25 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 reset↩ ExpandedStateTimer method.

Parameters

timeout	The timeout in ticks.

7.196.2.26 setHiddenPixelsWhenExpanded()

```
void setHiddenPixelsWhenExpanded (
            int16_t hiddenPixels )  [virtual]
```

Parameters

hiddenPixels	The hidden pixels.

7.196.2.27 setState()

```
void setState (
            SlideMenu::State newState )  [virtual]
```

Parameters

newState	The new state of the SlideMenu.

7.196.2.28 setStateChangedAnimationEndedCallback()

```
void setStateChangedAnimationEndedCallback (
            GenericCallback< const SlideMenu & > & callback )  [virtual]
```

Parameters

callback	The callback.

7.196.2.29 setStateChangedCallback()

```
void setStateChangedCallback (
            GenericCallback< const SlideMenu & > & callback )  [virtual]
```

Parameters

callback	The callback.

7.196.2.30 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, hiddenPixels↩ WhenExpanded = 0, animationDuration = 10, animationEquation = cubicEaseInOut.

Parameters

newExpandDirection	The new expand direction.
backgroundBMP	The background bitmap.
stateChangeButtonBMP	The state change button bitmap.
stateChangeButtonPressedBMP	The state change button pressed bitmap.

7.196.2.31 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, visiblePixelsWhen↩ Collapsed = 0, hiddenPixelsWhenExpanded = 0, animationDuration = 10, animationEquation = cubicEaseInOut.

Parameters

newExpandDirection	The new expand direction.
backgroundBMP	The background bitmap.
stateChangeButtonBMP	The state change button bitmap.
stateChangeButtonPressedBMP	The state change button pressed bitmap.
backgroundX	The background x coordinate.
backgroundY	The background y coordinate.
stateChangeButtonX	The state change button x coordinate.
stateChangeButtonY	The state change button y coordinate.

7.196.2.32 setVisiblePixelsWhenCollapsed()

```
void setVisiblePixelsWhenCollapsed (
             int16_t visiblePixels )  [virtual]
```

Parameters

visiblePixels	The visible pixels.

7.196.2.33 stateChangeButtonClickedHandler()

```
void stateChangeButtonClickedHandler (
            const AbstractButton & button )  [protected]
```

Parameters

button	The state change button.

7.197 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

7.197.1 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.

7.197.2 Constructor & Destructor Documentation

7.197.2.1 Slider()

```
Slider ( )
```

Default constructor. Set the value range to default 0-100.

7.197.3 Member Function Documentation

7.197.3.1 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

7.197.3.2 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

7.197.3.3 getIndicatorPositionRangeSize()

```
int getIndicatorPositionRangeSize ( ) const  [protected], [virtual]
```

Returns

The indicator position range size.

7.197.3.4 getIndicatorRadius()

```
uint16_t getIndicatorRadius ( ) const  [protected], [virtual]
```

Returns

The the indicator radius.

7.197.3.5 getMaxValue()

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

Gets the maximum value previously set using setValueRange.

Returns

The maximum value.

See also

setValueRange
getMinValue

7.197.3.6 getMinValue()

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

Gets the minimum value previously set using setValueRange.

Returns

The minimum value.

See also

setValueRange
getMaxValue

7.197.3.7 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_BUTTON.

Reimplemented from Container.

7.197.3.8 getValue()

```
int getValue ( )  [inline]
```

Gets the current value represented by the indicator.

Returns

The current value.

7.197.3.9 getValueRangeSize()

```
int getValueRangeSize ( ) const  [protected], [virtual]
```

Returns

The value range size.

7.197.3.10 handleClickEvent()

```
void handleClickEvent (
             const ClickEvent & evt )  [protected], [virtual]
```

Parameters

evt	The event.

Reimplemented from Drawable.

7.197.3.11 handleDragEvent()

```
void handleDragEvent (
             const DragEvent & evt )  [protected], [virtual]
```

Parameters

evt	The event.

Reimplemented from Drawable.

7.197.3.12 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

position	The position.

Returns

The value that corresponds to the coordinate.

7.197.3.13 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 sliderBackground↩ Selected.

Parameters

sliderBackground	The slider background with the slider range unselected.
sliderBackgroundSelected	The slider background with the slider range selected.
indicator	The indicator.

7.197.3.14 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 sliderBackground↩ Selected.

Parameters

sliderBackground	The slider background.
sliderBackgroundSelected	The slider background selected.
indicator	The indicator.

7.197.3.15 setNewValueCallback()

```
void setNewValueCallback (
            GenericCallback< const Slider &, int > & callback )  [inline]
```

Associates an action to be performed when the slider changes its value.

Parameters

callback	The callback to be executed. The callback will be given a reference to the Slider and the current value of the slider.

See also

GenericCallback

7.197.3.16 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

callback	The callback to be executed. The callback will be given a reference to the Slider and the current value of the slider at interaction start.

See also

GenericCallback

7.197.3.17 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

callback	The callback to be executed. The callback will be given a reference to the Slider and the current value of the slider at interaction end.

See also

GenericCallback

7.197.3.18 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

backgroundX	The background x coordinate inside the slider.
backgroundY	The background y coordinate inside the slider.
indicatorY	The indicator y coordinate inside the slider.
indicatorMinX	The indicator minimum x coordinate inside the slider. This is the position used when the slider is at its minimum value. Must be less than indicatorMaxX.
indicatorMaxX	The indicator maximum x coordinate inside the slider. This is the position used when the slider is at its maximum value. Must be greater than indicatorMinX.

7.197.3.19 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

backgroundX	The background x coordinate inside the slider.
backgroundY	The background y coordinate inside the slider.
indicatorX	The indicator x coordinate inside the slider.
indicatorMinY	The indicator minimum y coordinate inside the slider. This is the position used when the slider is at its maximum value. Must be less than indicatorMaxX.
indicatorMaxY	The indicator maximum y coordinate inside the slider. This is the position used when the slider is at its minimum value. Must be greater than indicatorMinX.

7.197.3.20 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

value	The value.

7.197.3.21 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

minValue	The minimum value. Must be less than maxValue.
maxValue	The maximum value. Must be greater than minValue.
newValue	The new value.

7.197.3.22 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

minValue	The minimum value. Must be less than maxValue.
maxValue	The maximum value. Must be greater than minValue.

7.197.3.23 updateIndicatorPosition()

```
void updateIndicatorPosition (
             int16_t position )  [protected], [virtual]
```

Updates the indicator position described by position. Calls the newSliderValueCallback with the new value.

Parameters

position	The position (x coordinate in horizontal mode and y coordinate in vertical mode).

7.197.3.24 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

value	The value.

Returns

The coordinate that corresponds to the value.

7.198 SlideTransition< templateDirection > Class Template Reference

A Transition that slides from one screen to the next.

`#include <touchgfx/transitions/SlideTransition.hpp>`

Public Member Functions

- SlideTransition (const uint8_t transitionSteps=20)

  Constructor.
- virtual ~SlideTransition ()

  Destructor.
- virtual void handleTickEvent ()

  Handles the tick event when transitioning.
- virtual void tearDown ()

  Tear down.
- virtual void init ()

  Initializes this object.

Protected Member Functions

- virtual void initMoveDrawable (Drawable &d)

  Moves the Drawable to its initial position.
- virtual void tickMoveDrawable (Drawable &d)

  Moves the Drawable.

Protected Attributes

- SnapshotWidget snapshot

  The SnapshotWidget that is moved when transitioning.
- SnapshotWidget * snapshotPtr

  Pointer pointing to the snapshot used in this transition.The snapshot pointer.

7.198.1 Detailed Description

template<Direction templateDirection>
class touchgfx::SlideTransition< templateDirection >

A Transition that slides from one screen to the next. It does so by moving a SnapShotWidget with a snapshot of the Screen transitioning away from, and by moving the contents of Screen transitioning to.

Template Parameters

templateDirection	Type of the template direction.

See also

Transition

7.198.2 Constructor & Destructor Documentation

7.198.2.1 SlideTransition()

```
SlideTransition (
            const uint8_t transitionSteps = 20 )  [inline]
```

Constructor.

Parameters

transitionSteps	Number of steps in the transition animation.

7.198.2.2 ~SlideTransition()

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

Destructor.

7.198.3 Member Function Documentation

7.198.3.1 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.

7.198.3.2 init()

```
void init ( )  [inline], [virtual]
```

Initializes this object.

See also

Transition::init()

Reimplemented from Transition.

7.198.3.3 initMoveDrawable()

```
void initMoveDrawable (
            Drawable & d )  [inline], [protected], [virtual]
```

Moves the Drawable to its initial position.

Parameters

in	*d*	The Drawable to move.

7.198.3.4 tearDown()

```
void tearDown ( )  [inline], [virtual]
```

Tear down.

See also

Transition::teadDown()

Reimplemented from Transition.

7.198.3.5 tickMoveDrawable()

```
void tickMoveDrawable (
            Drawable & d )  [inline], [protected], [virtual]
```

Moves the Drawable.

Parameters

in	*d*	The Drawable to move.

7.199 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.

7.199.1 Detailed Description

template<class T>
class touchgfx::Snapper< T >

A mix-in that will make class T draggable and able to snap to a position when a drag operation has ended. The mix-in is able to perform callbacks when the snapper gets dragged and when the Snapper snaps to its snap position.

Template Parameters

T	specifies the type to enable the Snap behavior to.

See also

Draggable<T>

7.199.2 Constructor & Destructor Documentation

7.199.2.1 Snapper()

```
Snapper ( )  [inline]
```

Default constructor.

7.199.2.2 ~Snapper()

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

Destructor.

7.199.3 Member Function Documentation

7.199.3.1 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

evt	The click event.

7.199.3.2 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

evt	The drag event.

Reimplemented from Draggable< T >.

7.199.3.3 setDragAction()

```
void setDragAction (
            GenericCallback< const DragEvent & > & callback )  [inline]
```

Associates an action to be performed when the Snapper is dragged.

Parameters

callback	The callback will be executed with the DragEvent.

See also

GenericCallback

7.199.3.4 setSnappedAction()

```
void setSnappedAction (
            GenericCallback<> & callback )  [inline]
```

Associates an action to be performed when the Snapper is snapped.

Parameters

in	*callback*	The callback to be executed on snap.

See also

GenericCallback

7.199.3.5 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

x	The x coordinate.
y	The y coordinate.

7.200 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

- BitmapId bitmapId

  BitmapId where copy is stored s copied to.
- uint8_t alpha

  The alpha with which to draw this snapshot.

Additional Inherited Members

7.200.1 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

7.200.2 Constructor & Destructor Documentation

7.200.2.1 SnapshotWidget()

```
SnapshotWidget ( )
```

Default constructor.

7.200.2.2 ~SnapshotWidget()

```
~SnapshotWidget ( )  [virtual]
```

Destructor.

7.200.3 Member Function Documentation

7.200.3.1 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

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Implements Drawable.

7.200.3.2 getAlpha()

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

Gets the current alpha value.

Returns

The alpha value.

7.200.3.3 getSolidRect()

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

Gets solid rectangle.

Returns

The solid rectangle.

Implements Drawable.

7.200.3.4 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_SNAPSHOTWIDGET.

Reimplemented from Widget.

7.200.3.5 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.

7.200.3.6 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

bmp	The target dynamic bitmap.

7.200.3.7 setAlpha()

```
void setAlpha (
```

```
const uint8_t a )  [inline]
```

Sets the alpha value.

Parameters

a	The alpha value.

7.201 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, int16_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).
- int16_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.

7.201.1 Detailed Description

The visual elements when writing a string.

7.201.2 Constructor & Destructor Documentation

7.201.2.1 StringVisuals() [1/2]

```
StringVisuals ( )  [inline]
```

Construct an empty StringVisuals object.

7.201.2.2 StringVisuals() [2/2]

```
StringVisuals (
            const Font * font,
            colortype color,
            uint8_t alpha,
            Alignment alignment,
            int16_t linespace,
            TextRotation rotation,
            TextDirection textDirection,
            uint8_t indentation,
            WideTextAction wideTextAction = WIDE_TEXT_NONE )  [inline]
```

Parameters

font	The Font with which to draw the text.
color	The color with which to draw the text.
alpha	Alpha blending. Default value is 255 (solid)
alignment	How to align the text.
linespace	Line space in pixels between each line, in case the text contains newline characters.
rotation	How to rotate the text.
textDirection	The text direction.
indentation	The indentation of the text from the left and right of the text area rectangle.
wideTextAction	What to do with lines longer than the width of the TextArea.

7.202 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 &highlighted↩ Page)

  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

7.202.1 Detailed Description

See also

touchgfx::Container.

7.202.2 Member Function Documentation

7.202.2.1 add()

```
void add (
            touchgfx::Drawable & page )  [virtual]
```

Adds a page to the container.

Parameters

in,out	*page*	The page to add.

Note

All pages must have the same width and height.

Reimplemented from Container.

7.202.2.2 getNumberOfPages()

```
uint8_t getNumberOfPages ( )  [inline]
```

Gets number of pages.

Returns

The number of pages.

7.202.2.3 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

evt	The ClickEvent received from the HAL.

Reimplemented from Drawable.

7.202.2.4 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

evt	The DragEvent received from the HAL.

Reimplemented from Drawable.

7.202.2.5 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

evt	The GestureEvent received from the HAL.

Reimplemented from Drawable.

7.202.2.6 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.

7.202.2.7 remove()

```
void remove (
             Drawable & page )  [virtual]
```

Removes the page from the container.

Parameters

in,out	*page*	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.

7.202.2.8 setEndSwipeElasticWidth()

```
void setEndSwipeElasticWidth (
             uint16_t width )
```

Parameters

width	The width in pixels.

7.202.2.9 setPageIndicatorBitmaps()

```
void setPageIndicatorBitmaps (
             const touchgfx::Bitmap & normalPage,
             const touchgfx::Bitmap & highlightedPage )
```

Sets the bitmaps that are used by the page indicator.

Parameters

normalPage	The normal page.
highlightedPage	The highlighted page.

7.202.2.10 setPageIndicatorXY()

```
void setPageIndicatorXY (
```

```
            int16_t x,
            int16_t y )
```

Parameters

x	The x coordinate.
y	The y coordinate.

7.202.2.11 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

x	The center x coordinate.
y	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.

7.202.2.12 setSelectedPage()

```
void setSelectedPage (
            uint8_t pageIndex )
```

Sets the selected page.

Parameters

pageIndex	Zero-based index of the page. Range from 0 to numberOfPages-1.

7.202.2.13 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

cutoff	The cutoff in pixels.

7.203 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 (int16_t space)

  Sets the line spacing of the TextArea.
- int16_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)

  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 resizeToCurrentTextWithAlignment ()

  Sets the dimensions of the TextArea.
- void resizeHeightToCurrentText ()

Sets the height of the TextArea.

- void setWideTextAction (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.
- colortype color

  The color to use.
- int16_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

7.203.1 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.

7.203.2 Constructor & Destructor Documentation

7.203.2.1 TextArea()

```
TextArea ( )  [inline]
```

Create an empty TextArea. Default color is black.

7.203.3 Member Function Documentation

7.203.3.1 draw()

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

Draws the text. Called automatically.

Parameters

area	The invalidated area.

Implements Drawable.

Reimplemented in TextAreaWithTwoWildcards, and TextAreaWithOneWildcard.

7.203.3.2 getAlpha()

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

Gets the alpha value of the text.

Returns

The alpha value. 255 = completely solid. 0 = invisible.

7.203.3.3 getColor()

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

Gets the color of the text.

Returns

The color to used for drawing the text.

7.203.3.4 getIndentation()

```
uint8_t getIndentation ( )  [inline]
```

Gets the indentation.

Returns

The indentation.

See also

setIndetation

7.203.3.5 getLinespacing()

```
int16_t getLinespacing ( ) const  [inline]
```

Gets the line spacing of the TextArea.

Returns

The line spacing.

7.203.3.6 getRotation()

```
TextRotation getRotation ( ) const  [inline]
```

Gets rotation of the text in the TextArea.

Returns

The rotation of the text.

7.203.3.7 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.

7.203.3.8 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.

7.203.3.9 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.

7.203.3.10 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.

7.203.3.11 getTypedText()

```
TypedText getTypedText ( ) const  [inline]
```

Gets the TypedText of the text area.

Returns

The currently used TypedText.

7.203.3.12 getWideTextAction()

```
WideTextAction getWideTextAction ( ) const  [inline]
```

Gets wide text action preciously set using setWideTextAction.

Returns

current WideTextAction setting.

See also

setWideTextAction

7.203.3.13 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

7.203.3.14 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

7.203.3.15 resizeToCurrentTextWithAlignment()

```
void resizeToCurrentTextWithAlignment ( )
```

Sets the dimensions of the TextArea to match the width and height of the current associated text for the current selected language.

When setting the width, the position of the TextArea might be changed in order to keep the text centered or right aligned.

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

7.203.3.16 setAlpha()

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

Sets the alpha value of the text.

Parameters

alpha	The alpha value. 255 = completely solid. 0 = invisible.

7.203.3.17 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

baselineY	The y coordinate of the baseline.

7.203.3.18 setColor()

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

Sets the color of the text.

Parameters

color	The color to use.

7.203.3.19 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

indent	The indentation from left (when left aligned text) and right (when right aligned text).

See also

getMaxPixelsLeft

7.203.3.20 setLinespacing()

```
void setLinespacing (
            int16_t space )  [inline]
```

Sets the line spacing of the TextArea.

Parameters

space	The line spacing of use in the TextArea.

7.203.3.21 setRotation()

```
void setRotation (
            const TextRotation rotation )  [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

rotation	The rotation of the text.

7.203.3.22 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

t	The TypedText for this widget to display.

7.203.3.23 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

action	The action to perform for wide lines of text.

See also

WideTextAction
getWideTextAction
resizeHeightToCurrentText

7.203.3.24 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

x	The x coordinate of the TextArea.
baselineY	The y coordinate of the baseline.

7.204 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

7.204.1 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

7.204.2 Constructor & Destructor Documentation

7.204.2.1 TextAreaWithOneWildcard()

```
TextAreaWithOneWildcard ( )  [inline]
```

Create an empty text area.

Note

No text can be displayed until a font is set. Default color is black.

7.204.3 Member Function Documentation

7.204.3.1 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

area	The invalidated area.

Reimplemented from TextArea.

7.204.3.2 getTextHeight()

```
int16_t getTextHeight ( )  [inline], [virtual]
```

Gets text height.

Returns

The text height.

Reimplemented from TextArea.

7.204.3.3 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.

7.204.3.4 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TEXTAREAWITHONEWILDCARD.

Reimplemented from TextArea.

7.204.3.5 getWildcard()

```
const Unicode::UnicodeChar * getWildcard ( ) const  [inline]
```

Gets the wildcard in the text.

Returns

The wildcard used in the text.

7.204.3.6 setWildcard()

```
void setWildcard (
            const Unicode::UnicodeChar * value )  [inline]
```

Sets the wildcard in the text.Must be a zero-terminated UnicodeChar array.

Parameters

value	A pointer to the UnicodeChar to set the wildcard to.

7.205 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

7.205.1 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

7.205.2 Constructor & Destructor Documentation

7.205.2.1 TextAreaWithTwoWildcards()

```
TextAreaWithTwoWildcards ( )  [inline]
```

Create an empty text area.

Note

No text can be displayed until a font is set. Default color is black.

7.205.3 Member Function Documentation

7.205.3.1 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

area	The invalidated area.

Reimplemented from TextArea.

7.205.3.2 getTextHeight()

```
int16_t getTextHeight ( )  [inline], [virtual]
```

Gets text height.

Returns

The text height.

Reimplemented from TextArea.

7.205.3.3 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.

7.205.3.4 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TEXTAREAWITHTWOWILDCARDS.

Reimplemented from TextArea.

7.205.3.5 getWildcard1()

```
const Unicode::UnicodeChar * getWildcard1 ( ) const  [inline]
```

Gets the first wildcard in the text.

Returns

The first wildcard from a TextArea with two wildcards.

7.205.3.6 getWildcard2()

```
const Unicode::UnicodeChar * getWildcard2 ( ) const  [inline]
```

Gets the second wildcard in the text.

Returns

The second wildcard from a TextArea with two wildcards.

7.205.3.7 setWildcard1()

```
void setWildcard1 (
            const Unicode::UnicodeChar * value )  [inline]
```

Sets the first wildcard in the text. Must be a zero-terminated UnicodeChar array.

Parameters

value	A pointer to the UnicodeChar to set the wildcard to.

7.205.3.8 setWildcard2()

```
void setWildcard2 (
            const Unicode::UnicodeChar * value )  [inline]
```

Sets the second wildcard in the text. Must be a zero-terminated UnicodeChar array.

Parameters

value	A pointer to the UnicodeChar to set the wildcard to.

7.206 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

7.206.1 Detailed Description

Base class for TextAreas displaying texts with one or more wildcards.

See also

TextAreaWithOneWildcard
TextAreaWithTwoWildcards

7.206.2 Constructor & Destructor Documentation

7.206.2.1 TextAreaWithWildcardBase()

```
TextAreaWithWildcardBase ( )  [inline]
```

Create an empty text area.

Note

No text can be displayed until a font is set. Default color is black.

7.206.3 Member Function Documentation

7.206.3.1 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

format	The text containing s wildcards.
...	Variable arguments providing additional information.

Returns

the total height needed by the text.

7.207 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.

7.207.1 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

T	Generic type parameter. Typically a AbstractButtonContainer subclass.

See also

AbstractButtonContainer

7.207.2 Member Function Documentation

7.207.2.1 setText()

```
void setText (
             TypedText t )  [inline]
```

Parameters

t	A TypedText to process.

7.207.2.2 setTextColors()

```
void setTextColors (
             colortype newColorReleased,
             colortype newColorPressed )  [inline]
```

Parameters

newColorReleased	The new color released.
newColorPressed	The new color pressed.

7.207.2.3 setTextPosition()

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

Parameters

x	The x coordinate.

Parameters

y	The y coordinate.
width	The width of the text.
height	The height of the text.

7.207.2.4 setTextRotation()

```
void setTextRotation (
            TextRotation rotation )  [inline]
```

Parameters

rotation	The rotation.

7.207.2.5 setTextX()

```
void setTextX (
            int16_t x )  [inline]
```

Parameters

x	The x coordinate.

7.207.2.6 setTextXY()

```
void setTextXY (
            int16_t x,
            int16_t y )  [inline]
```

Parameters

x	The x coordinate.
y	The y coordinate.

7.207.2.7 setTextY()

```
void setTextY (
            int16_t y )  [inline]
```

Parameters

y	The y coordinate.

7.208 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

7.208.1 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.

7.208.2 Constructor & Destructor Documentation

7.208.2.1 TextProgress()

```
TextProgress ( )
```

Default constructor.

7.208.2.2 ~TextProgress()

```
~TextProgress ( )  [virtual]
```

Destructor.

7.208.3 Member Function Documentation

7.208.3.1 getAlpha()

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

Gets the alpha of the text area.

Returns

The alpha.

See also

setAlpha
TextArea::getAlpha

7.208.3.2 getColor()

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

Gets the color of the text in the used text area.

Returns

The color.

7.208.3.3 getNumberOfDecimals()

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

Gets number of decimals.

Returns

The number of decimals.

See also

setNumberOfDecimals

7.208.3.4 getTypedText()

```
TypedText getTypedText ( ) const  [virtual]
```

Gets the typed text.

Returns

The typed text.

See also

setTypedText

7.208.3.5 setAlpha()

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

Sets the alpha of the text area.

Parameters

alpha	The alpha.

See also

getAlpha
TextArea::setAlpha

7.208.3.6 setColor()

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

Sets the color of the text in the used text area.

Parameters

color	The color.

See also

getColor
TextArea::setColor

7.208.3.7 setNumberOfDecimals()

```
void setNumberOfDecimals (
            uint16_t numberOfDecimals ) [virtual]
```

Sets number of decimals when displaying progress.

Parameters

numberOfDecimals	Number of decimals. Only up to two decimals is supported.

See also

getNumberOfDecimals

7.208.3.8 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

x	The x coordinate.
y	The y coordinate.
width	The width of the text progress indicator.
height	The height of the text progress indicator.

Reimplemented from AbstractProgressIndicator.

7.208.3.9 setTypedText()

```
void setTypedText (
            const TypedText & t ) [virtual]
```

Sets the typed text. The text should have one wildcard and could for example "< progress>%".

Parameters

t	The TypedText to process.

See also

getTypedText

7.208.3.10 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

value	The value.

Reimplemented from AbstractProgressIndicator.

7.209 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 *stringFormat, va_list pArg, const uint16_t *gsubTable=0)
  Initializes the TextProvider.
- Unicode::UnicodeChar getNextChar ()
  Gets the next character.
- Unicode::UnicodeChar getNextLigature (TextDirection direction)
  Gets the next ligature.
- Unicode::UnicodeChar getNextLigature (TextDirection direction, const Font *font, const GlyphNode *&glyph)
  Gets the next ligature.
- Unicode::UnicodeChar getNextLigature (TextDirection direction, const Font *font, const GlyphNode *&glyph, const uint8_t *&pixelData, uint8_t &bitsPerPixel)
  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.

7.209.1 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.

7.209.2 Constructor & Destructor Documentation

7.209.2.1 TextProvider()

```
TextProvider ( )
```

Empty constructor. The user must call initialize() before characters can be provided.

7.209.3 Member Function Documentation

7.209.3.1 getNextChar()

```
Unicode::UnicodeChar getNextChar ( )
```

Gets the next character. For Arabic and Thai, it is important to use the getNextLigature instead.

Returns

The next character of the expanded string or 0 if end of string is reached.

See also

TextProvider::getNextLigature()

7.209.3.2 getNextLigature() [1/3]

```
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. Thai character placement might also depend on previous characters. It is recommended to use getNextLigature with font and glyph parameters to ensure coming glyphs in a text are placed correctly.

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

direction	The direction.

Returns

The next character of the expanded string or 0 if end of string is reached.

See also

TextProvider::getNextChar()

7.209.3.3 getNextLigature() [2/3]

```
Unicode::UnicodeChar getNextLigature (
            TextDirection direction,
            const Font * font,
            const GlyphNode *& glyph )
```

Gets the next ligature. For most languages this is the same as getNextChar() but eg. Arabic has different ligatures for each character.

Also gets a glyph for the ligature in a font. For non-Thai unicodes, this is identical to using Font::getGlyph(), but for Thai characters where diacritics glyphs are not always placed at the same relative position, an adjusted GlyphNode will be generated with correct relative X/Y coordinates.

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

	direction	The direction.
	font	The font.
out	*glyph*	The glyph.

Returns

The next character of the expanded string or 0 if end of string i reached.

See also

TextProvider::getNextChar()
Font::getGlyph

7.209.3.4 getNextLigature() [3/3]

```
Unicode::UnicodeChar getNextLigature (
            TextDirection direction,
            const Font * font,
            const GlyphNode *& glyph,
            const uint8_t *& pixelData,
            uint8_t & bitsPerPixel )
```

Gets the next ligature. For most languages this is the same as getNextChar() but eg. Arabic has different ligatures for each character.

Also gets a glyph for the ligature in a font. For non-Thai unicodes, this is identical to using Font::getGlyph(), but for Thai characters where diacritics glyphs are not always placed at the same relative position, an adjusted GlyphNode will be generated with correct relative X/Y coordinates.

Furthermore a pointer to the glyph data and the bit depth of the font are returned in parameters.

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

	direction	The direction.
	font	The font.
out	*glyph*	The glyph.
out	*pixelData*	Information describing the pixel.
out	*bitsPerPixel*	The bits per pixel.

Returns

The next character of the expanded string or 0 if end of string is reached.

See also

TextProvider::getNextChar()
Font::getGlyph

7.209.3.5 initialize()

```
void initialize (
             const Unicode::UnicodeChar * stringFormat,
             va_list pArg,
             const uint16_t * gsubTable = 0 )
```

Initializes the TextProvider. Each '\2' character in the format is replaced by one UnicodeChar* argument from pArg.

Parameters

stringFormat	The string to format.
pArg	Format arguments in the form of a va_list.
gsubTable	Pointer to GSUB table with unicode substitution rules.

7.210 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.
- static void setTranslation (touchgfx::LanguageId id, const void *translation)

  Adds or replaces a translation.

7.210.1 Detailed Description

Class for setting language and getting texts. The language set will determine which texts will be used.

7.210.2 Member Function Documentation

7.210.2.1 getLanguage()

```
static LanguageId getLanguage ( )  [inline], [static]
```

Gets the current language.

Returns

The id of the language.

7.210.2.2 getText()

```
const Unicode::UnicodeChar * getText (
            TypedTextId id ) const
```

Get text in the set language.

Parameters

id	The id of the text to lookup.

Returns

The text.

7.210.2.3 setLanguage()

```
static void setLanguage (
            LanguageId id )  [static]
```

Sets the current language for texts.

Parameters

id	The id of the language.

7.210.2.4 setTranslation()

```
static void setTranslation (
            touchgfx::LanguageId id,
            const void * translation )  [static]
```

Adds or replaces a translation. This function allows an application to add a translation at runtime.

Parameters

id	The id of the language to add or replace.
translation	A pointer to the translation in flash or RAM.

7.211 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.

7.211.1 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

7.211.2 Member Enumeration Documentation

7.211.2.1 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.

7.211.3 Constructor & Destructor Documentation

7.211.3.1 ~TextureMapper()

```
~TextureMapper ( )  [virtual]
```

Destructor.

7.211.4 Member Function Documentation

7.211.4.1 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.

7.211.4.2 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

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

See also

Drawable::draw()

Implements Drawable.

7.211.4.3 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

	invalidatedArea	The invalidated area.
in,out	*fb*	If non-null, the fb.
	triangleXs	The triangle xs.
	triangleYs	The triangle ys.
	triangleZs	The triangle zs.
	triangleUs	The triangle us.
	triangleVs	The triangle vs.

7.211.4.4 getAlpha()

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

Gets the current alpha value.

Returns

The current alpha value.

See also

setAlpha

7.211.4.5 getBitmap()

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

Gets the bitmap for the image.

Returns

the bitmap.

7.211.4.6 getBitmapPositionX()

```
float getBitmapPositionX ( ) const  [inline], [virtual]
```

Gets bitmap position x coordinate.

Returns

The bitmap position x coordinate.

7.211.4.7 getBitmapPositionY()

```
float getBitmapPositionY ( ) const  [inline], [virtual]
```

Gets bitmap position y coordinate.

Returns

The bitmap position y coordinate.

7.211.4.8 getBoundingRect()

```
Rect getBoundingRect ( ) const  [protected]
```

Gets bounding rectangle of the transformed bitmap.

Returns

The bounding rectangle.

7.211.4.9 getCameraDistance()

```
float getCameraDistance ( ) const  [inline], [virtual]
```

Gets camera distance.

Returns

The camera distance.

7.211.4.10 getCameraX()

```
float getCameraX ( ) const  [inline], [virtual]
```

Gets camera x coordinate.

Returns

The camera x coordinate.

7.211.4.11 getCameraY()

```
float getCameraY ( ) const  [inline], [virtual]
```

Gets camera y coordinate.

Returns

The camera y coordinate.

7.211.4.12 getOrigoX()

```
float getOrigoX ( ) const  [inline], [virtual]
```

Gets transformation origo x coordinate.

Returns

The transformation origo x coordinate.

7.211.4.13 getOrigoY()

```
float getOrigoY ( ) const  [inline], [virtual]
```

Gets transformation origo y coordinate.

Returns

The transformation origo y coordinate.

7.211.4.14 getOrigoZ()

```
float getOrigoZ ( ) const  [inline], [virtual]
```

Gets transformation origo z coordinate.

Returns

The transformation origo z coordinate.

7.211.4.15 getRenderingAlgorithm()

```
RenderingAlgorithm getRenderingAlgorithm ( ) const  [inline], [virtual]
```

Gets the algorithm used when rendering.

Returns

The algorithm used when rendering.

7.211.4.16 getScale()

```
float getScale ( ) const  [inline], [virtual]
```

Gets the scale.

Returns

The scale.

7.211.4.17 getSolidRect()

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

Gets solid rectangle.

Returns

largest possible solid rect.

See also

Drawable::getSolidRect()

Implements Drawable.

7.211.4.18 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TEXTUREMAPPER.

Reimplemented from Widget.

7.211.4.19 getX0()

```
float getX0 ( ) const  [inline], [virtual]
```

Get the x coordinate of the top left corner of the transformed bitmap.

Returns

The X0 coordinate.

7.211.4.20 getX1()

```
float getX1 ( ) const  [inline], [virtual]
```

Get the x coordinate of the top right corner of the transformed bitmap.

Returns

The X1 coordinate.

7.211.4.21 getX2()

```
float getX2 ( ) const  [inline], [virtual]
```

Get the x coordinate of the bottom right of the transformed bitmap.

Returns

The X2 coordinate.

7.211.4.22 getX3()

```
float getX3 ( ) const  [inline], [virtual]
```

Get the x coordinate of the bottom left corner of the transformed bitmap.

Returns

The X3 coordinate.

7.211.4.23 getXAngle()

```
float getXAngle ( ) const  [inline], [virtual]
```

Get x angle.

Returns

The x angle.

7.211.4.24 getY0()

```
float getY0 ( ) const  [inline], [virtual]
```

Get the y coordinate of the top left corner of the transformed bitmap.

Returns

The Y0 coordinate.

7.211.4.25 getY1()

```
float getY1 ( ) const  [inline], [virtual]
```

Get the y coordinate of the top right corner of the transformed bitmap.

Returns

The Y1 coordinate.

7.211.4.26 getY2()

```
float getY2 ( ) const  [inline], [virtual]
```

Get the y coordinate of the bottom right corner of the transformed bitmap.

Returns

The Y2 coordinate.

7.211.4.27 getY3()

```
float getY3 ( ) const  [inline], [virtual]
```

Get the y coordinate of the bottom left corner of the transformed bitmap.

Returns

The Y3 coordinate.

7.211.4.28 getYAngle()

```
float getYAngle ( ) const  [inline], [virtual]
```

Get y angle.

Returns

The y angle.

7.211.4.29 getZ0()

```
float getZ0 ( ) const  [inline], [virtual]
```

Get the z coordinate of the top left corner of the transformed bitmap.

Returns

The Z0 coordinate.

7.211.4.30 getZ1()

```
float getZ1 ( ) const  [inline], [virtual]
```

Get the z coordinate of the top right corner of the transformed bitmap.

Returns

The Z1 coordinate.

7.211.4.31 getZ2()

```
float getZ2 ( ) const  [inline], [virtual]
```

Get the z coordinate of the bottom right corner of the transformed bitmap.

Returns

The Z2 coordinate.

7.211.4.32 getZ3()

```
float getZ3 ( ) const  [inline], [virtual]
```

Get the z coordinate of the bottom left corner of the transformed bitmap.

Returns

The Z3 coordinate.

7.211.4.33 getZAngle()

```
float getZAngle ( ) const  [inline], [virtual]
```

Get z angle.

Returns

The z angle.

7.211.4.34 lookupRenderVariant()

```
RenderingVariant lookupRenderVariant ( ) const  [protected]
```

Returns the rendering variant based on the bitmap format, alpha value and rendering algorithm.

Returns

The RenderingVariant.

7.211.4.35 setAlpha()

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

Sets the global alpha blending value.

Parameters

a	new alpha.

7.211.4.36 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

bmp	The bitmap to be used by the widget.

7.211.4.37 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

x	The x coordinate.
y	The y coordinate.

7.211.4.38 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

x	The x coordinate.
y	The y coordinate.

7.211.4.39 setCamera()

```
void setCamera (
            float x,
            float y )  [inline], [virtual]
```

Sets the camera coordinate.

Parameters

x	The x coordinate for the camera.
y	The y coordinate for the camera.

7.211.4.40 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

d	The new camera distance.

7.211.4.41 setOrigo() [1/2]

```
void setOrigo (
            float x,
            float y,
            float z )  [inline], [virtual]
```

Sets the transformation origo.

Parameters

x	The x coordinate.
y	The y coordinate.

Parameters

z	The z coordinate.

7.211.4.42 setOrigo() [2/2]

```
void setOrigo (
            float x,
            float y )  [inline], [virtual]
```

Sets the transformation origo.

Parameters

x	The x coordinate.
y	The y coordinate.

7.211.4.43 setRenderingAlgorithm()

```
void setRenderingAlgorithm (
            RenderingAlgorithm algorithm )  [inline], [virtual]
```

Sets the algorithm to be used. Default setting is NEAREST_NEIGHBOR.

Parameters

algorithm	The algorithm to use when rendering.

7.211.4.44 setScale()

```
void setScale (
            float scale )  [virtual]
```

Sets the scale of the image.

Parameters

scale	The new scale value.

7.211.4.45 updateAngles()

```
void updateAngles (
            float xAngle,
            float yAngle,
            float zAngle )  [virtual]
```

Updates the angles of the image.

Parameters

xAngle	The new x Angle.
yAngle	The new y Angle.
zAngle	The new x Angle.

7.211.4.46 updateXAngle()

```
void updateXAngle (
            float xAngle )  [inline], [virtual]
```

Updates the x coordinate angle described by xAngle.

Parameters

xAngle	The new x angle.

7.211.4.47 updateYAngle()

```
void updateYAngle (
            float yAngle )  [inline], [virtual]
```

Updates the y coordinate angle described by yAngle.

Parameters

yAngle	The new y angle.

7.211.4.48 updateZAngle()

```
void updateZAngle (
            float zAngle )  [inline], [virtual]
```

Updates the z coordinate angle described by zAngle.

Parameters

zAngle	The new z angle.

7.212 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 pixel bits or indexe for color in CLUT entries.
- const uint8_t * extraData

  The alpha channel or clut data.
- int32_t width

  The width.
- int32_t height

  The height.
- int32_t stride

  The stride.

7.213 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

7.213.1 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

7.213.2 Constructor & Destructor Documentation

7.213.2.1 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

bmp	The bitmap to display.

7.213.3 Member Function Documentation

7.213.3.1 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

invalidatedArea	The rectangle to draw, with coordinates relative to this drawable.

Reimplemented from Image.

7.213.3.2 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	*x*	The x coordinate offset.
out	*y*	The y coordinate offset.

See also

getXOffset
getYOffset

7.213.3.3 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.

7.213.3.4 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_IMAGE.

Reimplemented from Image.

7.213.3.5 getXOffset()

```
int16_t getXOffset ( )  [virtual]
```

Returns

The x coordinate offset.

See also

getYOffset
getOffset

7.213.3.6 getYOffset()

```
int16_t getYOffset ( )  [virtual]
```

Returns

The y coordinate offset.

See also

getXOffset
getOffset

7.213.3.7 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

bmp	The bitmap instance.

See also

Bitmap

Reimplemented from Image.

7.213.3.8 setOffset()

```
void setOffset (
            int16_t x,
            int16_t y )  [virtual]
```

Sets an offset into the bitmap where the tile drawing should start.

Parameters

x	The x coordinate offset.
y	The y coordinate offset.

See also

setXOffset
setYOffset

7.213.3.9 setXOffset()

```
void setXOffset (
```

```
            int16_t x )  [virtual]
```

Sets x offset into the bitmap where the tile drawing should start.

Parameters

x	The x coordinate offset.

See also

setYOffset
setOffset

7.213.3.10 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

7.214 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.

7.214.1 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

T	Generic type parameter. Typically a AbstractButtonContainer subclass.

See also

AbstractButtonContainer

7.214.2 Member Function Documentation

7.214.2.1 setHeight()

```
void setHeight (
            int16_t height )  [inline], [virtual]
```

Parameters

height	The height.

7.214.2.2 setTileBitmaps()

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

Parameters

bmpReleased	The bitmap released.
bmpPressed	The bitmap pressed.

7.214.2.3 setTileOffset()

```
void setTileOffset (
            int16_t x,
            int16_t y )  [inline], [virtual]
```

Sets an offset into the bitmap where the tile drawing should start.

Parameters

x	The x coordinate offset.
y	The y coordinate offset.

7.214.2.4 setWidth()

```
void setWidth (
            int16_t width )  [inline], [virtual]
```

Parameters

width	The width.

7.215 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

7.215.1 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

7.215.2 Constructor & Destructor Documentation

7.215.2.1 ToggleButton()

```
ToggleButton ( )
```

Default constructor.

7.215.3 Member Function Documentation

7.215.3.1 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

activeState	If true, display the bmpPressed bitmap when not pressed. If false display the bmpReleased bitmap.

7.215.3.2 getState()

```
bool getState ( ) const  [inline]
```

Gets the state.

Returns

true if state is currently active.

7.215.3.3 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TOGGLEBUTTON.

Reimplemented from Button.

7.215.3.4 handleClickEvent()

```
void handleClickEvent (
             const ClickEvent & event )  [virtual]
```

Overrides handleClickEvent in order to swap the bitmaps after being clicked.

Parameters

event	The event to handle.

Reimplemented from AbstractButton.

7.215.3.5 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

bmpReleased	The bitmap to show in the "normal" state, ie when button is not pressed.
bmpPressed	The bitmap to show when the button is pressed.

See also

Button::setBitmaps

Reimplemented from Button.

7.216 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

7.216.1 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.

7.216.2 Member Function Documentation

7.216.2.1 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

activeState	If true, the AbstractButtonContainer will appear pressed.

7.216.2.2 getToggleCanceled()

```
bool getToggleCanceled ( )  [inline]
```

Returns

True if it succeeds, false if it fails.

7.216.2.3 handleClickEvent()

```
void handleClickEvent (
            const ClickEvent & event )  [inline], [virtual]
```

Parameters

event	The event.

Reimplemented from Drawable.

7.216.2.4 setToggleCanceled()

```
void setToggleCanceled (
            bool isToggleCanceled )  [inline]
```

Parameters

isToggleCanceled	True if is toggle canceled, false if not.

7.217 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

7.217.1 Detailed Description

Invisible widget used to capture touch events. The TouchArea consumes drag events without the widget it self moving.

See also

AbstractButton

7.217.2 Constructor & Destructor Documentation

7.217.2.1 TouchArea()

```
TouchArea ( )  [inline]
```

Default constructor.

7.217.3 Member Function Documentation

7.217.3.1 draw()

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

A TouchArea will not draw anything.

Parameters

invalidatedArea	The region of the toucharea to draw. Ignored.

Implements Drawable.

7.217.3.2 getSolidRect()

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

A TouchArea has no solid rectangle.

Returns

an empty rect.

Implements Drawable.

7.217.3.3 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_TOUCHAREA.

Reimplemented from AbstractButton.

7.217.3.4 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

event	The event to handle.

Reimplemented from AbstractButton.

7.217.3.5 handleDragEvent()

```
void handleDragEvent (
            const DragEvent & evt )  [inline], [virtual]
```

A TouchArea will not move when dragged.

Parameters

evt	The event to handle. Ignored.

Reimplemented from Drawable.

7.217.3.6 setPressedAction()

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

Associates an action to be performed when the TouchArea is pressed.

Parameters

callback	The callback is given a reference to this touch area.

7.218 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

7.218.1 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.

7.218.2 Member Function Documentation

7.218.2.1 handleClickEvent()

```
void handleClickEvent (
            const ClickEvent & event )  [inline], [virtual]
```

Parameters

event	The event.

Reimplemented from Drawable.

7.219 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.

7.219.1 Detailed Description

Class TouchCalibraiton is responsible for translating coordinates (Point) based on matrix of calibration values.

7.219.2 Member Function Documentation

7.219.2.1 setCalibrationMatrix()

```
static void setCalibrationMatrix (
            const Point * ref,
            const Point * scr )  [static]
```

Initializes the calibration matrix based on reference and measured values.

Parameters

ref	Pointer to array of three reference points.
scr	Pointer to array of three measured points.

7.219.2.2 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

in,out	*p*	The point to translate.

7.220 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.

7.220.1 Detailed Description

Basic Touch Controller interface.

7.220.2 Constructor & Destructor Documentation

7.220.2.1 ~TouchController()

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

Destructor.

7.220.3 Member Function Documentation

7.220.3.1 init()

```
void init ( )  [pure virtual]
```

Initializes touch controller.

Implemented in I2CTouchController, SDL2TouchController, SDLTouchController, and NoTouchController.

7.220.3.2 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	*x*	The x position of the touch
out	*y*	The y position of the touch

Returns

True if a touch has been detected, otherwise false.

Implemented in I2CTouchController, NoTouchController, SDL2TouchController, and SDLTouchController.

7.221 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.

7.221.1 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

7.221.2 Constructor & Destructor Documentation

7.221.2.1 Transition()

`Transition ( ) [inline]`

Constructs the Transition.

7.221.2.2 ~Transition()

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

Destructor.

7.221.3 Member Function Documentation

7.221.3.1 handleTickEvent()

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

Called for every tick when transitioning. Base does nothing.

Reimplemented in CoverTransition< templateDirection >, SlideTransition< templateDirection >, and NoTransition.

7.221.3.2 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 >.

7.221.3.3 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.

7.221.3.4 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	*cont*	The container the transition should have access to.

7.221.3.5 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 >.

7.222 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.

7.222.1 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 AbstractButtonContainer). 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

T	Generic type parameter. Typically a AbstractButtonContainer subclass.

See also

AbstractButtonContainer

Template Parameters

T	Generic type parameter.

7.222.2 Member Function Documentation

7.222.2.1 setTwoWildcardText()

```
void setTwoWildcardText (
            TypedText t )  [inline]
```

Parameters

t	A TypedText to process.

7.222.2.2 setTwoWildcardTextColors()

```
void setTwoWildcardTextColors (
            colortype newColorReleased,
            colortype newColorPressed )  [inline]
```

Parameters

newColorReleased	The new color released.
newColorPressed	The new color pressed.

7.222.2.3 setTwoWildcardTextPosition()

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

Parameters

x	The x coordinate.
y	The y coordinate.
width	The width of the text.
height	The height of the text.

7.222.2.4 setTwoWildcardTextRotation()

```
void setTwoWildcardTextRotation (
            TextRotation rotation )  [inline]
```

Parameters

rotation	The rotation.

7.222.2.5 setTwoWildcardTextX()

```
void setTwoWildcardTextX (
            int16_t x )  [inline]
```

Parameters

x	The x coordinate.

7.222.2.6 setTwoWildcardTextXY()

```
void setTwoWildcardTextXY (
             int16_t x,
             int16_t y )  [inline]
```

Parameters

x	The x coordinate.
y	The y coordinate.

7.222.2.7 setTwoWildcardTextY()

```
void setTwoWildcardTextY (
             int16_t y )  [inline]
```

Parameters

y	The y coordinate.

7.222.2.8 setWildcardTextBuffer1()

```
void setWildcardTextBuffer1 (
             const Unicode::UnicodeChar * value )  [inline]
```

Sets the first wildcard in the text. Must be a zero-terminated UnicodeChar array.

Parameters

value	A pointer to the UnicodeChar to set the wildcard to.

7.222.2.9 setWildcardTextBuffer2()

```
void setWildcardTextBuffer2 (
             const Unicode::UnicodeChar * value )  [inline]
```

Sets the second wildcard in the text. Must be a zero-terminated UnicodeChar array.

Parameters

value	A pointer to the UnicodeChar to set the wildcard to.

7.223 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.
- FontId getFontId () const

  Gets the font ID 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.

7.223.1 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

7.223.2 Constructor & Destructor Documentation

7.223.2.1 TypedText()

```
TypedText (
            const TypedTextId id = TYPED_TEXT_INVALID )  [inline], [explicit]
```

Construct a typed text.

Parameters

id	The id of the TypedText.

7.223.3 Member Function Documentation

7.223.3.1 getAlignment()

```
Alignment getAlignment ( ) const  [inline]
```

Gets the alignment associated with this TypedText.

Returns

The alignment.

7.223.3.2 getFont()

```
const Font * getFont ( ) const  [inline]
```

Gets the font associated with this TypedText.

Returns

The font.

7.223.3.3 getFontId()

```
FontId getFontId ( ) const  [inline]
```

Gets the font ID associated with this TypedText.

Returns

The font.

7.223.3.4 getId()

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

Gets the id of the typed text.

Returns

The id.

7.223.3.5 getText()

```
const Unicode::UnicodeChar * getText ( ) const  [inline]
```

Gets the text associated with this TypedText.

Returns

The text.

7.223.3.6 getTextDirection()

```
TextDirection getTextDirection ( ) const  [inline]
```

Gets the text direction associated with this TypedText.

Returns

The alignment.

7.223.3.7 hasValidId()

```
bool hasValidId ( ) const  [inline]
```

Has the TypedText been set to a proper value.

Returns

Is the id valid.

7.223.3.8 registerTexts()

```
static void registerTexts (
            const Texts * t )  [inline], [static]
```

Registers an array of texts.

Parameters

t	The array of texts.

7.223.3.9 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

data	A reference to the TypedTextData storage array.
f	The fonts associated with the array.
n	The number of typed texts in the array.

7.224 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.

7.224.1 Detailed Description

The data structure for typed texts.

7.225 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.

7.225.1 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.

7.225.2 Constructor & Destructor Documentation

7.225.2.1 ~UIEventListener()

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

Destructor.

7.225.3 Member Function Documentation

7.225.3.1 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

event	The event data.

Reimplemented in Application.

7.225.3.2 handleDragEvent()

```
void handleDragEvent (
            const DragEvent & event )  [inline], [virtual]
```

This handler is invoked when a drag event has been detected by the system.

Parameters

event	The event data.

Reimplemented in Application.

7.225.3.3 handleGestureEvent()

```
void handleGestureEvent (
            const GestureEvent & event )  [inline], [virtual]
```

This handler is invoked when a gesture event has been detected by the system.

Parameters

event	The event data.

Reimplemented in Application.

7.225.3.4 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

c	The key or button pressed.

Reimplemented in Application.

7.225.3.5 handlePendingScreenTransition()

```
void handlePendingScreenTransition ( )  [inline], [virtual]
```

This handler is invoked when a change screen event is pending.

Reimplemented in Application, and MVPApplication.

7.225.3.6 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.

7.226 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.

7.226.1 Member Typedef Documentation

7.226.1.1 UnicodeChar

```
uint16_t UnicodeChar
```

Use the UnicodeChar typename when referring to strings.

7.226.2 Member Function Documentation

7.226.2.1 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

s	DECIMAL zero-terminated unicode string to convert.

Returns

The converted integer value of the string, 0 if the string does not start with a digit.

7.226.2.2 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

	utf8	The UTF8 string.
out	*dst*	The destination buffer for the converted string.
	maxchars	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.

7.226.2.3 itoa()

```
static void itoa (
            int32_t value,
            UnicodeChar * buffer,
```

```
            uint16_t bufferSize,
            int radix )  [static]
```

Integer to ASCII conversion.

Parameters

	value	to convert.
out	*buffer*	to place result in.
	bufferSize	Size of buffer (number of 16-bit values).
	radix	to use (8 for octal, 10 for decimal, 16 for hex)

7.226.2.4 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	*dst*	Buffer for the formatted string.
	dstSize	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	*format*	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

7.226.2.5 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	*dst*	Buffer for the formatted string.
	dstSize	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	*format*	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

7.226.2.6 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	*dst*	Buffer for the formatted string.
	dstSize	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	*format*	The format string containing exactly on occurrence of f.
	value	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

7.226.2.7 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	*dst*	Buffer for the formatted string.
	dstSize	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	*format*	The format string containing exactly on occurrence of f.
	value	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 long format %f", 200);
touchgfx::Unicode::snprintfFloats(dst, dstSize, tmpfmt, values);
```

See also

snprintf, snprintfFloats

7.226.2.8 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	*dst*	Buffer for the formatted string.
	dstSize	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	*format*	The format string containing f's.
in	*values*	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

7.226.2.9 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	*dst*	Buffer for the formatted string.
	dstSize	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	*format*	The format string containing f's.
in	*values*	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 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 %f", 200);
touchgfx::Unicode::snprintfFloats(dst, dstSize, tmpfmt, values);
```

See also

snprintf, snprintfFloat

7.226.2.10 strlen() [1/2]

```
static uint16_t strlen (
            const UnicodeChar * str )  [static]
```

Gets the length of a 0-terminated unicode string.

Parameters

str	The string in question.

Returns

Length of string.

7.226.2.11 strlen() [2/2]

```
static uint16_t strlen (
            const char * str )  [static]
```

Gets the length of a 0-terminated string.

Parameters

str	The string.

Returns

Length of string.

7.226.2.12 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

str1	The first string.
str2	The second string.
maxchars	The maximum number of chars to compare.

Returns

Returns an integral value indicating the relationship between the strings: A zero value indicates that the characters compared in both strings are all equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2; And a value less than zero indicates the opposite.

7.226.2.13 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

str1	The first string.
str2	The second string.
maxchars	The maximum number of chars to compare.

Returns

Returns an integral value indicating the relationship between the strings: A zero value indicates that the characters compared in both strings are all equal. A value greater than zero indicates that the first character that does not match has a greater value in str1 than in str2; And a value less than zero indicates the opposite.

7.226.2.14 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	*dst*	The destination buffer. Must have a size of at least maxchars.
in	*src*	The source string (UnicodeChars)
	maxchars	Maximum number of characters to copy.

Returns

The number of characters copied (excluding zero-termination if encountered)

Warning

If there is no null-termination among the first n UnicodeChars of src, the string placed in destination will NOT be zero-terminated!

7.226.2.15 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	*dst*	The destination buffer. Must have a size of at least maxchars.
in	*src*	The source string as an array of chars.
	maxchars	Maximum number of characters to copy.

Returns

The number of characters copied (excluding zero-termination if encountered)

Warning

If there is no null-termination among the first n bytes of src, the string placed in destination will NOT be zero-terminated!

7.226.2.16 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

	unicode	The unicode string.
out	*utf8*	The destination buffer for the converted string.
	maxbytes	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.

7.226.2.17 utoa()

```
static void utoa (
            uint32_t value,
            UnicodeChar * buffer,
            uint16_t bufferSize,
            int radix )  [static]
```

Integer to ASCII conversion.

Parameters

	value	to convert.
out	*buffer*	to place result in.
	bufferSize	Size of buffer (number of 16-bit values).
	radix	to use (8 for octal, 10 for decimal, 16 for hex)

7.226.2.18 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	*dst*	Buffer for the formatted string.
	dstSize	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	*format*	The format string.
	pArg	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

7.226.2.19 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	*dst*	Buffer for the formatted string.
	dstSize	Size of the dst buffer measured by number of UnicodeChars the buffer can hold.
in	*format*	The format string.
	pArg	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

7.227 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.

7.227.1 Detailed Description

template<class T, uint16_t capacity>
class touchgfx::Vector< T, capacity >

A very simple container class using pre-allocated memory.

Template Parameters

T	The type of objects this container works on.
capacity	The maximum number of objects this container can store.

7.227.2 Constructor & Destructor Documentation

7.227.2.1 Vector()

```
Vector ( )  [inline]
```

Default constructor. Constructs an empty vector.

7.227.3 Member Function Documentation

7.227.3.1 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

e	The element to add to the Vector.

7.227.3.2 clear()

```
void clear ( )  [inline]
```

Clears the contents of the container. It does not destruct any of the elements in the Vector.

7.227.3.3 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

elem	The element.

Returns

true if the Vector contains the element, false otherwise.

7.227.3.4 isEmpty()

```
bool isEmpty ( ) const  [inline]
```

Query if this object is empty.

Returns

true if the Vector contains no elements.

7.227.3.5 maxCapacity()

```
uint16_t maxCapacity ( ) const  [inline]
```

Query the maximum capacity of the vector.

Returns

The capacity the Vector was initialized with.

7.227.3.6 operator[]() [1/2]

```
T & operator[] (
             uint16_t idx )  [inline]
```

Index operator.

Parameters

idx	The index of the element to obtain.

Returns

A reference to the element placed at index idx.

7.227.3.7 operator[]() [2/2]

```
const T & operator[] (
             uint16_t idx ) const  [inline]
```

Const version of the index operator.

Parameters

idx	The index of the element to obtain.

Returns

A const reference to the element placed at index idx.

7.227.3.8 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

index	The index to remove.

Returns

The value of the removed element.

7.227.3.9 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

e	The element to remove from the Vector.

7.227.3.10 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

index	The index to remove.

Returns

The value of the removed element.

7.227.3.11 reverse()

```
void reverse ( )  [inline]
```

Reverses the ordering of the elements in the Vector.

7.227.3.12 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.

7.228 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

7.228.1 Detailed Description

This class represents a homogeneous 3D vector.

See also

quadruple

7.228.2 Constructor & Destructor Documentation

7.228.2.1 Vector4() [1/2]

```
Vector4 ( )  [inline]
```

Default constructor.

7.228.2.2 Vector4() [2/2]

```
Vector4 (
            float x,
            float y,
            float z )  [inline]
```

Constructor.

Parameters

x	The x value.
y	The y value.
z	The z value.

7.228.3 Member Function Documentation

7.228.3.1 crossProduct()

```
Vector4 crossProduct (
            const Vector4 & operand )  [inline]
```

Cross product.

Parameters

operand	The second operand.

Returns

The result of the operation.

7.229 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

7.229.1 Detailed Description

template<class T>
class touchgfx::View< T >

This is a generic touchgfx::Screen specialization for normal applications. It provides a link to the Presenter class.

Note

All views in the application must be a subclass of this type.

Template Parameters

T	The type of Presenter associated with this view.

See also

Screen

7.229.2 Constructor & Destructor Documentation

7.229.2.1 View()

```
View ( )  [inline]
```

Default constructor.

7.229.3 Member Function Documentation

7.229.3.1 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	*presenter*	The specific Presenter to be associated with the View.

7.230 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

7.230.1 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

7.230.2 Constructor & Destructor Documentation

7.230.2.1 Widget()

```
Widget ( )  [inline]
```

Default constructor.

7.230.2.2 ~Widget()

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

Destructor.

7.230.3 Member Function Documentation

7.230.3.1 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

	x	The point of intersection expressed in coordinates relative to the parent.
	y	The y coordinate.
out	*last*	Result will be placed here.

Implements Drawable.

7.230.3.2 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, Scalable↩Image, BoxWithBorder, ButtonWithLabel, TiledImage, TextAreaWithOneWildcard, Box, ButtonWithIcon, Image, SnapshotWidget, Button, TouchArea, ToggleButton, and AbstractButton.

7.231 WildcardTextButtonStyle< T > Class Template Reference

A wildcard text button style.

```
#include <WildcardTextButtonStyle.hpp>
```

Public Member Functions

- WildcardTextButtonStyle ()

  Default constructor.
- virtual ~WildcardTextButtonStyle ()

  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.

7.231.1 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

T	Generic type parameter. Typically a AbstractButtonContainer subclass.

See also

AbstractButtonContainer

Template Parameters

T	Generic type parameter.

7.231.2 Member Function Documentation

7.231.2.1 setWildcardText()

```
void setWildcardText (
            TypedText t )  [inline]
```

Parameters

t	A TypedText to process.

7.231.2.2 setWildcardTextBuffer()

```
void setWildcardTextBuffer (
            const Unicode::UnicodeChar * buffer )  [inline]
```

Parameters

in,out	buffer	If non-null, the buffer.

7.231.2.3 setWildcardTextColors()

```
void setWildcardTextColors (
            colortype newColorReleased,
            colortype newColorPressed )  [inline]
```

Parameters

newColorReleased	The new color released.
newColorPressed	The new color pressed.

7.231.2.4 setWildcardTextPosition()

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

Parameters

x	The x coordinate.
y	The y coordinate.
width	The width of the text.
height	The height of the text.

7.231.2.5 setWildcardTextRotation()

```
void setWildcardTextRotation (
            TextRotation rotation )  [inline]
```

Parameters

rotation	The rotation.

7.231.2.6 setWildcardTextX()

```
void setWildcardTextX (
            int16_t x )  [inline]
```

Parameters

x	The x coordinate.

7.231.2.7 setWildcardTextXY()

```
void setWildcardTextXY (
            int16_t x,
            int16_t y )  [inline]
```

Parameters

x	The x coordinate.
y	The y coordinate.

7.231.2.8 setWildcardTextY()

```
void setWildcardTextY (
            int16_t y )  [inline]
```

Parameters

y	The y coordinate.

7.232 ZoomAnimationImage Class Reference

Class for optimizing and wrapping move and zoom operations on ScalableImages.

`#include <touchgfx/containers/ZoomAnimationImage.hpp>`

Public Types

- enum ZoomMode {
  FIXED_CENTER = 0, **FIXED_LEFT**, **FIXED_RIGHT**, **FIXED_TOP**, **FIXED_BOTTOM**, **FIXED_LEFT_AND_TOP**, **FIXED_RIGHT_AND_TOP**,
  FIXED_LEFT_AND_BOTTOM, **FIXED_RIGHT_AND_BOTTOM** }

  A ZoomMode describes in which direction the image will grow/shrink when do a zoom animation. A FIXED direction means that the image will not grow/shrink in that direction.

Public Member Functions

- ZoomAnimationImage ()

  Default constructor.

- virtual ~ZoomAnimationImage ()

  Destructor.
- void startZoomAnimation (int16_t endWidth, int16_t endHeight, uint16_t duration, ZoomMode zoomMode=FIXED_LEFT_AND_TOP, touchgfx::EasingEquation widthProgressionEquation=&touchgfx::EasingEquations::linearEaseNone, EasingEquation heightProgressionEquation=&touchgfx::EasingEquations::linearEaseNone)

  Setup and starts the zoom animation.
- void startZoomAndMoveAnimation (int16_t endX, int16_t endY, int16_t endWidth, int16_t endHeight, uint16_t duration, ZoomMode zoomMode=FIXED_LEFT_AND_TOP, EasingEquation xProgressionEquation=&touchgfx::EasingEquations::linearEaseNone, EasingEquation yProgressionEquation=&touchgfx::EasingEquations::linearEaseNone, EasingEquation widthProgressionEquation=&touchgfx::EasingEquations::linearEaseNone, EasingEquation heightProgressionEquation=&touchgfx::EasingEquations::linearEaseNone)

  Setup and starts the zoom and move animation.
- void cancelZoomAnimation ()

  Cancel zoom animation.
- virtual void handleTickEvent ()

  The tick handler.
- void setBitmaps (const Bitmap &smallBitmap, const Bitmap &largeBitmap)

  Initializes the bitmap of the image to be used.
- Bitmap getSmallBitmap () const

  Gets the small bitmap.
- Bitmap getLargeBitmap () const

  Gets the large bitmap.
- virtual void setPosition (int16_t x, int16_t y, int16_t width, int16_t height)

  Sets the size and position of the image, relative to its parent.
- virtual void setWidth (int16_t width)

  Sets the width of the image.
- virtual void setHeight (int16_t height)

  Sets the height of the image.
- virtual void setDimension (int16_t width, int16_t height)

  Sets the width and height of the image.
- virtual void setScalingMode (ScalableImage::ScalingAlgorithm mode)

  Sets the scaling algorithm of the ScalableImage.
- virtual ScalableImage::ScalingAlgorithm getScalingMode ()

  Gets the scaling algorithm of the ScalableImage.
- virtual void setAlpha (uint8_t alpha)

  Sets the alpha channel for the image.
- virtual uint8_t getAlpha () const

  Gets the current alpha value.
- virtual void setAnimationDelay (uint16_t delay)

  Sets a delay on animations done by the ZoomAnimationImage.
- virtual uint16_t getAnimationDelay () const

  Gets the current animation delay.
- void setAnimationEndedCallback (touchgfx::GenericCallback< const ZoomAnimationImage & > &callback)

  Associates an action to be performed when the animation ends.
- virtual bool isRunning () const

  Is there currently an animation running.
- virtual 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

7.232.1 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

7.232.2 Member Enumeration Documentation

7.232.2.1 States

```
enum States  [protected]
```

Enumerator

ANIMATE_ZOOM	Zoom animation state.
ANIMATE_ZOOM_AND_MOVE	Zoom and move animation state.
NO_ANIMATION	No animation state.

7.232.3 Constructor & Destructor Documentation

7.232.3.1 ZoomAnimationImage()

```
ZoomAnimationImage ( )
```

Creates and initialize the ZoomAnimationImage.

7.232.3.2 ~ZoomAnimationImage()

```
~ZoomAnimationImage ( )  [virtual]
```

Destroys the ZoomAnimationImage.

7.232.4 Member Function Documentation

7.232.4.1 getAlpha()

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

Gets the current alpha value.

Returns

The current alpha value.

7.232.4.2 getAnimationDelay()

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

Gets the current animation delay.

Returns

The current animation delay. Expressed in ticks.

7.232.4.3 getLargeBitmap()

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

Gets the large bitmap.

Returns

the large bitmap.

7.232.4.4 getScalingMode()

```
ScalableImage::ScalingAlgorithm getScalingMode ( )  [virtual]
```

Gets the scaling algorithm of the ScalableImage.

Returns

the scaling algorithm used.

7.232.4.5 getSmallBitmap()

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

Gets the small bitmap.

Returns

the small bitmap.

7.232.4.6 getType()

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

For GUI testing only. Returns type of this drawable.

Returns

TYPE_ZOOMANIMATIONIMAGE.

Reimplemented from Container.

7.232.4.7 handleTickEvent()

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

The tick handler.

Reimplemented from Drawable.

7.232.4.8 isRunning()

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

Is there currently an animation running.

Returns

true if there is an animation running.

7.232.4.9 isZoomAnimationRunning()

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

Is there currently an animation running.

Returns

true if there is an animation running.

7.232.4.10 setAlpha()

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

Sets the alpha channel for the image.

Parameters

alpha	The alpha value. 255 = completely solid.

7.232.4.11 setAnimationDelay()

```
void setAnimationDelay (
            uint16_t delay )  [virtual]
```

Sets a delay on animations done by the ZoomAnimationImage. Defaults to 0.

Parameters

delay	The delay in ticks.

7.232.4.12 setAnimationEndedCallback()

```
void setAnimationEndedCallback (
            touchgfx::GenericCallback< const ZoomAnimationImage & > & callback )  [inline]
```

Associates an action to be performed when the animation ends.

Parameters

callback	The callback to be executed. The callback will be given a reference to the ZoomAnimationImage.

See also

GenericCallback

7.232.4.13 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

smallBitmap	The image in the smallest resolution.
largeBitmap	The image in the largest resolution.

7.232.4.14 setCurrentState()

```
void setCurrentState (
            States state )  [protected], [virtual]
```

Sets the current animation state.

Parameters

state	The new state.

7.232.4.15 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

width	The new width.
height	The new height.

7.232.4.16 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

height	The new height.

Reimplemented from Drawable.

7.232.4.17 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

x	The x coordinate of this Drawable.
y	The y coordinate of this Drawable.
width	The width of this Drawable.
height	The height of this Drawable.

Reimplemented from Drawable.

7.232.4.18 setScalingMode()

```
void setScalingMode (
             ScalableImage::ScalingAlgorithm mode )  [virtual]
```

Sets the scaling algorithm of the ScalableImage.

Parameters

mode	The new mode.

7.232.4.19 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

width	The new width.

Reimplemented from Drawable.

7.232.4.20 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 startZoomAndMove↩ Animation(). If both delay and duration is zero, the end position and size is applied and the animation is ended.

Parameters

endWidth	The end width.
endHeight	The end height.
duration	The duration.
zoomMode	The zoom mode.
widthProgressionEquation	The width progression equation.
heightProgressionEquation	The height progression equation.

7.232.4.21 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::linearEase↩
None,
             EasingEquation heightProgressionEquation = &touchgfx::EasingEquations::linear↩
EaseNone )
```

Setup and starts the zoom and move animation. At end of the animation the image will have been resized to the endWidth and endHeight and have moved from its original position to the endX and endY. Please note that the ZoomMode might influence the actual end position since the zoom transformation might change the X and Y of the image. The ZoomMode FIXED_LEFT_AND_TOP ensures that the endX and endY will be the actual end position.

The development of the width, height, X and Y during the animation is described by the supplied EasingEquations. The container is registered as a TimerWidget. Unregistering is handled automatically when the animation has finished.

Parameters

endX	The X position of the image at animation end. Relative to the container or view that holds the ZoomAnimationImage.
endY	The Y position of the image at animation end. Relative to the container or view that holds the ZoomAnimationImage.
endWidth	The width of the image at animation end.
endHeight	The height of the image at animation end.
duration	The duration of the animation measured in ticks.

Parameters

zoomMode	The zoom mode that will be used during the animation. Default = FIXED_LEFT_AND_TOP.
xProgressionEquation	The equation that describes the development of the X position during the animation. Default = EasingEquations::linearEaseNone.
yProgressionEquation	The equation that describes the development of the Y position during the animation. Default = EasingEquations::linearEaseNone.
widthProgressionEquation	The equation that describes the development of the width during the animation. Default = EasingEquations::linearEaseNone.
heightProgressionEquation	The equation that describes the development of the height during the animation. Default = EasingEquations::linearEaseNone.

7.232.4.22 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::linear↩
EaseNone )
```

Setup and starts the zoom animation. At end of the animation the image will have been resized to the end↩ Width and endHeight. The development of the width and height during the animation is described by the supplied EasingEquations. The container is registered as a TimerWidget. Unregistering is handled automatically when the animation has finished.

Note that the animation follows the specified ZoomMode so the X and Y of the image might change during animation.

Parameters

endWidth	The width of the image at animation end.
endHeight	The height of the image at animation end.
duration	The duration of the animation measured in ticks.
zoomMode	The zoom mode that will be used during the animation. Default = FIXED_LEFT_AND_TOP.
widthProgressionEquation	The equation that describes the development of the width during the animation. Default = EasingEquations::linearEaseNone.
heightProgressionEquation	The equation that describes the development of the height during the animation. Default = EasingEquations::linearEaseNone.

7.232.4.23 updateRenderingMethod()

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

Chooses the optimal rendering of the image given the current width and height.

7.232.4.24 updateZoomAnimationDeltaXY()

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

Calculates the change in X and Y caused by the zoom animation given the current ZoomMode.

Index