# DEEPSCREEN TARGET USER'S GUIDE

## STM32F4 CHROM-ART SW-RENDER FOR NO O/S

*Altia® DeepScreen®*
*HMI Software*

*May 15, 2015*

## Please read these important notices before using this release.

**Altia Design 10.0 and newer Save Design Files in a New Format**

A design (.dsn) file saved in Altia Design 9.0 or newer has a new format and will not open in earlier versions of Altia Design, Altia Runtime, or Altia FacePlate. Please back-up existing design files before saving to the same files with Altia Design 9.0 or newer. Doing so allows you to open a back-up version of a design in an earlier release if desired.

**Do Not Use Altia Design 10.0 or newer to Develop Design Files that must Open on Linux or Solaris**

Design (.dsn) files saved in Altia Design 9.0 or newer will not open in a 7.06 or older release of Altia Design, Runtime, or FacePlate for Linux or Solaris. Please continue to use Altia Design 8.075 or older on Windows to develop design files that must open in a 7.06 version of Altia Design, Runtime, or FacePlate for Linux or Solaris.

**This document assumes the Altia software is installed in c:\usr\altia**

Replace occurrences of c:\usr\altia with YOUR_INSTALL_DIR if the installation directory is different.

## TABLE OF CONTENTS

## TABLE OF FIGURES

# 1.0 Target Overview

DeepScreen can generate code for the ST Micro STM32 device. The ST Micro STM32 device features a 24 bit color LCD controller with a maximum resolution of 800x600. The current port of this target supports an RGB565 display. This DeepScreen target can be ported to use multiple tool environments. This version of the DeepScreen target ships with the following target ports:

- **stm32f429ii_sk** – This target port uses the IAR compiler tools and is specific to the IAR Kickstart Kit for STM32F429II. A minimalist, working example BSP that works with the target and the STM32F429II board can be provided upon request.

## 1.1 Document Scope

The scope of this documentation is to only cover using the ST Micro STM32 driver and the specifics of the implementation. It is not meant to explain using Altia Design, DeepScreen, etc. but rather complement existing Altia documentation found in the `help/` directory under the Altia Design installation directory.

## 1.2 Requirements

### 1.2.1 Kickstart Kit for STM32F429II Target Port

This DeepScreen target port requires the following:

- IAR 7.x Embedded Workbench IDE with associated licenses
- License for the Altia ST Micro STM32 DeepScreen target
- Development hardware and associated programming and debugging tools for the ST Micro STM32 target
    - Examples: Segger J-link probe + IAR EW IDE
- Properly configured linker file
- GNU make v3.82 (or later)
    - A Win32 MinGW make executable is provided as part of the target
- Kickstart Kit for STM32F429II board with the 800x480 LCD touch screen panel
- Working example BSP for Kickstart Kit for STM32F429II board
    - Provided upon request by Altia. BSP is provided as example only and is not officially supported as part of the target.

# 2.0 Architectural Overview

The high level architecture of the Altia generated code is represented as follows:

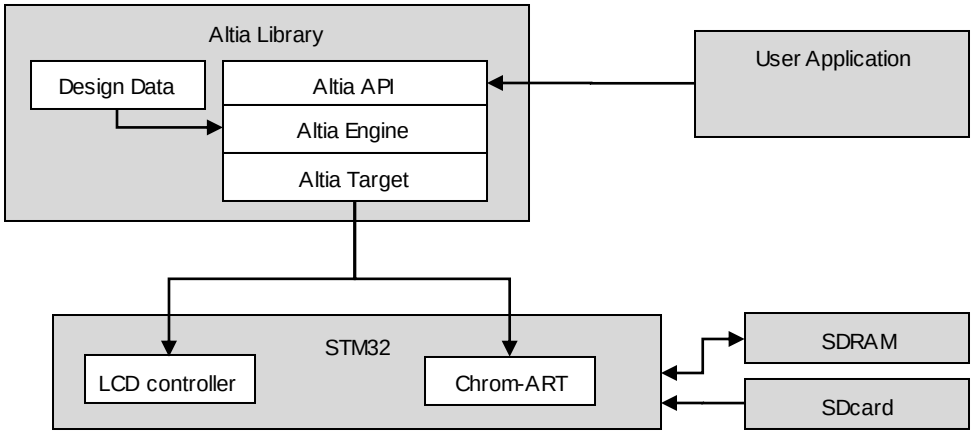


Figure 1: Design Hierarchy

The Altia Engine and API Layers handle all interactions with the User Application via the Altia C-Code API. It manages the HMI Design data and executes the behaviors for all Altia objects and User defined control. It interacts with the Altia Target Layer to render the resulting graphics on the display.

The Altia Target Layer handles all rendering operations (i.e. drawing images, etc.). It achieves the rendering using the Altia SW Render pipeline.

The Altia SW Render pipeline is used for the entire RGB565 layer. The pipeline is implemented using the standard Altia SW Render pipeline and the ST Micro STM32 hardware.

Refer to the *DeepScreen User's Guide* for more information.

## 2.1 Target Contents

The DeepScreen target is located in the Altia installation directory. This target contains source code used by the Altia DeepScreen code generator. The following folders are provided in the target:

- **ds_altiagl** – contains target software render pipeline source code
- **ds_altiagl_stmicro_stm32_noos** – contains target source code
- **ds_build_tools** – contains target build tools
- **ds_engine** – contains common source code shared between the software render pipeline and the target source code
- **ds_image** – contains target source code related to image manipulation (*not used*)
- **help** – contains the documentation for this target

## 2.2 Kickstart Kit for STM32F429II Target Port

### 2.2.1 IAR tool chain

The Kickstart Kit for STM32F429II IAR tool chain port is located in the **ds_altiagl_stmicro_stm32_noos/src/stm32f429ii_sk** folder with the ST Micro STM32 target in your Altia installation directory. This target port is configured as follows:

- GNU based makefile
- The heap and certain generated code object files that contain lots of .bss and .data sections are expected to be stored in external RAM via a section named "SDRAM_region". This region is defined in the BSP's linker configuration file.
- Operating system interfaces: system time retrieval, no semaphores

The customer is expected to create a new port when using the Altia ST Micro STM32 target for other configurations (i.e. different display hardware, operating system, input devices, or target hardware). Refer to the section on creating custom ports for this target.

# 3.0 Target Build Process

## 3.1 Overview

Refer to the *DeepScreen User's Guide* for an overview of the target code generation and build process.

For the ST Micro STM32 target port, the build process produces a set of libraries. These libraries are expected to be linked into the final user application.

## 3.2 Code Generation Considerations

### 3.2.1 Code Generation Options

To produce DeepScreen code for the ST Micro STM32 target, make sure to select the **ST Microelectronics STM32 Software Render** target from the target drop-down list.

Many code generation options are available. These are standard code generation options used across all DeepScreen targets. Refer to the *DeepScreen User's Guide* for more information on individual code generation options.

The following list of code generation options is not exhaustive, but lists some of the code generation options that are commonly used with this target.

| Code generation option | Required | Recommended | Not supported |
|---|---|---|---|
| Stimulus | | ✓ | |
| Control Code | | ✓ | |
| Timers | | ✓ | |
| Built-in Animations | | | |
| Software Scaling for Images | | ✓ | |
| Unicode Font Characters | | | |
| External Resource Files* (required for large designs that will not fit into internal STM32 flash) | ✓ | ✓ | |
| Full Screen Mode | | | ✓ |
| Use Dynamic Memory Allocation | | | |

### 3.2.2 Makefile Script

The makefile script configures the build environment and conducts a build of the Altia generated code. The build is typically started using the "Make Standalone" command from the "Generate Code" Menu in Altia Design.

Use the "Browse..." button in the dialog to view the makefile scripts available with this DeepScreen Target.

This script is typically of the form listed below.

- **altmake_<template | target_port>.bat** – This makefile script must be used with the .gen files specified in section 3.2.3.
    - Example: altmake_template.bat

### 3.2.3 Additional Target Files List

A target file (ie a file with a ".gen" suffix) selects the Driver Layer implementation for the generated code.

Use the "Browse..." button in the dialog to view the target file lists available with this DeepScreen Target.

- **stm32f429ii_sk_int.gen** – This target file generates DIB data for all the rasters in the design. The DIB rasters are typically compressed, making them small and storage space efficient. The trade-off is that they cause higher target memory consumption and require a longer load time since they must be decompressed first before copying into the framebuffer. This target file generates fixed-point only code. Refer to the *DeepScreen User's Guide* for more information on fixed-point vs floating point code.
- **stm32f429ii_sk_int_unicode.gen** – This target file is functionally the same as stm32f429ii_sk_int.gen with one important difference; it widens the default supported font character range to include all 2-byte Unicode characters. Typically, this font character range is manually fine-tuned as part of a custom target port. Refer to the *DeepScreen User's Guide* for more information on specifying font character ranges.

### 3.2.4 Code Generation Options Dialog

Below is a screenshot showing a sample code generation options dialog with many of the options already filled in.

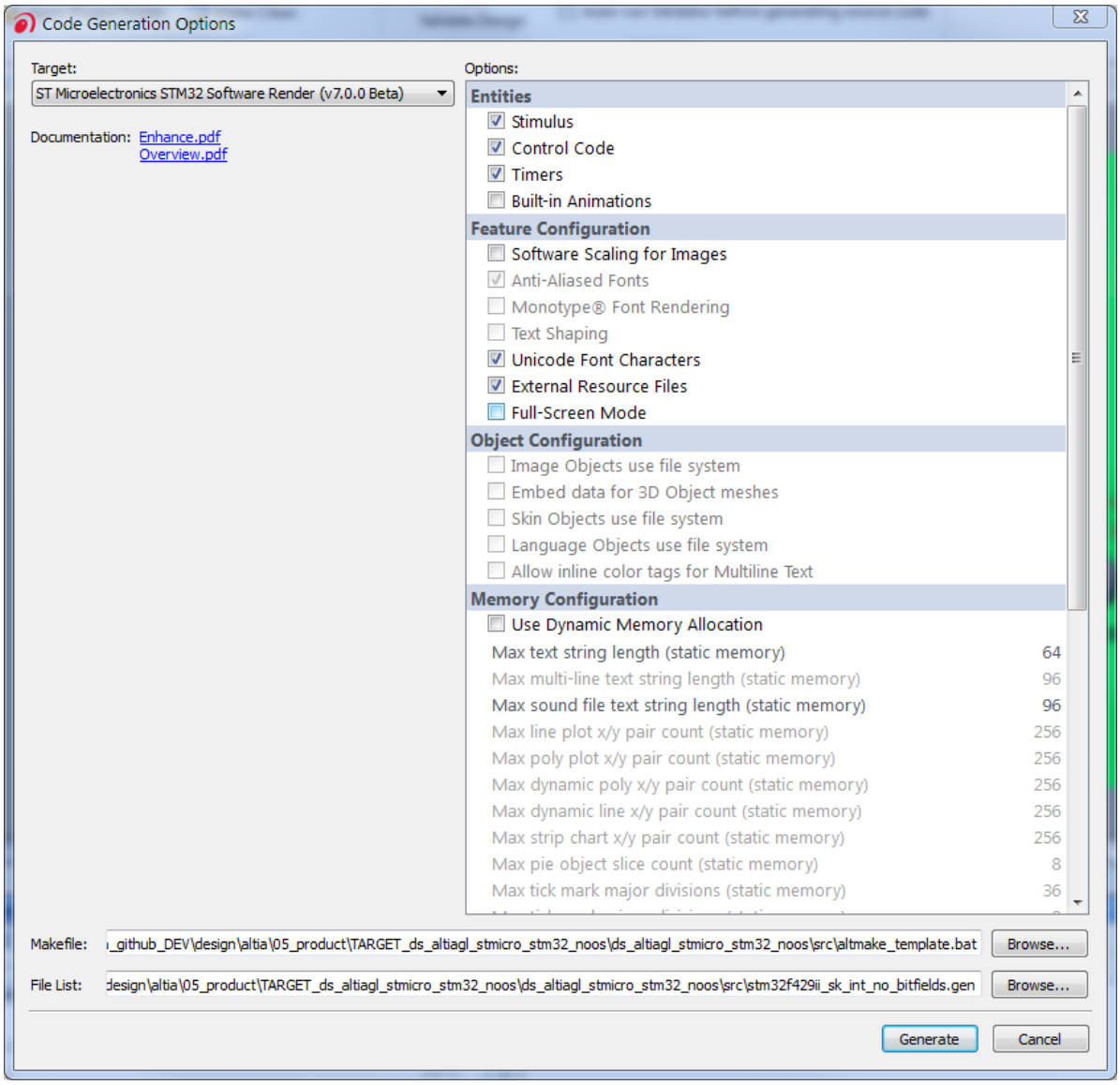


Figure 2: Typical code generation options

## 3.3 Build the Altia Code / User Application

### 3.3.1 Overview

A build script and GNU makefile are typically provided with each ST Micro STM32 target port. An example of these is provided in all the supported target ports.

The Altia libraries can be built using the "`Make Standalone`" button from the Altia "`Code Generation`" menu. A DOS shell will automatically launch, displaying the build information as it executes.

**IMPORTANT:** Your build environment must be properly configured in order to run the command line build using "`Make Standalone`".

**IMPORTANT:** Using "`Make Standalone`" requires the GNU "`make`" executable or equivalent. A Win32 compatible MinGW "`make`" executable is automatically supplied as part of each port and is copied to the `.dsn` file's root directory at code generation time.

The end result of this build step is typically a couple library files in the same root directory as the original `.dsn` file.

#### 3.3.2 Build from the Command Line

Also note that it is possible to build the Altia generated code directly from the command line. This can be done using the `altmake.bat` file that was copied to the same root directory of the `.dsn` file during the code generation step. For instructions on how to compile using the `altmake.bat` file directly from the command line, read the comments inside the batch file or use the following command to get the batch file to print out a short help message: "`altmake.bat /?`".

#### 3.3.3 Clean from the Command Line

To clean out artifacts from the application build directly from the command line, use the following command sequence:

- `.\altmake.bat clean`

#### 3.3.4 Environment Variable: TOOLCHAIN_BASE_PATH

The DeepScreen target build file utilizes an environment variable named: TOOLCHAIN_BASE_PATH.

This environment variable is expected to contain the root path to the target tool chains binary path. If the environment variable is not set, the target port uses the default target path for the tool chain it was developed with.

For example, the IAR tool chain port was developed using IAR version 7.30.4. So, the default path it uses is: "`C:\Program Files (x86)\IAR Systems\Embedded Workbench 7.0\arm`". If a different version of the IAR tool chain is desired, or a non-standard tool chain install path was used, this environment variable can be used to make sure the target's build scripts look in the right place to retrieve the tool chain's main build tool related executables.

### 3.4 External Resource Files

As part of the build process, some binary (`.bin`) files are created. Typically, there is only 1 `.bin` file generated, as listed in the table below.

| Binary File Name | Notes |
|---|---|
| `altiaImageDataBank0.bin` | Contains all image raster and font data associated with a design.<br>**IMPORTANT:** Make sure to rename file per target port requirements. Refer to section 3.4.1 for more information. |

All the external resource files, or `.bin` files, must be copied to the target HW. This process is typically BSP / HW dependent. Refer to the BSP / HW documentation for more information on this process.

#### 3.4.1 Kickstart Kit for STM32F429II Target Port

The Kickstart Kit for STM32F429II target port requires the `altiaImageDataBank0.bin` file be copied to a FAT32 formatted SDcard and renamed `altImg0.bin` in the process.

## 3.5 Target Port Project Files

#### 3.5.1 IAR Tool Chain

No IAR IDE project or workspace files are provided with the target port(s).

To create a project, some reverse engineering is required using the `altmake.mk` file (located in the port folder for this target). Make sure to note the compile-time definitions used in the `altmake.mk` and `altmake.bat` files. These can be obtained by either manually inspecting the output of the "`altmake.bat`" file or by using the following command: "`altmake.bat printBuildConfig`".

Typically, the libraries and necessary header file paths are added to the build system / IDE project settings as needed by the BSP. Refer to section 4.3.2 for more information.

# 4.0 User Application Responsibilities

It is the responsibility of the user's application to perform the tasks described in detail in the following subsections.

Further information can be found in section 6.0.

## 4.1 Target Driver Configuration

The runtime behavior of the generated code can be modified by setting specific compile time definitions. The following sections describe the different compile time definitions that are available and that can be configured. These definitions reside in the following file: `<target installation folder>\ds_altiagl_stmicro_stm32_noos\src\stm32f429ii_sk\egl_md.h`.

Compile time definitions can be adjusted by making modifications to `egl_md.h` or by adding defines to the target's compiler build options.

**IMPORTANT:** It is recommended to leave these settings at their defaults as they were installed with the target port code. If they must be changed, it is recommended contacting Altia for assistance as changing them from the defaults may have unintended side effects.

### 4.1.1 ALTIA_DOUBLE_BUFFERING

Default value: 1 (one)

If this define is set to 1 (one), graphics will be drawn to an off-screen buffer and then blitted to a framebuffer that the LCD controller reads periodically. This prevents flashing and possible tearing of the image on display since the graphics are drawn all at once. This requires more memory as an off-screen buffer that is as big as the size of the display is required. Set this define to 0 (zero) for the graphics to be drawn directly to the screen.

### 4.1.2 ALTIA_ALPHA_BLENDING

Default value: 1 (one)

Set this define to a 1 (one) to configure the Altia graphics engine to perform alpha blending where necessary. Setting this to 1 (one) will also include the required frame buffer code needed to do the blending. Note that a frame buffer is required for alpha blending (ie `USE_FB` must be enabled). If a scanline driver is being used, double buffering (`ALTIA_DOUBLE_BUFFERING`) must be enabled.

### 4.1.3 USE_FB

Default value: (defined)

If this definition is defined, the generated code will include references to the frame buffer code. The frame buffer code is used both to draw to a memory mapped display frame buffer (double buffering) and if the user wants to display text, rasters, or stencils. The Altia graphics engine will override the user's choice and defines `USE_FB` if the design file that code was generated for includes any of these types of objects. This must be defined for this frame buffer based driver.

### 4.1.4 EGL_USE_INIT_REFRESH

Default value: 0 (zero)

On some targets the initial refresh of the display is handled by a refresh event after the window has been created. This means no initial drawing of the window is needed since the refresh event will come in and that will cause the initial draw. On those targets set this define to 1 (one). On targets that don't have refresh events, Altia will do the initial draw in the initialization code. In that case, this define must be set to 0.

#### 4.1.5 ALTIA_SEMAPHORES_ENABLED

Default value: 0 (zero)

If multiple threads are used to access the target's code, this define must be set to 1 (one). This will protect the target's non-reentrant code. Set it to 0 (zero) if threads are not being used - the code will execute faster.

#### 4.1.6 ALTIA_SAVE_FONT_BITMAP

Default value: (defined)

If this definition is defined, each monochrome bitmap created when a text character is drawn is saved for when the character is drawn again. It then gets destroyed when the font is destroyed. This makes text drawing much faster but requires more memory. If this is not defined, then the monochrome bitmap is created when the text is drawn and is not saved for later use.

#### 4.1.7 EGL_USE_CURSOR_HANDLING

Default value: 0 (zero)

If this definition is set to 1 (one), the Altia graphics engine will render a mouse cursor to the display. This can be useful for a Linux target platform that utilizes a standard mouse device. If this behavior is not desired, disable this definition by changing it to 0.

#### 4.1.8 USE_MOUSE_SIMULATION_KEYS

Default value: 0 (zero)

If this definition is set to 1 (one), the Altia graphics engine will interpret the up, down, right, and left cursor keys as mouse movement. The enter key (up and down) will be treated as a left mouse button (up and down) event. This is very useful for systems without a touch screen/mouse device. In order for this to make sense EGL_USE_CURSOR_HANDLING should be set to a 1 (one).

Note that defining this will override any stimulus behavior assigned to the up, down, right, left, and enter keys within your design.

If this definition is set to 1 (one), the Altia graphics engine will render a mouse cursor to the display. This can be useful for a Linux target platform that utilizes a standard mouse device. If this behavior is not desired, disable this definition by changing it to 0.

### 4.2 Header File Includes

To use the Altia generated code with an application, simply add the following header files to the source code where Altia APIs are going to be called:

- `altia.h`

The include path for the `altia.h` header file is as follows for the integer version of the API:

- `<.dsn file path>/altia/altiaGL/lib`

## 4.3 Linker Configuration

### 4.3.1 Overview

Ultimately, memory and storage utilization depends upon the linker script configuration file used when building a final user application executable.

For the IAR tool chain, this is typically an `.icf` file.

### 4.3.2 Linking with Altia Libraries

Typically, a user application with multiple sources and/or libraries will need to integrate the target's generated and built libraries.

When the design code has been generated and built using the methodology outlined in section 3.0, some build artifacts are produced as described below. These build artifacts are typically linked into a user application's standalone executable.

| Library Name | Notes |
|---|---|
| `libaltiaAPIlib.a` | This library contains an implementation instance of the DeepScreen C API.<br>Only the fixed point (aka integer) library must be specified.<br>The library file named `libaltiaAPIlibfloat.a`, if it exists, should be ignored. |
| `libaltiaWinLib.a` | This library contains the data that describes the specific design and the executable code to render it. |

**IMPORTANT:** Order of the libraries in the tool chain's link command is very important, especially if the tool chain only makes a single pass through the object files / libraries. Be sure to specify the `libaltiaAPIlib.a` library before the `libaltiaWinLib.a` library. These libraries must come after any application code that calls Altia API functions.

### 4.3.3 Framebuffer Considerations

If using the ST Micro STM32 HW's DMA capability to transfer framebuffer(s) to an attached LCD display, care must be taken to ensure the framebuffers are aligned on an 8 byte boundary. This constraint is required by the ST Micro STM32 HW's DMA engine.

Typically, for most user applications, the framebuffer(s) will be located in external RAM since one framebuffer alone is typically larger than the entire ST Micro STM32's internal RAM.

### 4.3.4 Linker Configuration File Recommendations

A user application leveraging the Altia ST Micro STM32 target will most likely be using the MCU's internal flash as well as both internal and external RAM. In addition, depending on the HW, external flash may be used as well.

Altia recommends the following considerations be taken into account when constructing a user application executable:

| File | Section | Notes |
|---|---|---|
| data.o | .bss / zi | Depending on the design, the Altia generated data.c file can end up having some very large arrays that are assigned to either .bss and .data linker file sections.<br>The IAR documentation refers to .bss sections as "zi" or "zero initialized" data, and .data sections as "rw" or "read write" data.<br>Altia recommends sticking these specific object files' sections into external RAM to conserve internal RAM. |
| | .data / rw | |
| egl_font.o | .bss / zi | Depending on the design, the Altia generated egl_font.c file may also end up having some very large arrays that are assigned to either .bss and .data linker file sections.<br>The IAR documentation refers to .bss sections as "zi" or "zero initialized" data, and .data sections as "rw" or "read write" data.<br>Altia recommends sticking these specific object files' sections into external RAM to conserve internal RAM. |
| | .data / rw | |
| * | .rodata | Depending on the design, the Altia generated code may have large amounts of data in the linker file ".rodata" sections (ie read-only or const data).<br>These sections typically live in internal flash. If a user application needs more internal flash space, Altia recommends sticking all its .rodata sections into external storage / flash. |

## 4.4 External Resource Files

Some binary resource (.bin) files are created as part of the build process. These are described in section 3.4.

Usage of these .bin files is BSP / HW dependent. Refer to the BSP / HW documentation for more information on this process.

## 4.5 BSP Dependencies

The BSP must satisfy the dependencies listed in the table below to properly support the DeepScreen target. This list covers the major highlights and is not exhaustive.

| Dependency | Notes |
|---|---|
| STM32F4 HAL | The STM32F4 family comes with a HAL (Hardware Abstraction Layer) SW library that is used to configure and use various on-board STM32 HW peripherals.<br>This SW module is used by the DeepScreen target to manage the Chrom-ART (aka DMA2D) and LTDC STM32 HW peripherals. |
| STM32F4 HAL DMA2D ISR handler | The STM32F4 HAL's DMA2D (aka Chrom-ART) ISR handler must be plugged into the BSP's interrupt vector table to be able to properly leverage the DMA2D callbacks that get called when a DMA2D transaction completes. |
| STM32F4 HAL LTDC ISR handler | The STM32F4 HAL's LTDC ISR handler must be plugged into the BSP's interrupt vector table to be able to properly leverage the LTDC callbacks that get called when a VSYNC interrupt occurs. |
| LCD configuration | The BSP must configure the LTDC STM32 HW peripheral to support the proper configured required by the DeepScreen target port (RGB565, RGB888 (32-bit), etc). |

# 5.0 Unicode Support

## 5.1 Overview

The ST Micro STM32 target supports Unicode. It is strongly recommended that the `.gen` file provided with the target be copied and customized per the user's project requirements to avoid large amounts of potentially unnecessary character image data.

Please refer to the following documentation for further information about Unicode character support and customization:

- *Unicode User's Guide* in the Altia Design editor `Help` menu
- Comments in this target's `.gen` files

# 6.0 HOWTO: Create a Customer Specific Port for the ST Micro STM32 Target

## 6.1 Overview

It is possible to create a customer specific port of the ST Micro STM32 target. In general, it involves making a copy of the existing generic target and modifying it as necessary.

This process is described in detail in the *DeepScreen User's Guide*. Refer to the sections related to target customization and integration as well as custom application considerations.

## 6.2 DeepScreen Considerations

The Altia graphics engine and Altia Design editor are very powerful in terms of creational scope and functionality.

To learn more about the features, capabilities, and constraints of the Altia graphics engine, please refer to the *DeepScreen User's Guide*.

To learn more about the Altia API, including its capabilities and features, please refer to the *Altia API Reference Manual*.

To learn more about the Altia Design editor, please refer to the *Altia Design User's Guide*.

# 7.0 Troubleshooting

This section describes some techniques for determining why an HMI is not functioning correctly on the target hardware.

When an error is encountered in the Altia ST Micro STM32 target code, the `DS_DBG_PRINT()` macro is typically called. Assuming this macro was ported successfully, error information will be reported to an attached serial console.

## 7.1 Serial Terminal / Console

The Kickstart Kit for STM32F429II board supports the use of a serial terminal. The required serial terminal configuration settings are: 19200 baud rate and 8-N-1 for flow control.

Altia strongly recommends connecting a serial terminal, as very useful debug information, as well as some general status information, is often printed to the serial console.

## 7.2 altiaDebug.h

The target port source file, `altiaDebug,h`, contains several preprocessor definitions related to debugging. These preprocessor definitions turn on various debug checks and debug prints in the Altia target driver layer code. It is recommended that a serial console be attached to the target's serial port for these debug preprocessor definitions to be of any use since the code enabled by them makes heavy use of the `DS_DBG_PRINT()` macro.

Enabling debug options will slow down execution speed of the Altia graphics engine code on the target HW. They are intended for debug use only and should not be enabled for a production release version of a software application.

## 7.3 driver_error.c

The target port source file, `driver_error.c`, contains several functions related to debugging. These functions are called by core Altia graphics engine code. code. It is recommended that a serial console be attached to the target's serial port for these debug preprocessor definitions to be of any use since the code enabled by them makes heavy use of the `DS_DBG_PRINT()` macro.

### 7.3.1 Driver Error Codes

The `driver_error()` function will receive one of the enumerated values detailed in the ensuing sections if an error is thrown.

#### 7.3.1.1 DRIVER_ERROR_STM32_HAL_DMA2D_INIT_FAILURE

Error thrown when there is a Chrom-ART (aka DMA2D) initialization failure.

#### 7.3.1.2 DRIVER_ERROR_STM32_HAL_DMA2D_CONFIG_LAYER_FAILURE

Error thrown when there is a Chrom-ART (aka DMA2D) layer configuration / initialization failure.

#### 7.3.1.3 DRIVER_ERROR_STM32_HAL_DMA2D_START_FAILURE

Error thrown when there is a problem kicking off a Chrom-ART (aka DMA2D) transaction.

#### 7.3.1.4 DRIVER_ERROR_STM32_HAL_DMA2D_TRANSFER_ERROR

Error thrown when there is a problem during a Chrom-ART (aka DMA2D) transaction.

#### 7.3.1.5 DRIVER_ERROR_STM32_HAL_DMA2D_TRANSFER_TIMEOUT

Error thrown when there is a timeout for during a Chrom-ART (aka DMA2D) transaction.

The default timeout is 10 milliseconds for Chrom-ART memory transfer transactions used by the target driver. This value can be changed by defining and overriding the default C preprocessor macro, ALTIA_DRV_DMA2D_TIMEOUT_MILLISECONDS, at build time for the Altia build artifacts.

#### 7.3.1.6 DRIVER_ERROR_STM32_LTDC_PROGRAM_LINE_EVENT

Error thrown when there is an issue configuring and enabling the LTDC VSYNC interrupt.

# Appendix

## A. Acronyms

The following acronyms are used in this document:

| Acronym | Expanded Information |
|---|---|
| BSP | Board Support Package |
| CRC | Cyclic Redundancy Check |
| DDB | Device Dependent Bitmap |
| DIB | Device Independent Bitmap |
| DMA2D | A synonym for the STM32 HW peripheral called "Chrom-ART", a HW accelerated graphics blitter engine |
| HMI | Human Machine Interface |
| HW | Hardware |
| IDE | Integrated Development Environment |
| JTAG | Joint Test Action Group<br>Term typically used as a reference to a type of testing and in-system programming interface commonly found on microprocessors |
| LCD | Liquid Crystal Display |
| LTDC | LCD TFT Display Controller, an STM32 HW peripheral |
| MCU | Microcontroller |
| MinGW | Minimalist GNU for Windows |
| OS | Operating System |
| RAM | Random Access Memory |
| RTOS | Real Time Operating System |
| TFT | Thin Film Transistor display |
| VS | Visual Studio |
| Win32 | Microsoft Windows 32-bit |