3D35 3.5-inch Display Programming Guide

Rev. 2

Revision	Date	Description
1	March 4, 2024	Pre-Release
2	April 9, 2024	Updated screenshots with the new Grayhill logo

561 West Hillgrove Avenue • La Grange, Illinois 60525 USA (+1) 708-354-1040 grayhill.com

1. INTRODUCTION

1.1 OVERVIEW

The purpose of this document is to provide instructions on how to develop a software application for the Grayhill 3D35 display.

The process covered involves:

- obtaining a 3D35 demo project
- importing a 3D35 project in the STM32CubeIDE development tool
- building a 3D35 application using the TouchGFX and the STM32CubeIDE development tools
- running and debugging a 3D35 application in the STM32CubeIDE development tool

Also covered is:

- loading the 3D35 application binary to flash using a UDS tool
- the 3D35 software API used to interface to the 3D35 hardware

2. HARDWARE

The following hardware is required for programming the 3D35 display.

2.1 3D35DEV-100 DEVELOPMENT KIT

The 3D35DEV-100 kit contains a 3D35XK-101D development unit and a 3DXX1314-1 development cable. The kit/cable is not absolutely required but helpful. This kit is available for purchase from Grayhill.

2.2 3D35XK-101D DEVELOPMENT UNIT

The 3D35XK-101D development unit is the same as the 3D35XK-100 production unit except the backplate has an additional debug connector exposed. This unit is not sealed and for this reason will not meet all of the specifications of the 3D35XK-100 production unit. This unit can be purchased as part of a kit or standalone from Grayhill.

The 3D35XK-100 unit is the production version of the 3D 3.5 inch display. It does not contain a software debug interface and is not intended for application development. However software can be loaded to its flash via the CAN1 port.

CONNECTOR INFORMATION

PIN	SIGNAL	PIN	SIGNAL
1	VIN POSITIVE	7	INPUT 3
2	VIN RETURN	8	OUTPUT 1
3	CAN2 HI	9	OUTPUT 2
4	CAN2 LO	10	ANALOG GND
5	INPUT 1	11	CAN1 HI
6	INPUT 2	12	CAN1 LO

3D35XK-101D and 3D35XK-100 pinout

2.3 3DXX1314-1 DEVELOPMENT CABLE

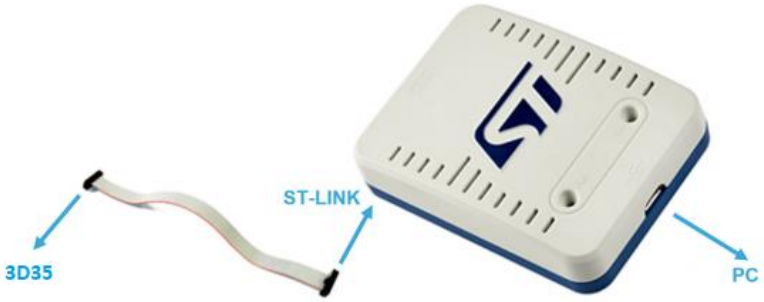
The 3DXX1314-1 development cable provides power to the unit via a DT06-12SA connector and a DB9 cable with termination that can be used to connect the 3D35 CAN1 port to a CAN network. This cable can be purchased as part of a kit or standalone from Grayhill.

2.4 ST-LINK/V3 DEBUGGER/PROGRAMMER

The ST-LINK/V3 from STMicroelectronics is an in-circuit debugger and programmer for the STM32 microcontroller used in the 3D35 display. For detailed information about the product, visit the STMicroelectronics web page:

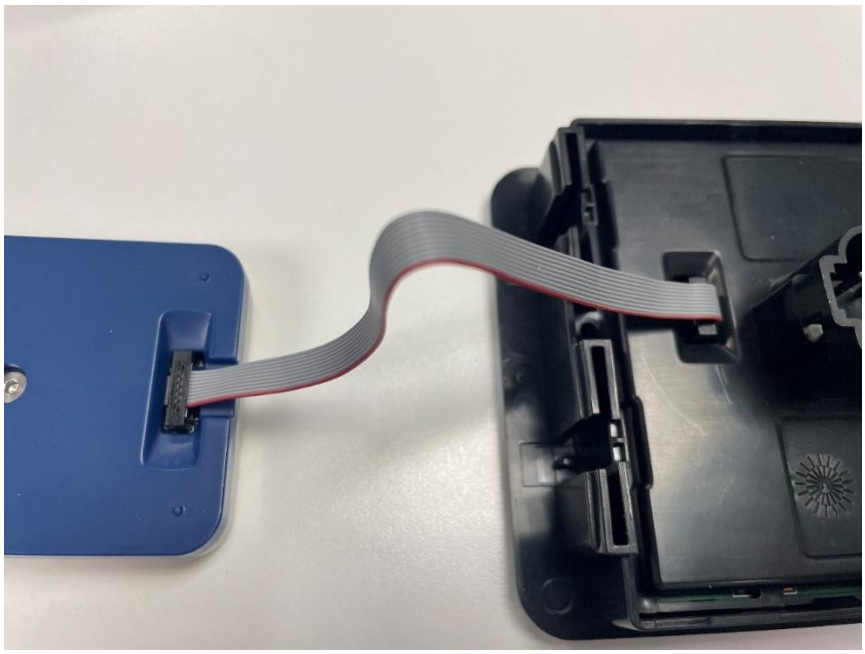
https://www.st.com/en/development-tools/stlink-v3set.html

The ST-LINK/V3 is readily available through online distributors such as DigiKey, Mouser, etc. Please contact your favorite distributor to purchase.



The ST-LINK/V3 comes with several cables. Used here is the 14-pin to 10-pin ribbon cable. The 14-pin connector goes to the ST-LINK/V3 JTAG adapter and the 10-pin connecter goes to the Grayhill 3D35 display.

The picture below illustrates the correct way to connect the cable from the ST-LINK/V3 to the 3D35 display. In particular, please note the orientation of the cable connection to the 3D35 display.



3. SOFTWARE TOOLS

This document is not an in-depth reference for TouchGFX Designer or for the STM32Cube PC Tools. Those documents can be found online, as well as instructional videos and other resources from STMicroelectronics.

https://support.touchgfx.com/docs/introduction/welcome

https://www.st.com/en/development-tools/stm32cubeide.html

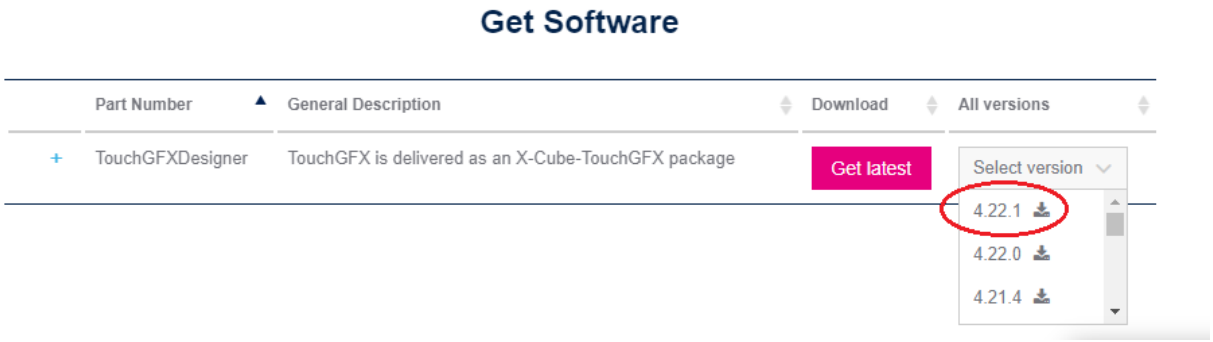
3.1 TOUCHGFX DESIGNER

TouchGFX Designer is a graphical development tool from ST Microelectronics for creating the visual appearance of your TouchGFX application. It allows you to develop screens using widgets, images, text, etc.

Download and install TouchGFX Designer online from ST Microelectronics:

https://www.st.com/en/development-tools/touchgfxdesigner.html

NOTE
The standard application that comes pre-loaded on the 3D35 was developed using TouchGFX 4.22.1.



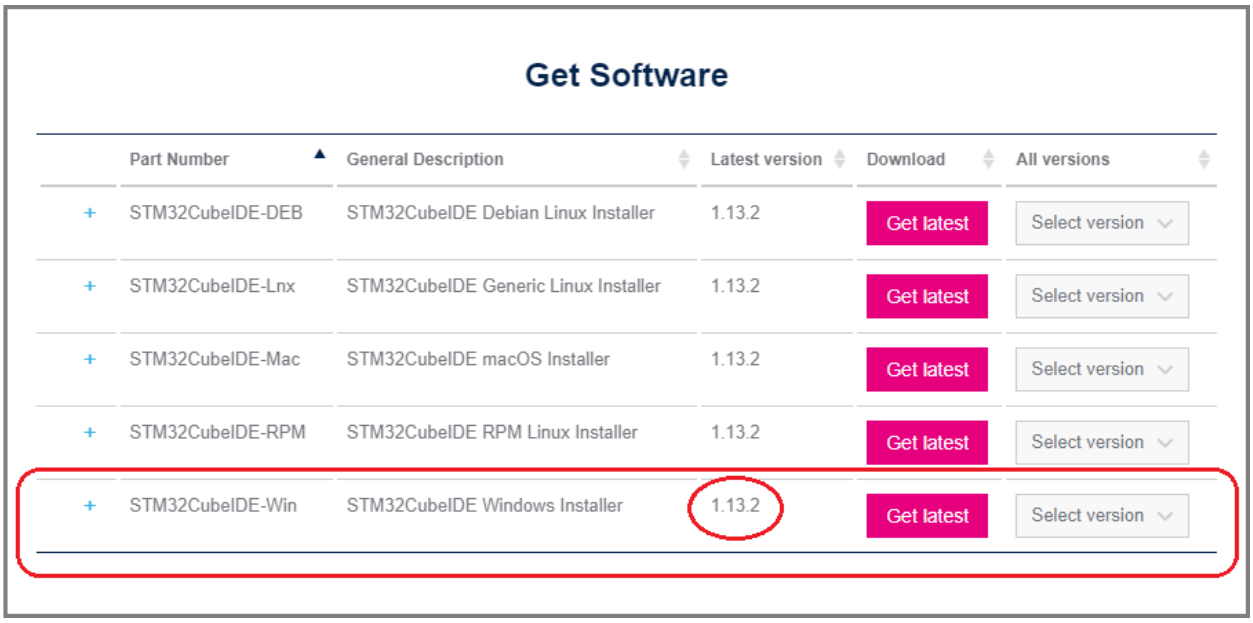
3.2 STM32CUBE IDE (INTEGRATED DEVELOPMENT ENVIRONMENT)

Download and install the STM32Cube IDE from ST Microelectronics:

https://www.st.com/en/development-tools/stm32cubeide.html

The STM32Cube IDE is a C/C++ development tool for use with STM32 microcontroller. The tool is based on the Eclipse framework, the GCC toolchain for building applications, and GDB for debugging.

Note: the standard application that comes pre-loaded on the 3D35 was developed using STM32CubeIDE 1.13.2.

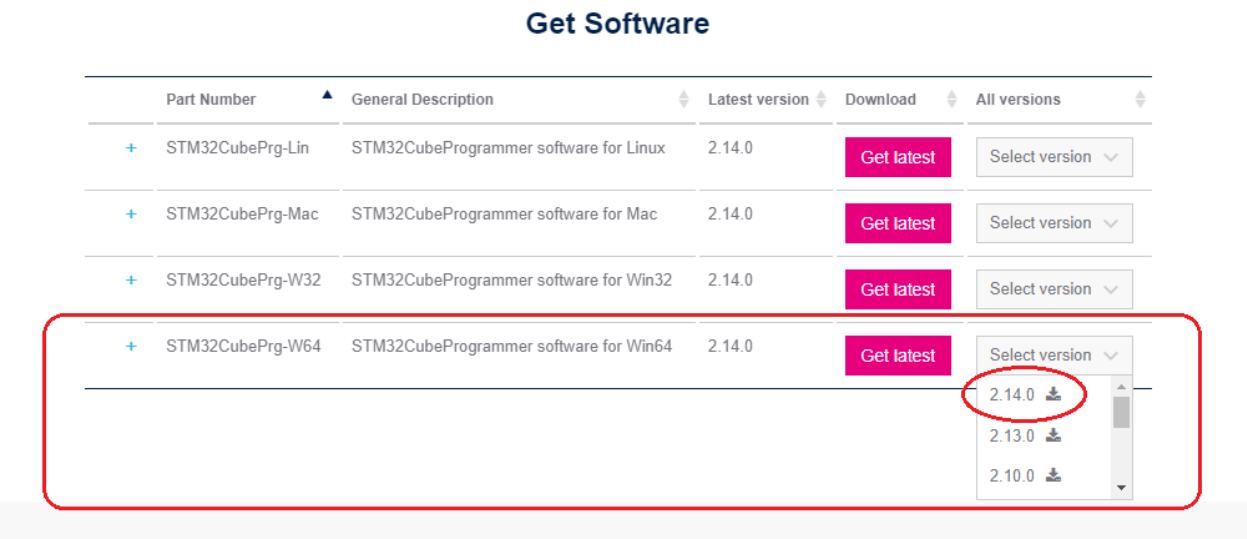


3.3 STM32CUBE PROGRAMMER

Download and install the STM32Cube Programmer software from ST Microelectronics:

https://www.st.com/en/development-tools/stm32cubeprog.html

The STM32CubeProgrammer will download an application developed with the STM32Cube IDE to the 3D35 display.



4. OBTAIN GRAYHILL DEMO PROJECT

The Grayhill 3D35_Demo application is provided as a zip file and is available for download from Grayhill web page:

https://grayhill.com/offhwyfiles/

If you are unable to locate the demo zip file, please contact your Grayhill representative to obtain a link.

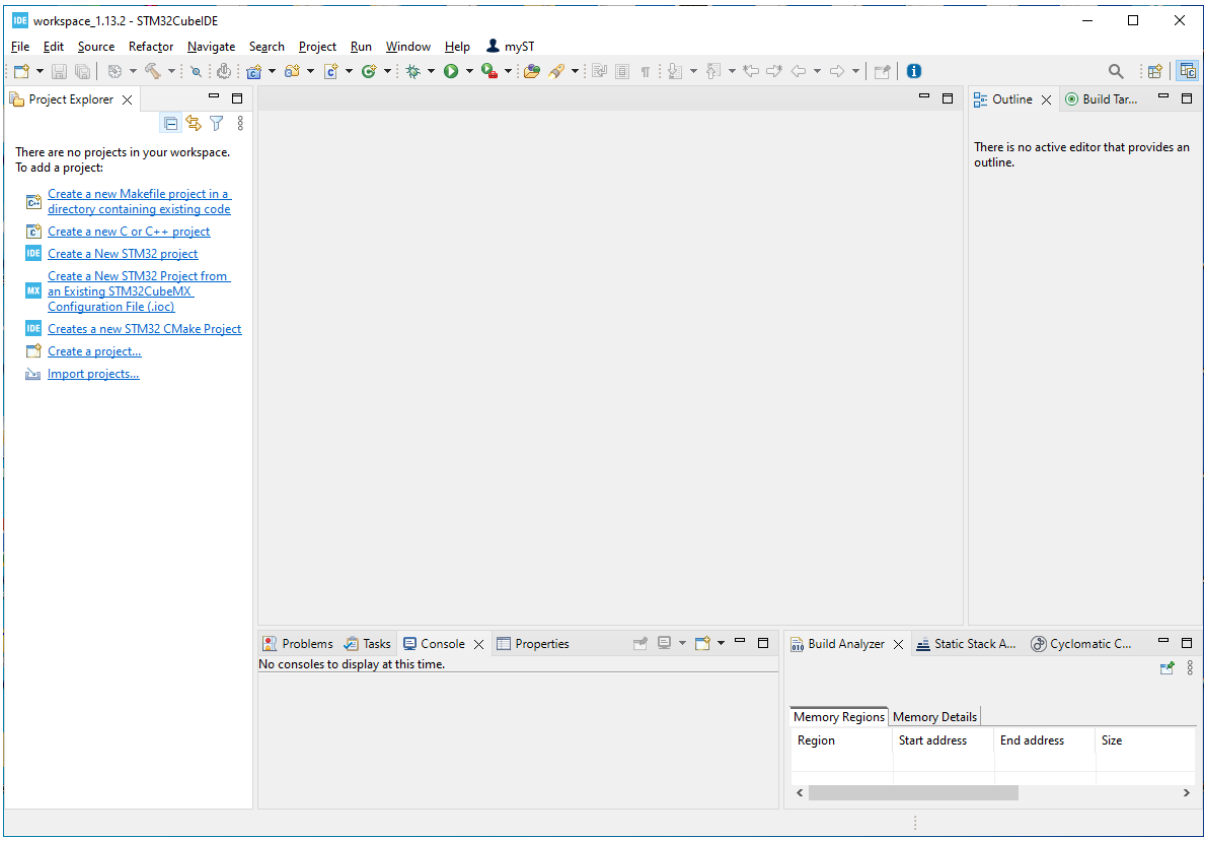
5. IMPORT GRAYHILL DEMO PROJECT

This section describes how to import the Grayhill 3D35_Demo project into the STM32CubeIDE.

After obtaining the project zip file as described in the previous section, follow these steps to import the project from the zip file into the STM32CubeIDE workspace:

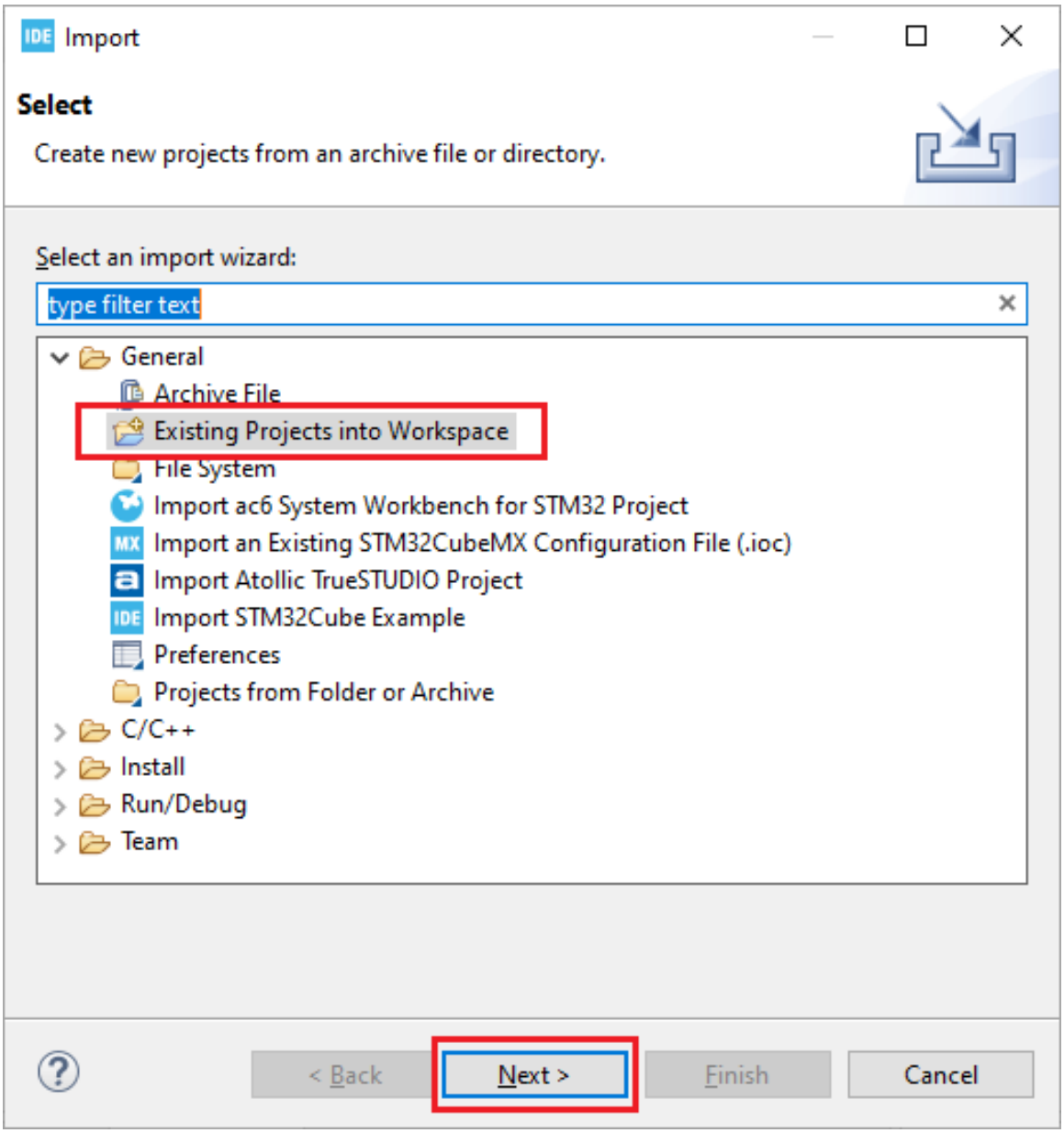
Extract the zip file into the STM32CubeIDE workspace:
<…>\STM32CubeIDE\workspace_1.13.2\demo_3D35

Open STM32CubeIDE 1.13.2.



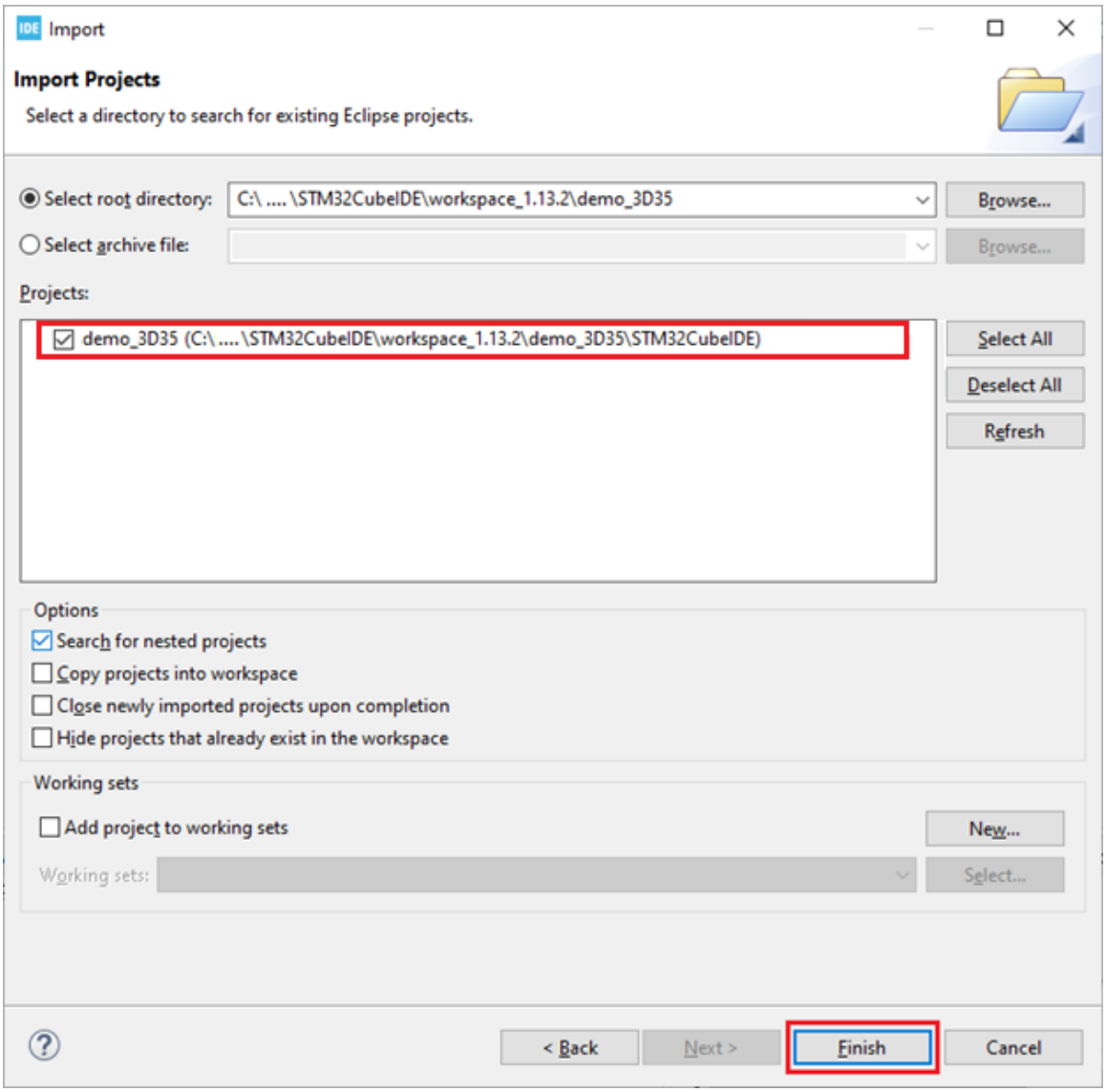
In the Project Explorer window, right click and select **Import…**

Under General, select **Existing Projects into Workspace** and click **Next**.

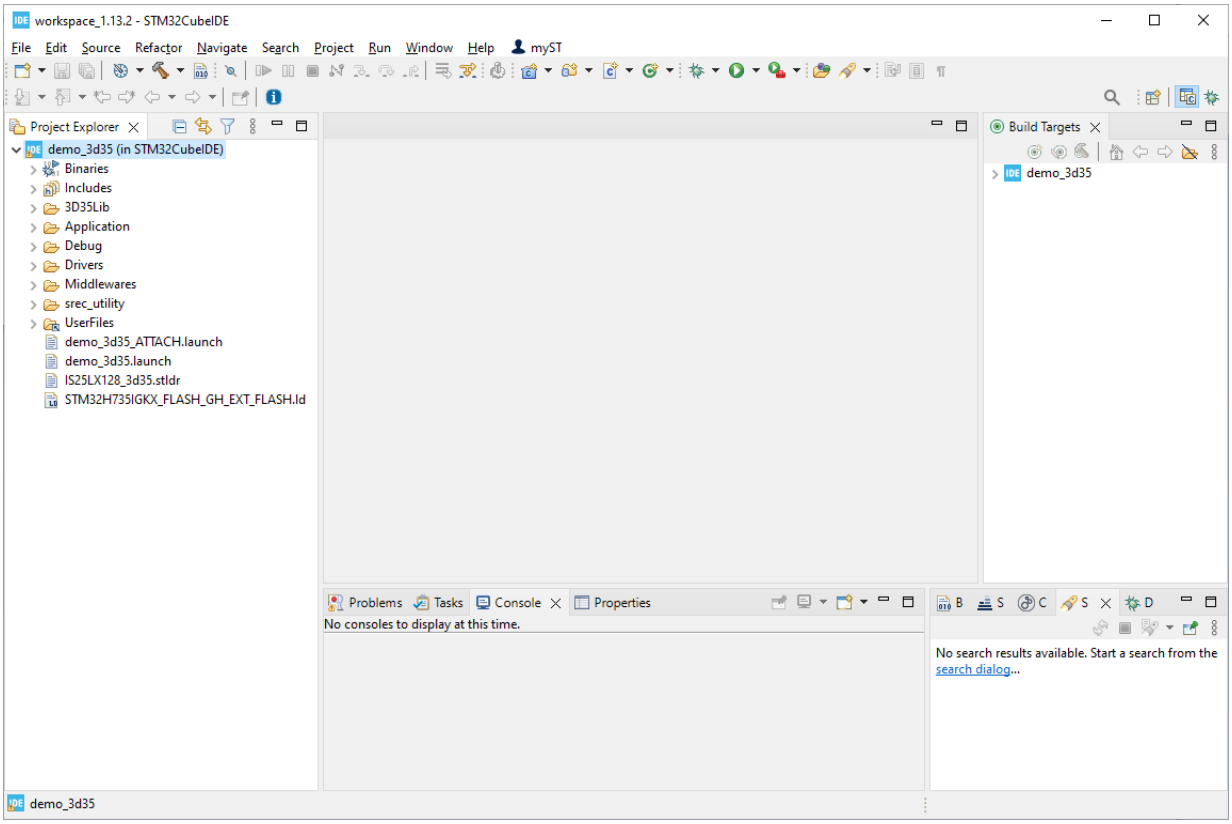


In the Import dialog, for “Select root directory” click **Browse** and navigate to the demo_3D35 folder in the workspace.

Select project **demo_3D35** and click **Finish**.



The project will be imported into the Workspace.



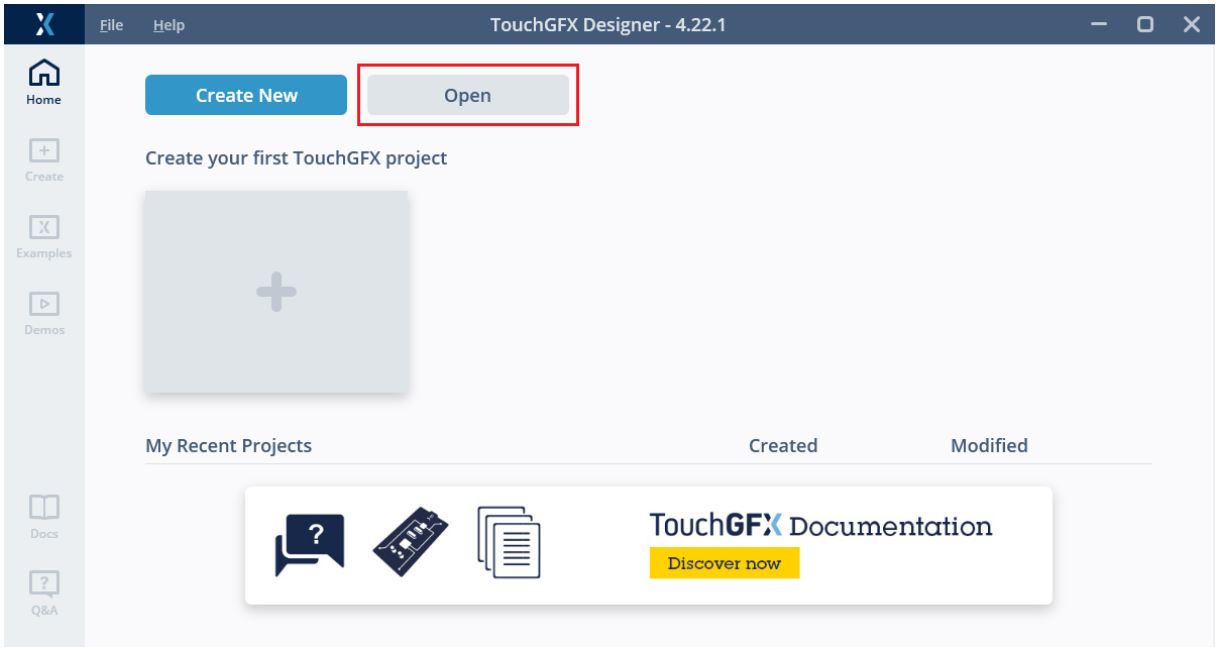
6. BUILD DEMO APPLICATION

This section describes how to build the Grayhill 3D35_Demo application using the ToughGFX and STM32CubeIDE tools.

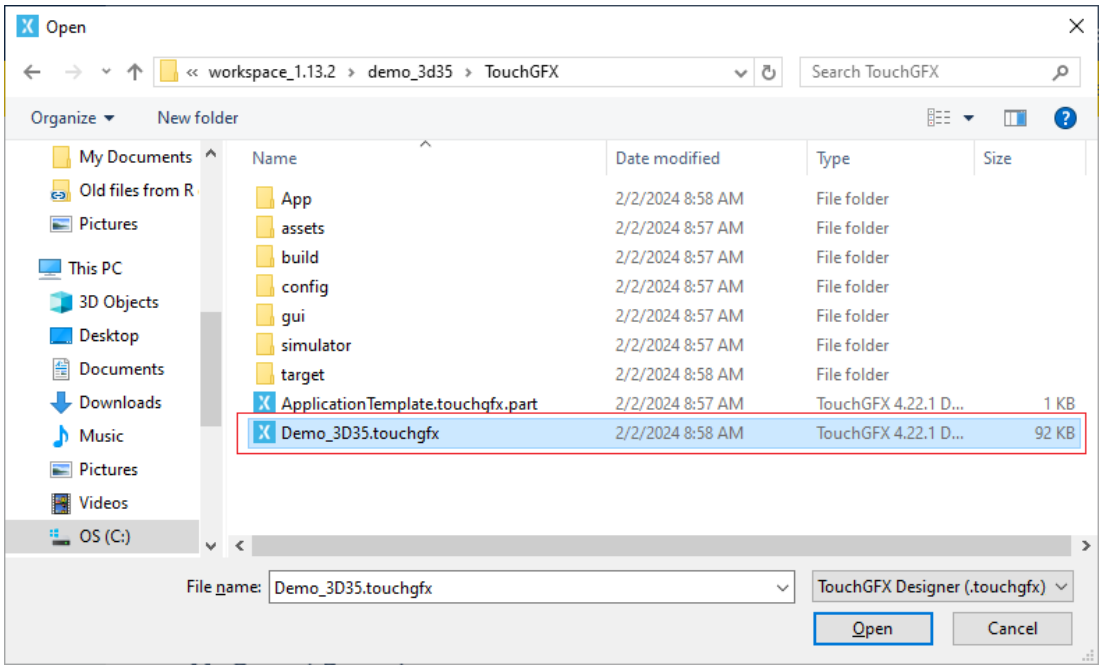
Once the demo project has been imported into the IDE workspace, follow these steps to build it:

6.1 GENERATE THE TOUCHGFX CODE

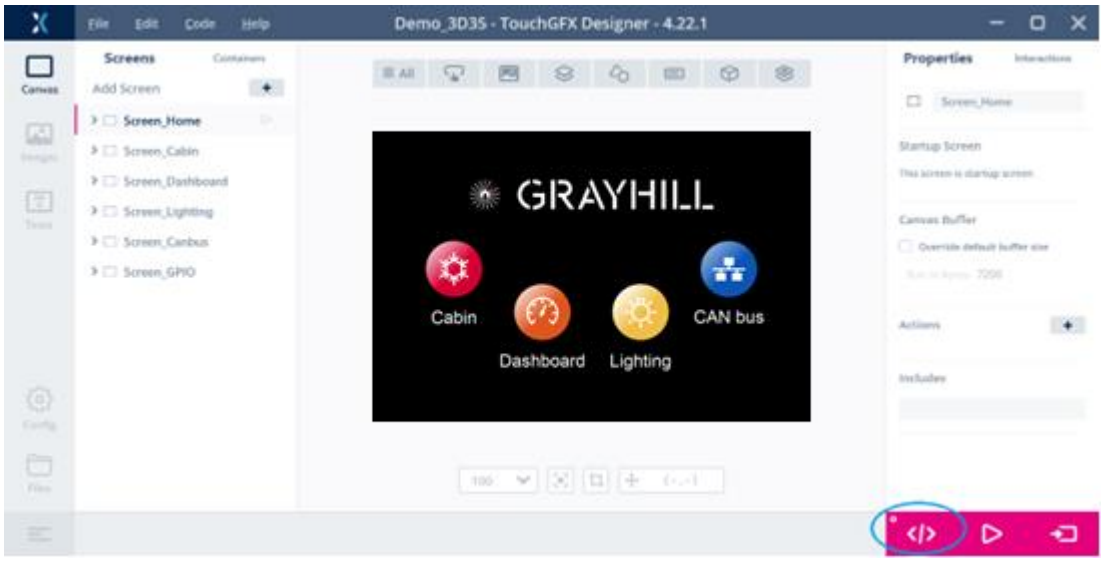
1. Open TouchGFX Designer 4.22.1.
2. Click **Open**.



3. In the Open dialog, navigate to the TouchGFX folder in the Demo_3D35 project.
4. Select **Demo_3D35.touchgfx** and click **Open**.



5. Click the **Generate Code** icon in the lower right corner.

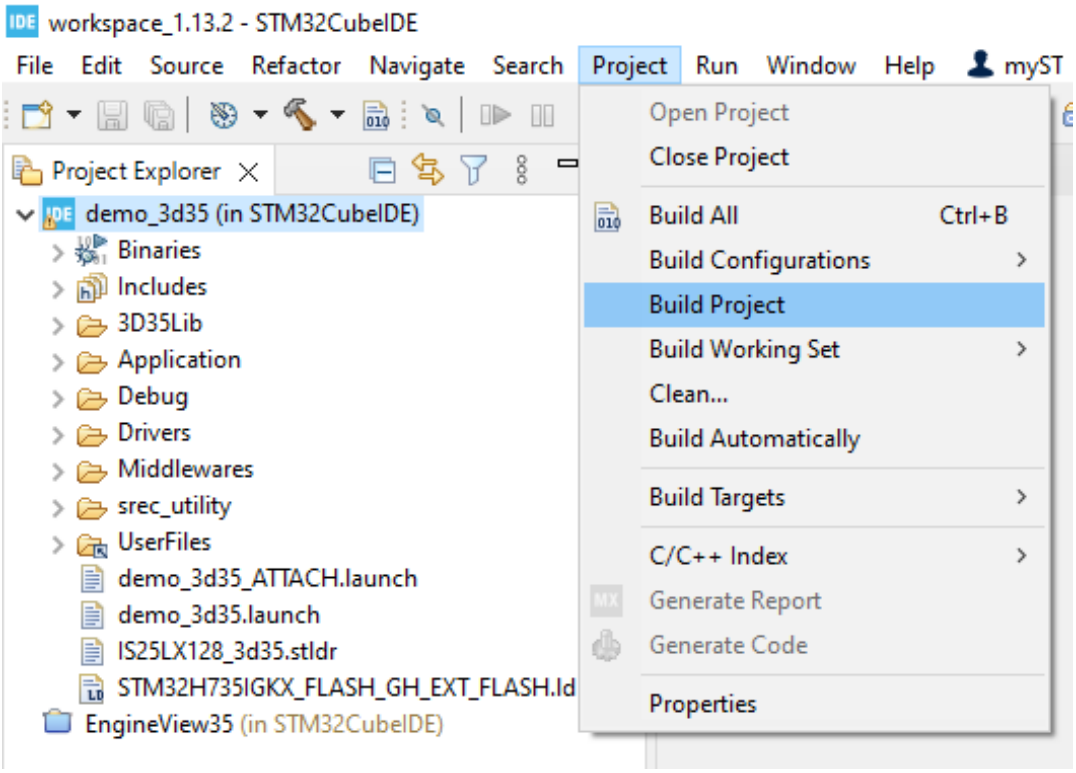


The TouchGFX code should generate, and when it finished successfully should display the following message in the bottom left corner of the TouchGFX Designer window:

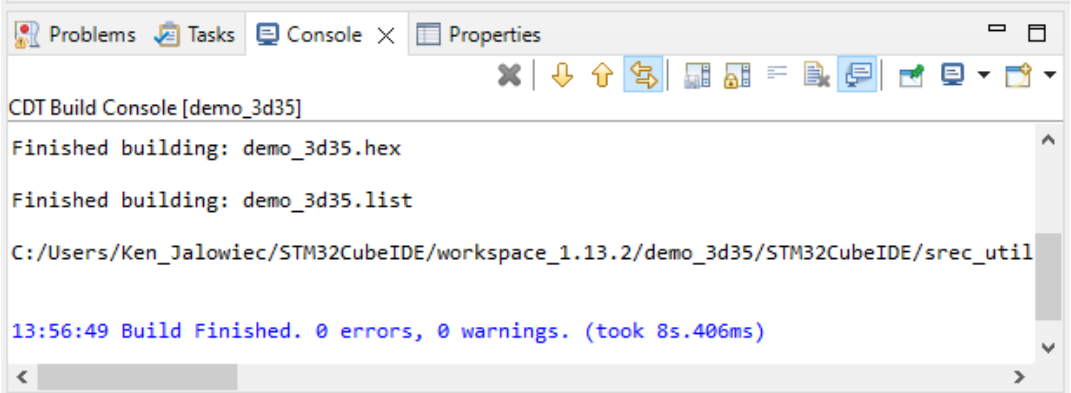
6.2 BUILD THE DEMO PROJECT IN STM32CUBEIDE

After the TouchGFX code has been generated, follow these steps to build the project in the STM32CubeIDE:

1. Returning to the STM32Cube IDE, execute 'Build Project' either by clicking **Project** from the main menu or by right-clicking in the Project Explorer window.



2. The build results are shown in the CDT Build Console window:



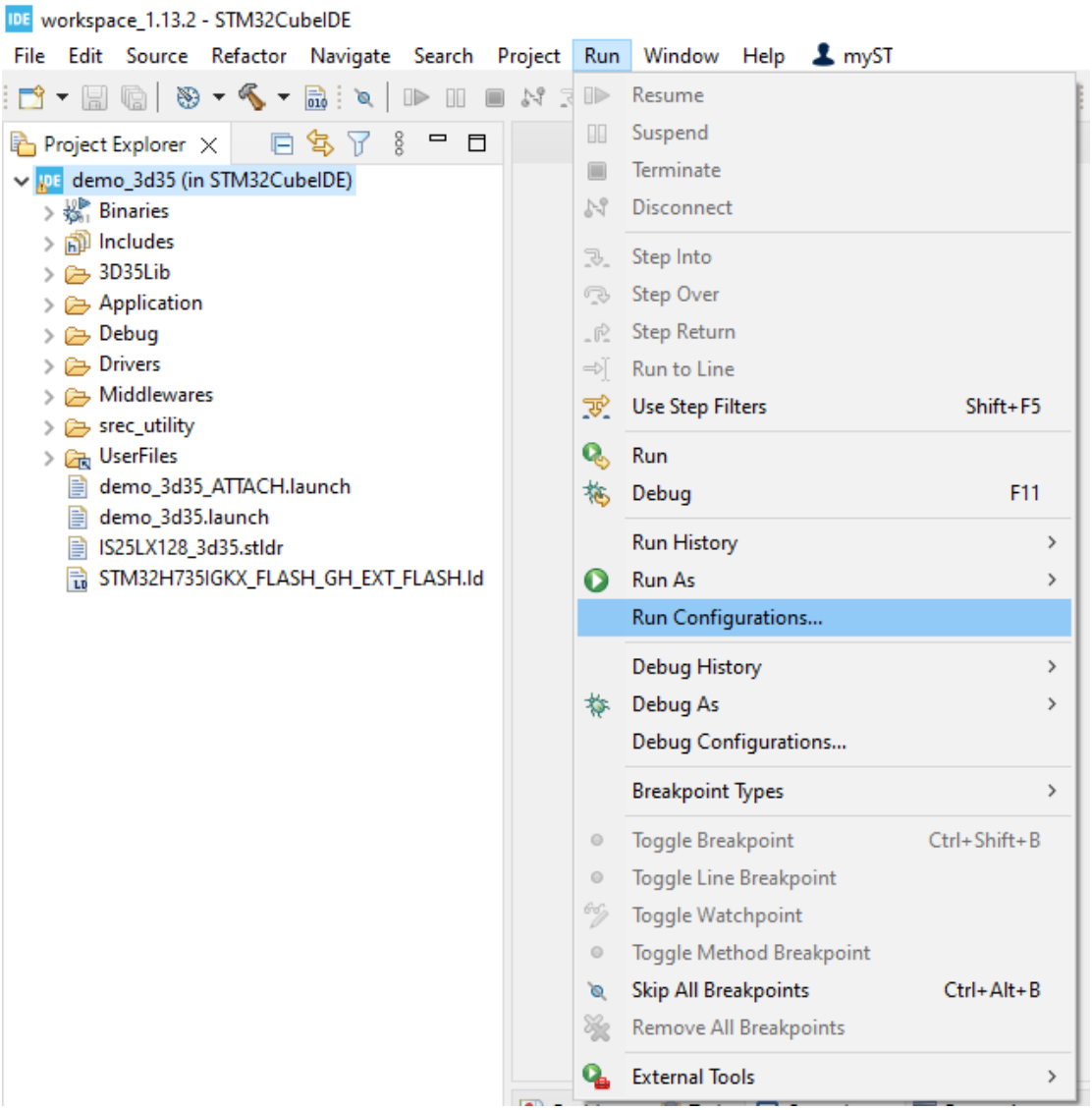
7. RUN/DEBUG DEMO APPLICATION

The successfully built demo application can be run/debugged directly via the STM32Cube IDE.

7.1 RUN THE 3D35_DEMO APPLICATION

The Run configuration needs to be configured to work with your STLINK/V3 JTAG adapter. Follow these steps to complete the Run Configuration and run/debug the demo on the 3D35 display.

1. From the STM32CubeIDE Main menu, click **Run** and select **Run Configurations…**

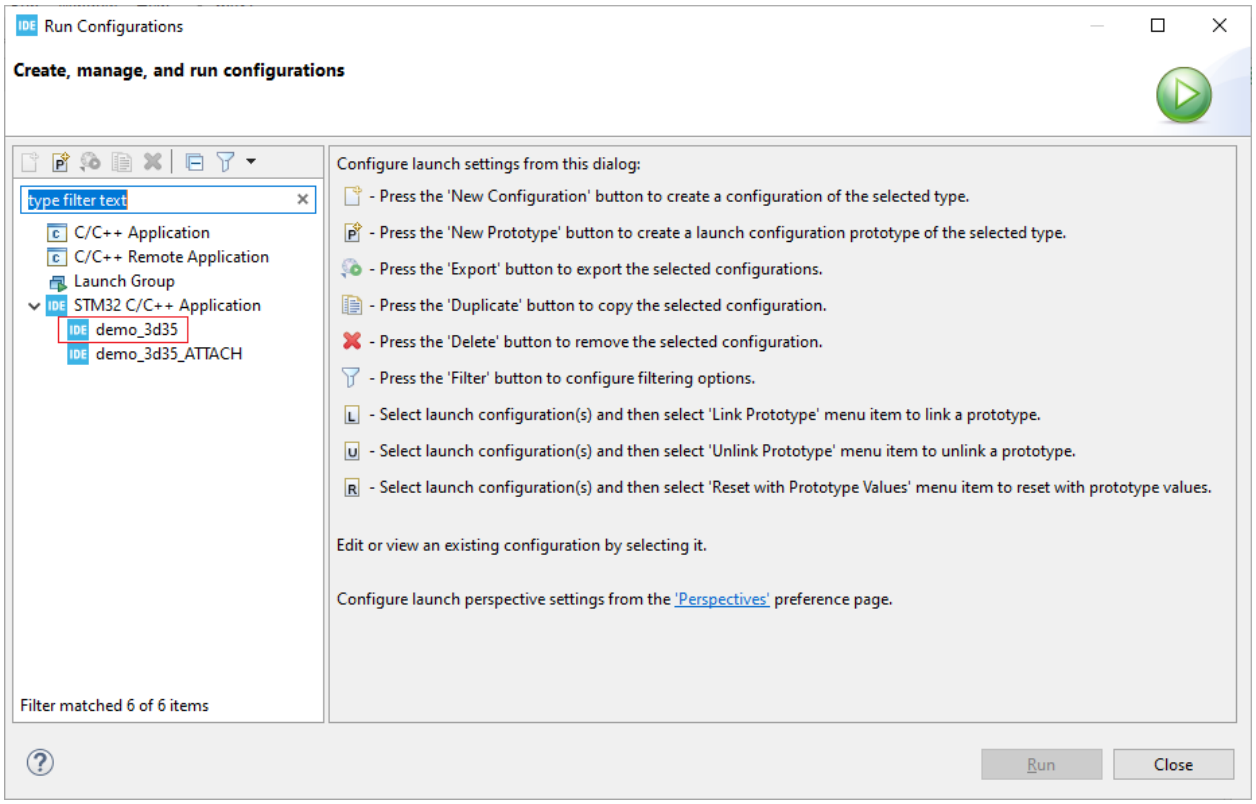


2. On the left side of the Run Configurations dialog box, click **STM32 C/C++ Application**. There should be two launch configurations available:

   - demo_3d35
   - demo_3d35_ATTACH

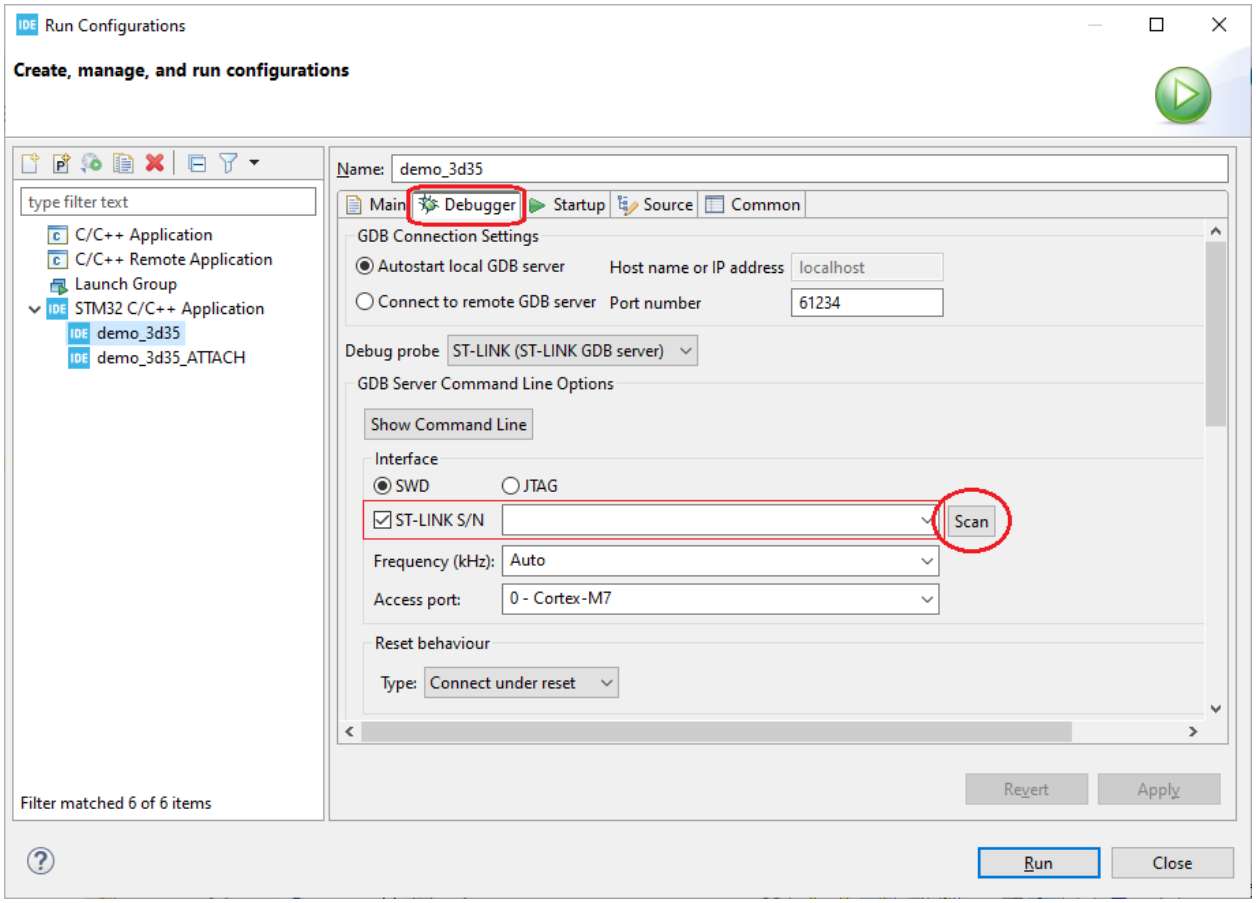
   The demo_3d35 launch configuration will download the application to the target display before running, while the demo_3d35_ATTACH will attach to an already running target.

   Select **demo_3d35**.



3. On the right side of the Run Configurations dialog box, click the **Debugger** tab.

   In the Interface box under GDB Server Command Line Options, find **ST-LINK S/N**.
   Click the **Scan** button. The STM32CubeIDE should find your STLINK/V3 and fill in the Serial Number.



4. Click **Apply** to accept the change.
5. Repeat steps 2-4 for the demo_3d35_ATTACH launch configuration.
6. Click **Close**.

7. From the STM32CubeIDE Main menu, click **Run** then select either:

   Run ← downloads and launches the demo application on the 3D35 display
   - or -
   Debug ← downloads the demo application to the 3D35 display and launches the debugger

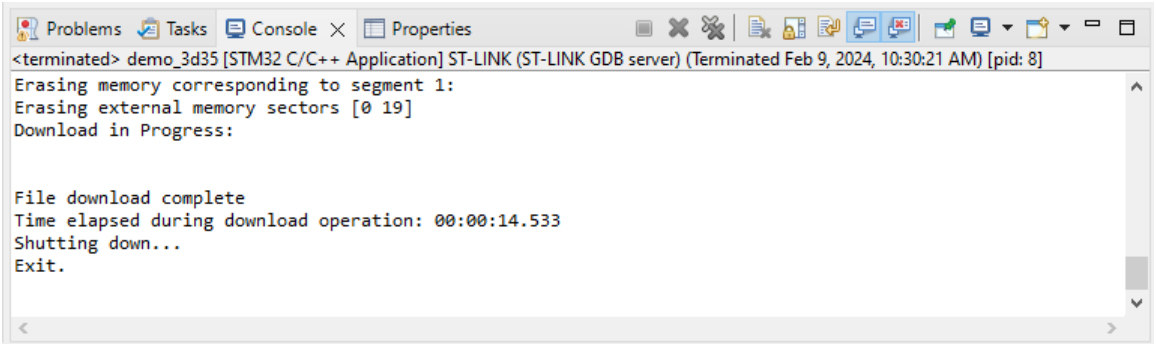
   Alternately, use the icons in the top menu bar:

   RUN 

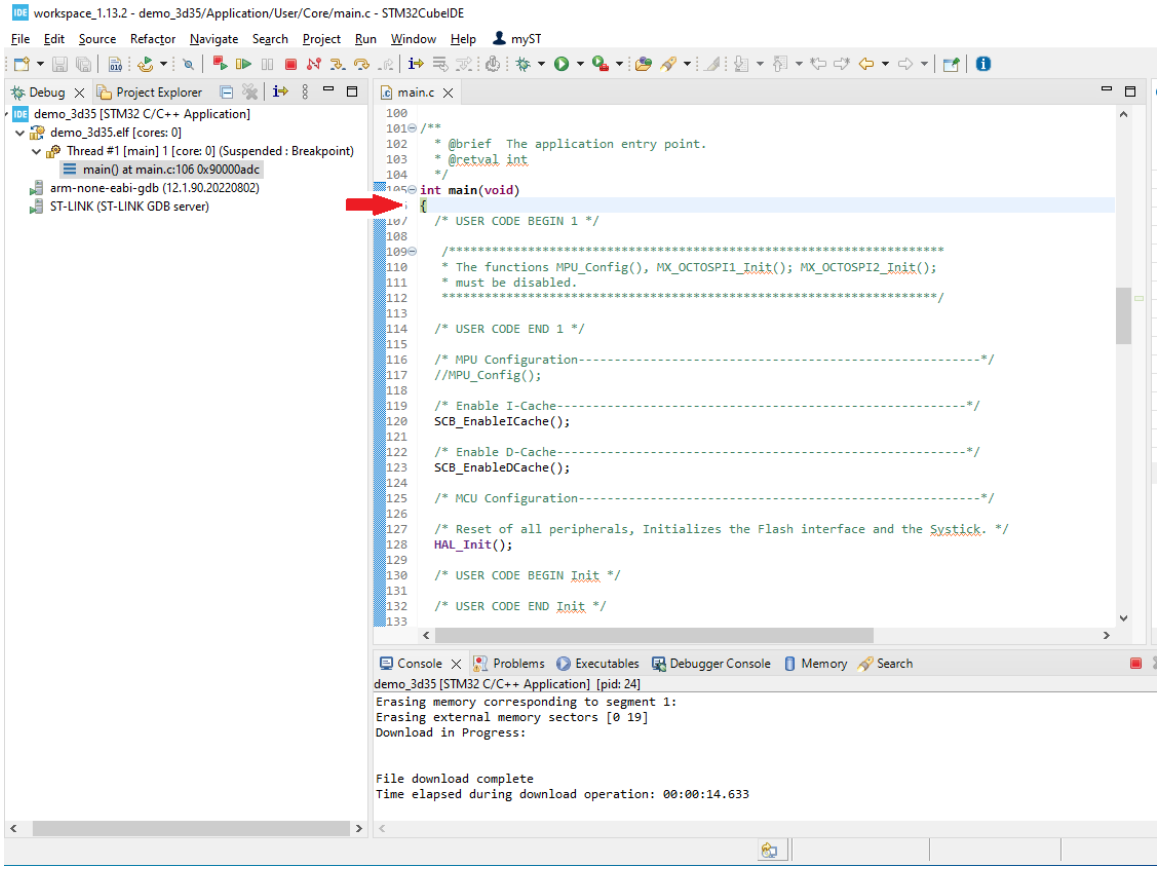
   DEBUG 

   When **Run** is clicked, the application will be downloaded to the 3D35 display.

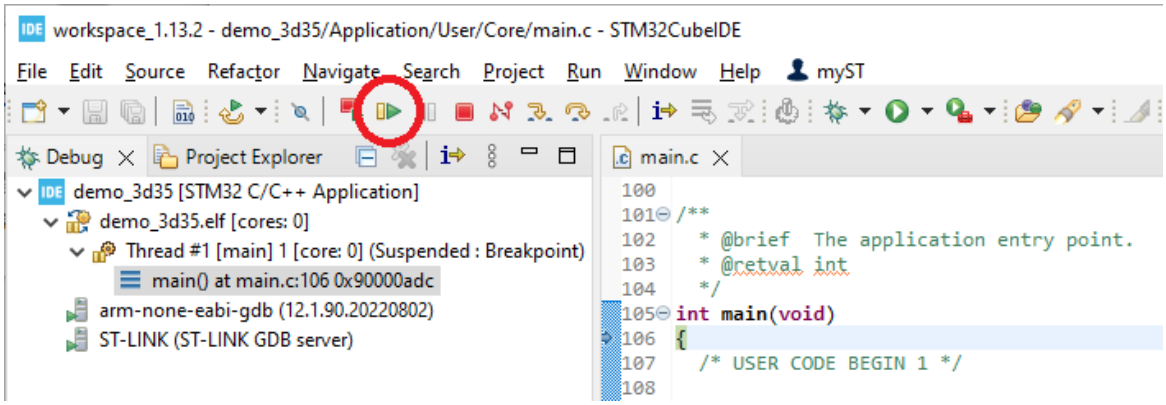
   When the download is finished the Console window will indicate “File download complete”, and the demo application should be running on the display.



When **Debug** is clicked, the application will be downloaded and the program will stop at a breakpoint at main().



Click the **Resume** button to continue running the application in the Debugger.



8. PROGRAM DEMO APPLICATION ON 3D35 DISPLAY

There are 3 different options that can be used for loading the demo application onto the 3D35 display.

1. STM32Cube IDE
2. STM32Cube Programmer
3. Grayhill CAN Tool

8.1 STM32CUBE IDE

When the STM32Cube IDE downloads the application to the 3D35 display, the application is programmed into Flash memory and thus remains loaded in the unit. See Section 7 for additional information about programming with the STM32Cube IDE.

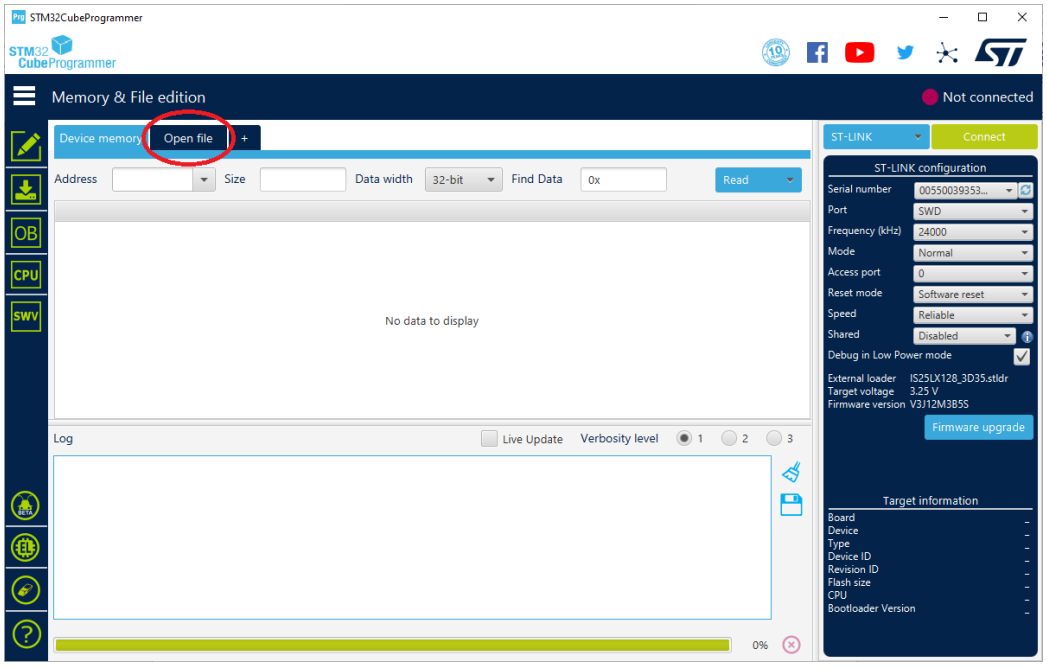
8.2 STM32CUBE PROGRAMMER

The STM32Cube Programmer can also be used to load the demo onto the 3D35 display.

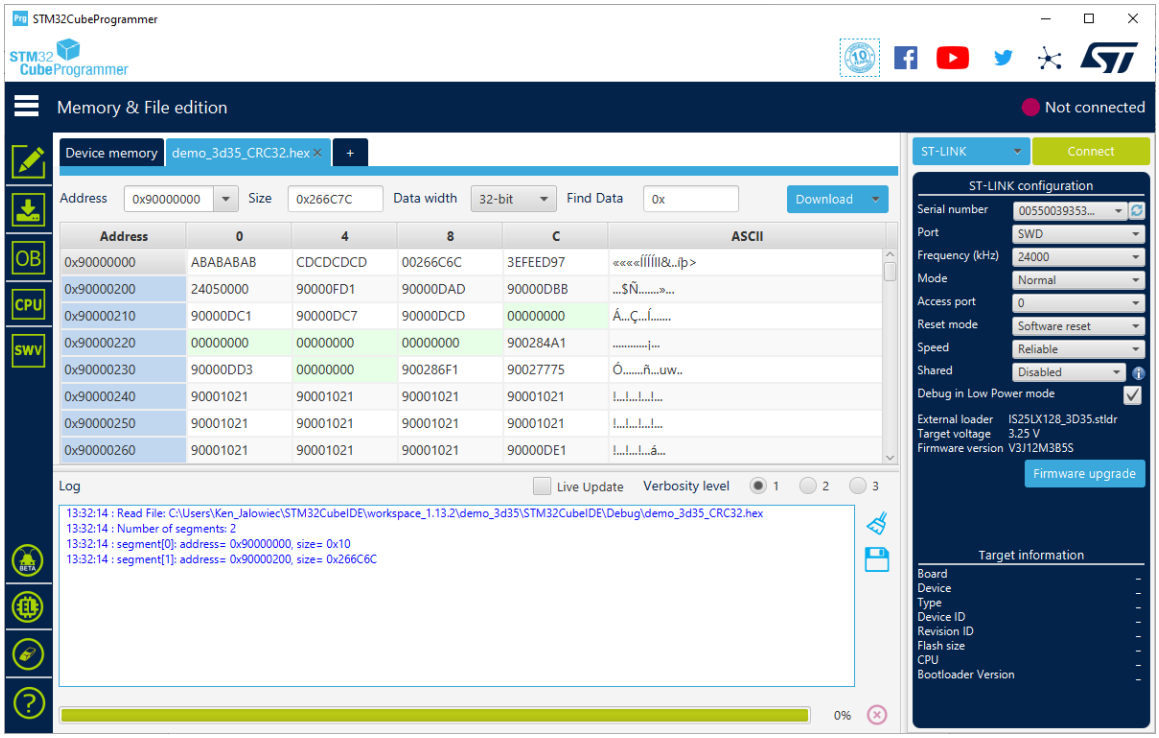
NOTE
There is a one-time setup that needs to be completed first. Please follow the instructions in Appendix A to complete the setup before proceeding.

The following steps describe how to load the demo application onto the 3D35 display using the STM32Cube Programmer.

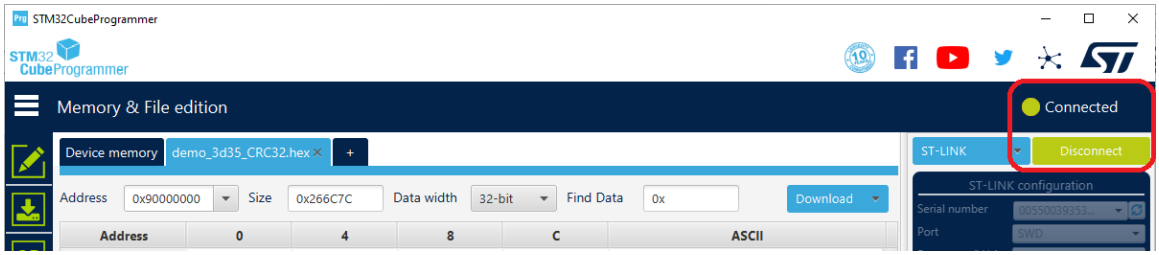
1. Launch the STM32Cube Programmer.
2. From the main screen, click the **Open file** tab.



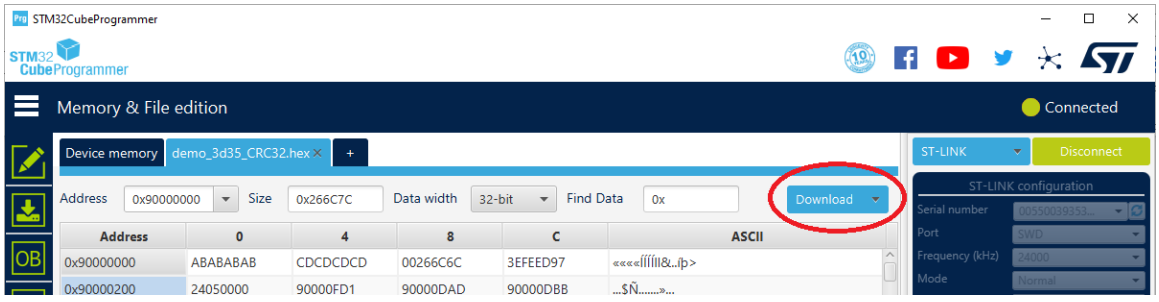
3. From the Open file dialog, navigate to the location of the successfully built demo application. Select **demo_3d35_CRC32.hex** and click **Open**. The result should look like the following:



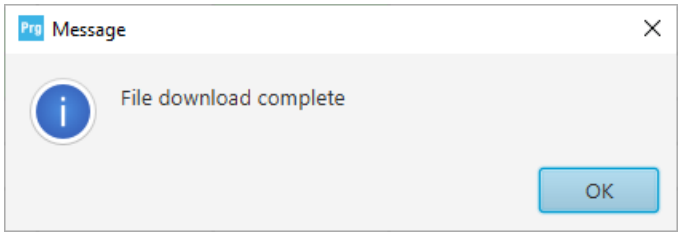
4. Click the **Connect** button in the upper right of the screen. The button text should change to “Disconnect” and the indicator above the button should change to green and say “Connected”.



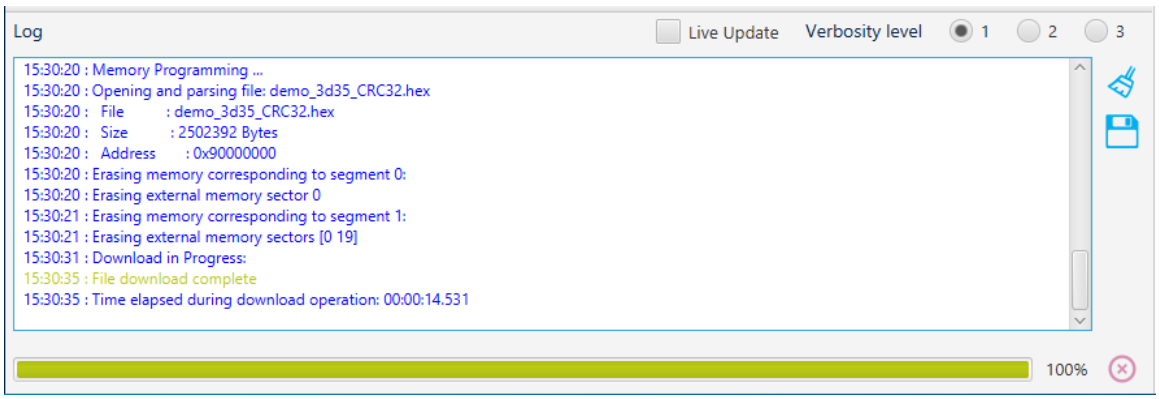
5. Click the blue **Download** button.



6. The application will be programmed to the 3D35 display. When programming is complete, a message box will display:



7. The Log section of the window should indicate programming is complete, with no errors shown.



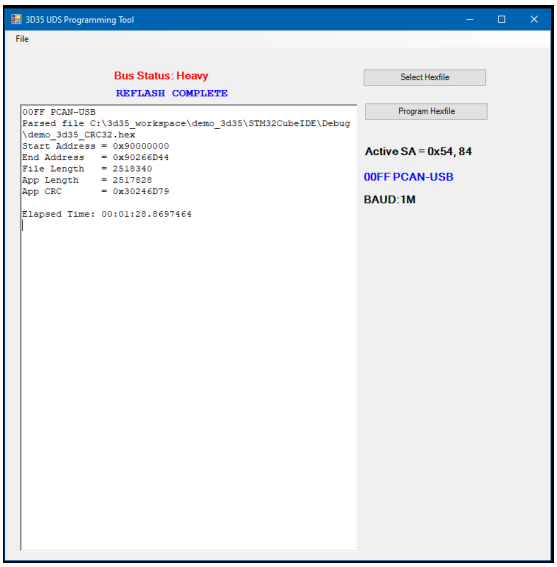
8. Click the **Disconnect** button, then power cycle the 3D35 display. The application should now be running!

8.3 GRAYHILL CAN TOOL

This section describes how to load the demo application onto the 3D35 display using the Grayhill UDS Downloader via can bus.

1. Have the 3D35 display connected to the same can bus as the PC. The 3DXX1314-1 cable and Peak PCAN adapter can be used for this connection.
2. Repower or reboot the 3D35 into bootloader mode. This is done by holding down buttons 1 and 2 on the display and then powering it up. Note that the capability to invoke the bootloader from the application is being refined and will be available soon.
3. Launch the Grayhill UDS Downloader.
4. The program automatically uses a device address of 0x54 and a baud rate of 1Mbit. Note that the capability to program at FD speeds is being refined and will be available soon.
5. Click **Select Hexfile** to select the hex file to download. (demo_3d35_CRC32.hex)
6. Click **Program Hexfile** to program the hex file.
7. When programming is complete the unit will launch the programmed application.

Programming underway



Programming completed

9. GRAYHILL DEMO APPLICATION OVERVIEW

The Grayhill demo application for the 3D35 display is a TouchGFX-based project. The purpose of the demo application is to illustrate some of the capabilities of the 3D35 display. It is intended as a reference for how to interface with the 3D35 display from a TouchGFX application.

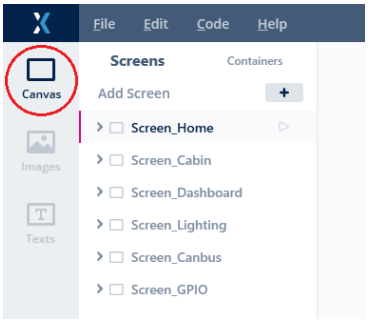
9.1 TOUCHGFX DESIGNER

This section will provide an overview of some of the basic areas in TouchGFX Designer as they relate to the demo application.

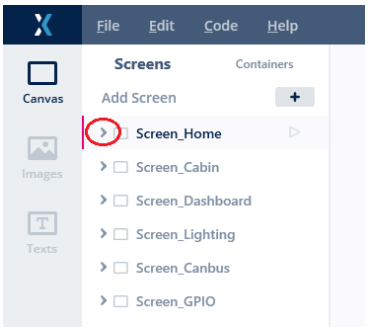
CANVAS
The Canvas window is opened by clicking on the **Canvas icon** on the upper left of the main window. Screens are defined in in the Canvas window.

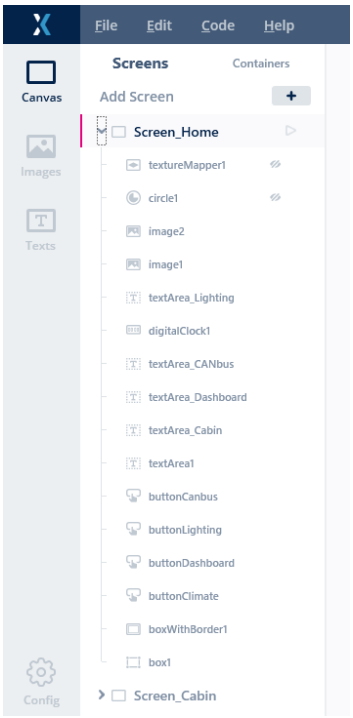
The demo consists of 6 screens:

- Screen_Home
- Screen_Cabin
- Screen_Dashboard
- Screen_Lighting
- Screen_Canbus
- Screen_GPIO

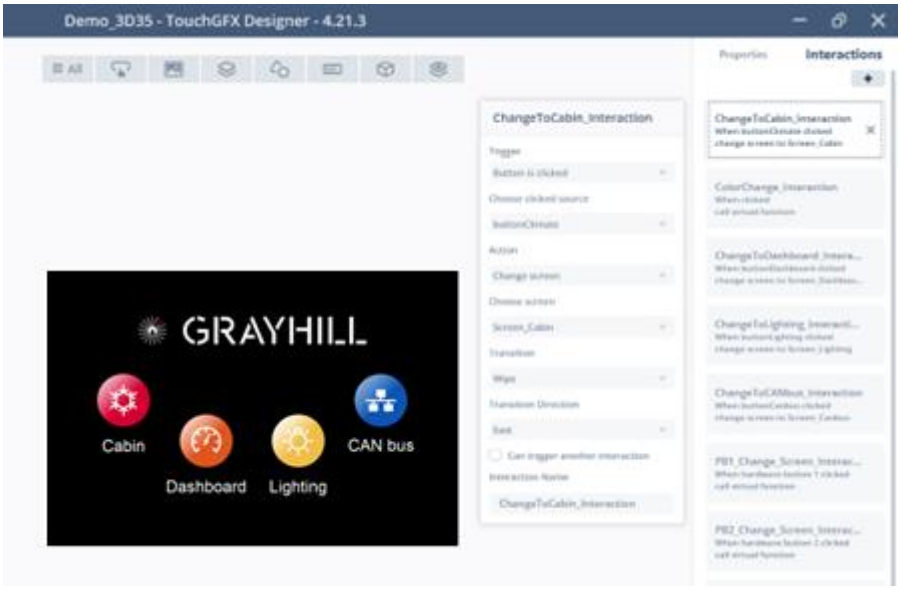


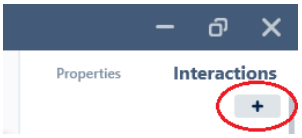
To view the various Widgets (e.g., boxes, text, images, buttons, etc.) assigned to a Screen, click on the **right arrow** to the left of the screen name.





When a Widget is selected, it is highlighted on the screen and its Properties are displayed in the window on the right side. In the Properties window, various features for the selected item are displayed; many are available to be modified.



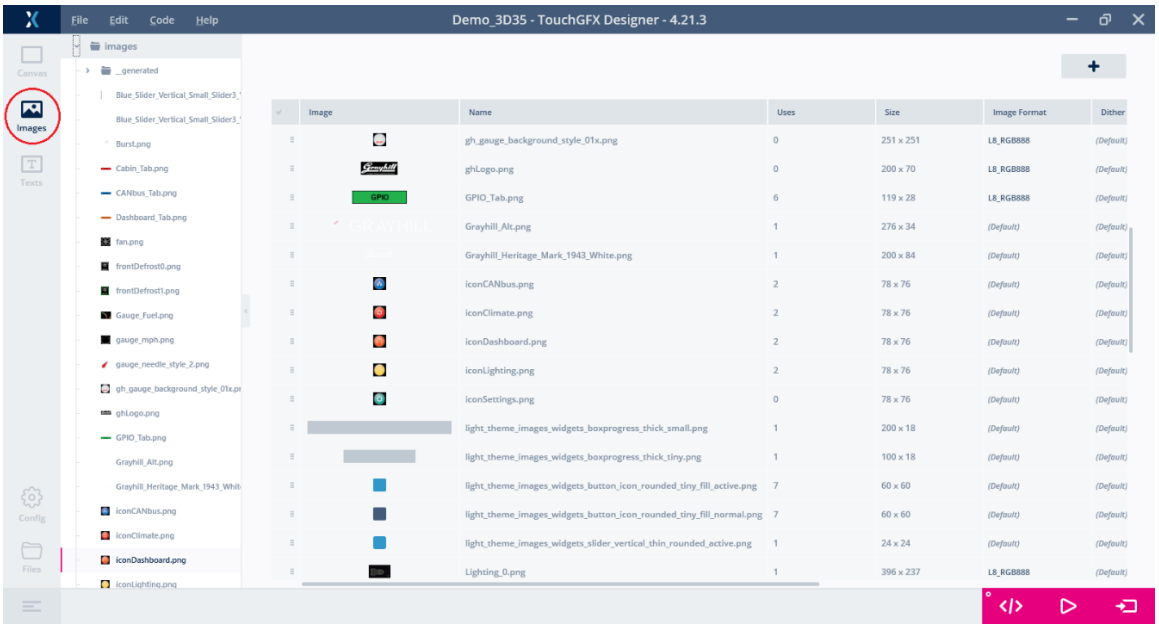


To define a new Interaction, click the **+** button under Interactions and configure with the desired selections.

Next to Properties, clicking on Interactions will bring up a window showing the Interactions assigned to the selected Widget. An Interaction defines an action to take for a particular Widget, like changing screens when a button is clicked. Clicking on one of the Interactions will pop up a window with the action assignments for that Interaction.

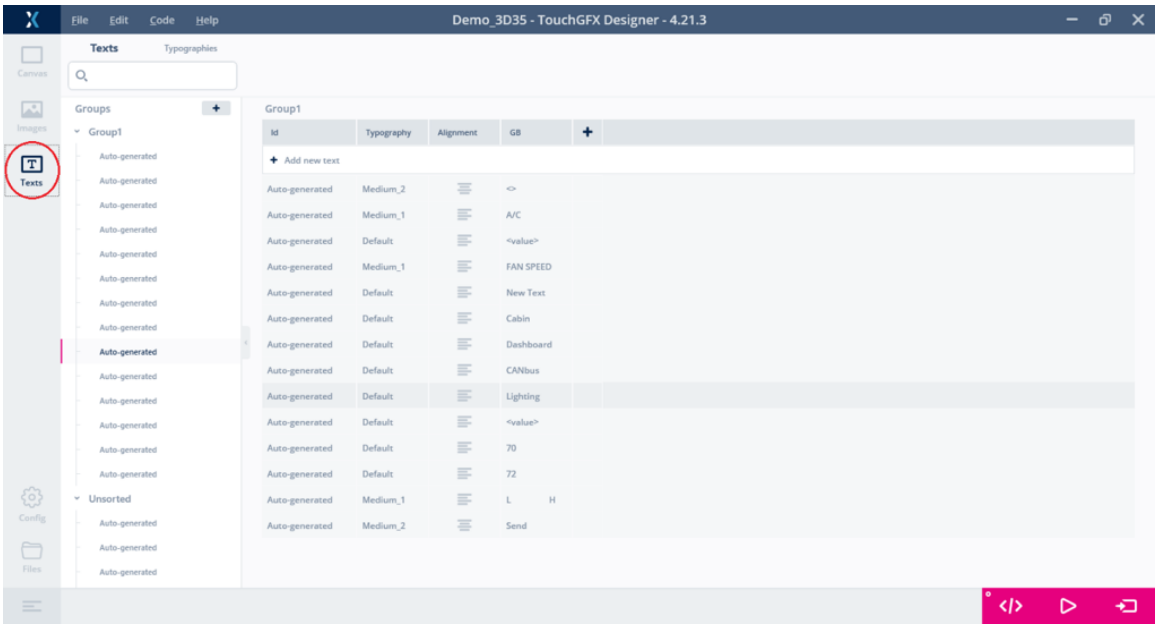
IMAGES

The Images window displays various information (e.g., resolution, format, # of uses, etc.) about the images that are assigned to Widgets.



TEXTS

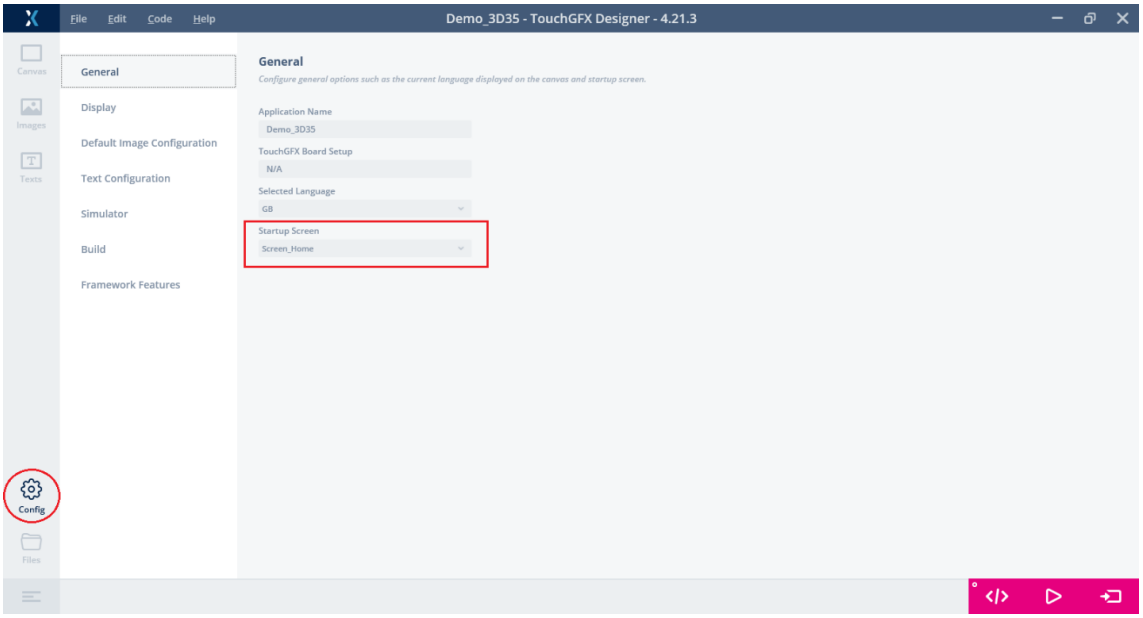
The Texts window displays various information (e.g., typography, alignment, etc.) about the text fields that are assigned to Widgets.



CONFIG

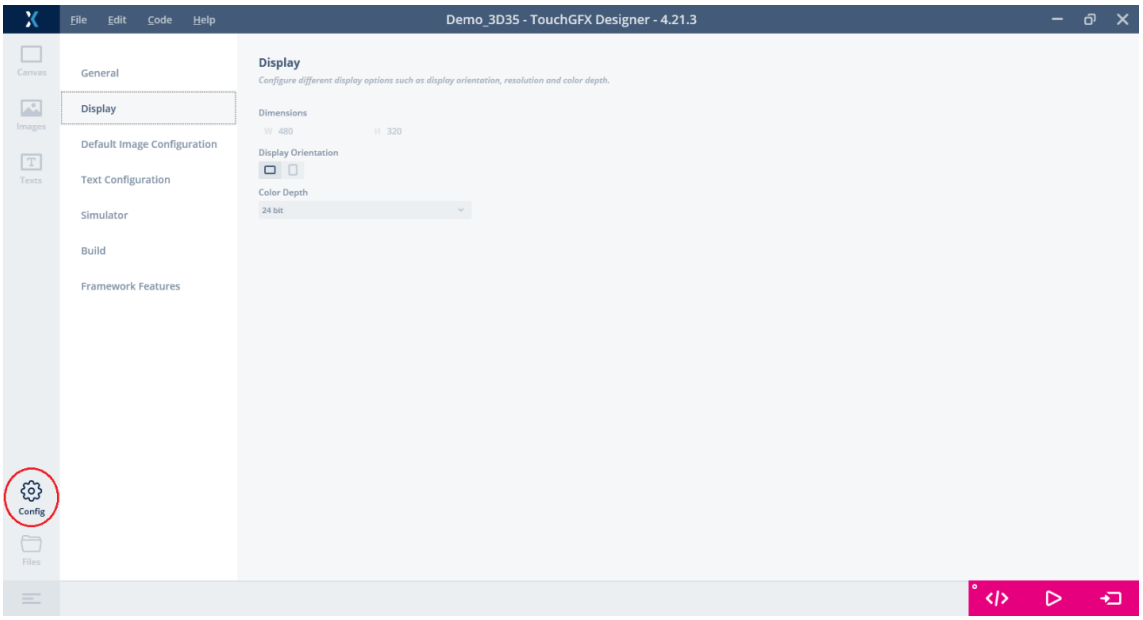
The Config window displays various Project configuration information. Following are some settings of note:

The General screen contains the Application Name and Startup Screen settings.

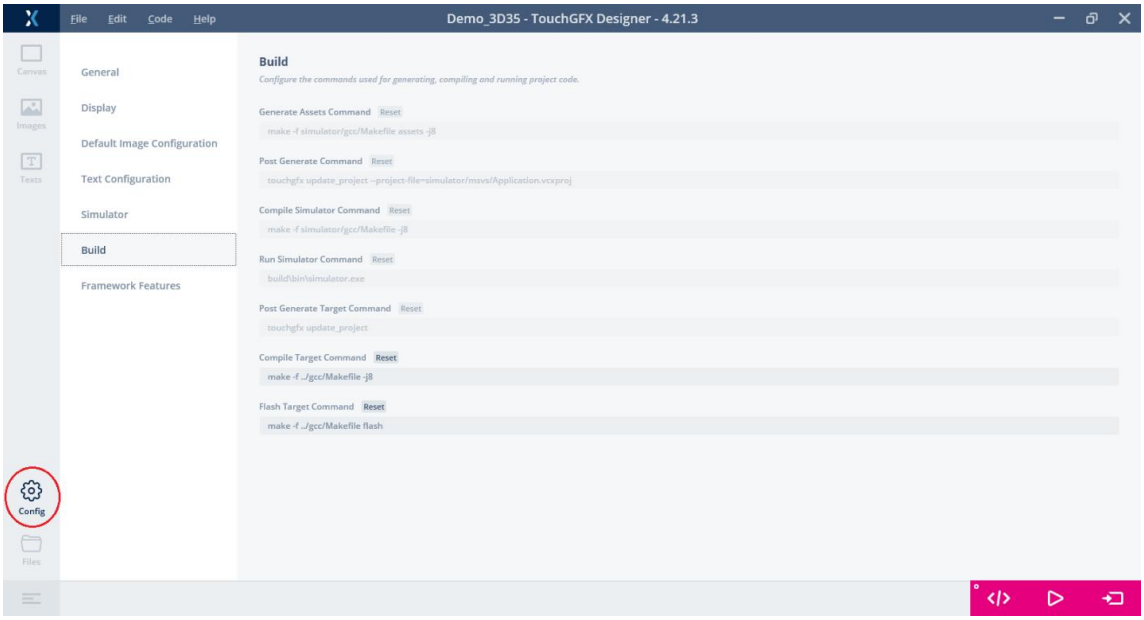


The Display screen contains settings for the 3D35.

- Dimensions = 480 x 320
- Orientation = Landscape
- Color Depth = 24 bit

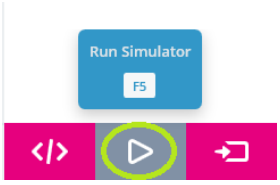


The Build screen contains commands for compiling and running the code.



9.2 TOUCHGFX SIMULATOR

To run the GUI application on the development machine, click the Run Simulator icon in the bottom right corner of the screen.



The main demo screen should appear:

10. GRAYHILL SOFTWARE LIBRARY

10.1 OVERVIEW

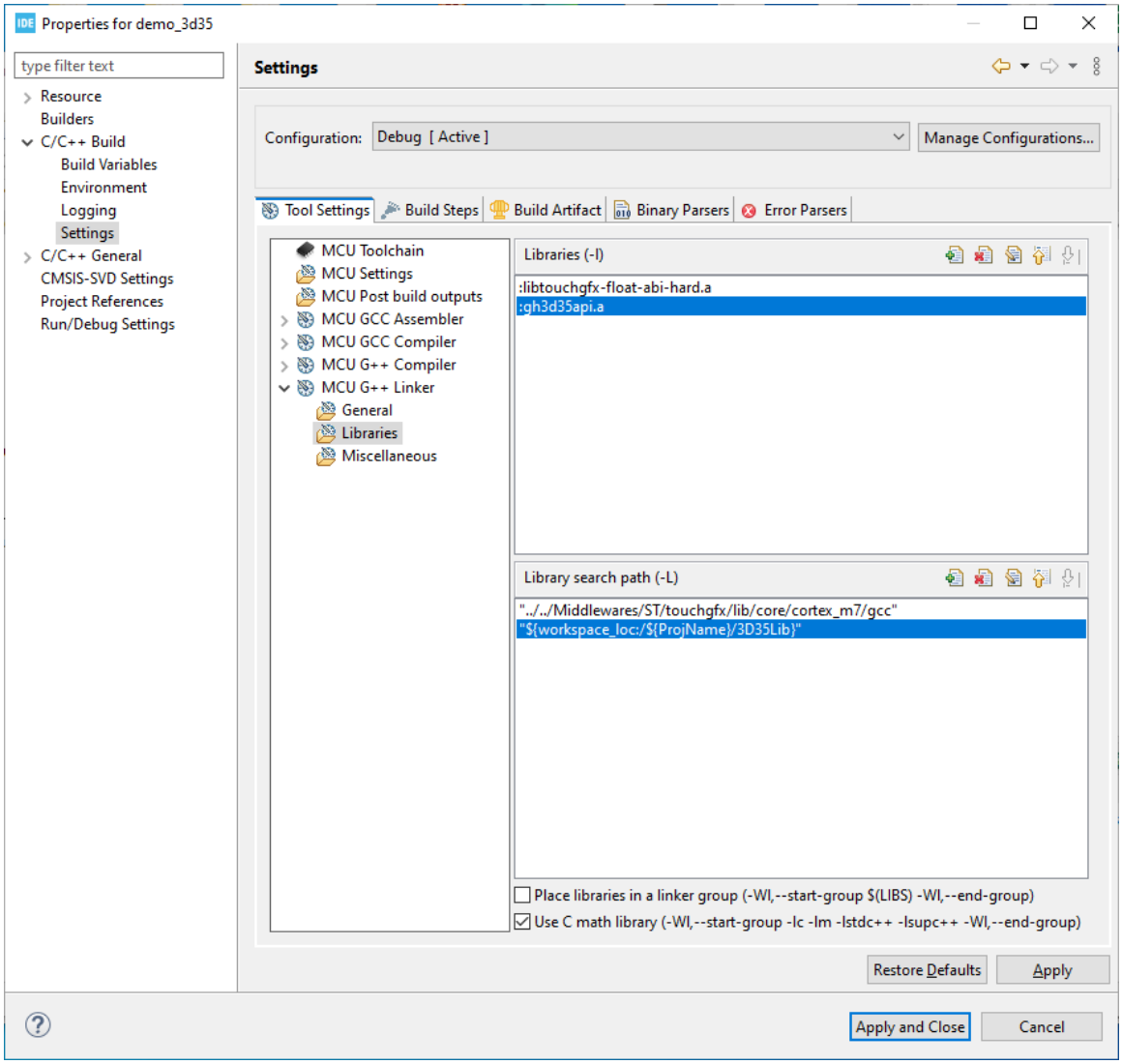
Grayhill provides a software library containing API functions for interfacing with the 3D35 display.
The following files are provided:

Static library	Headers
gh3d35api.a	3d35_api.h

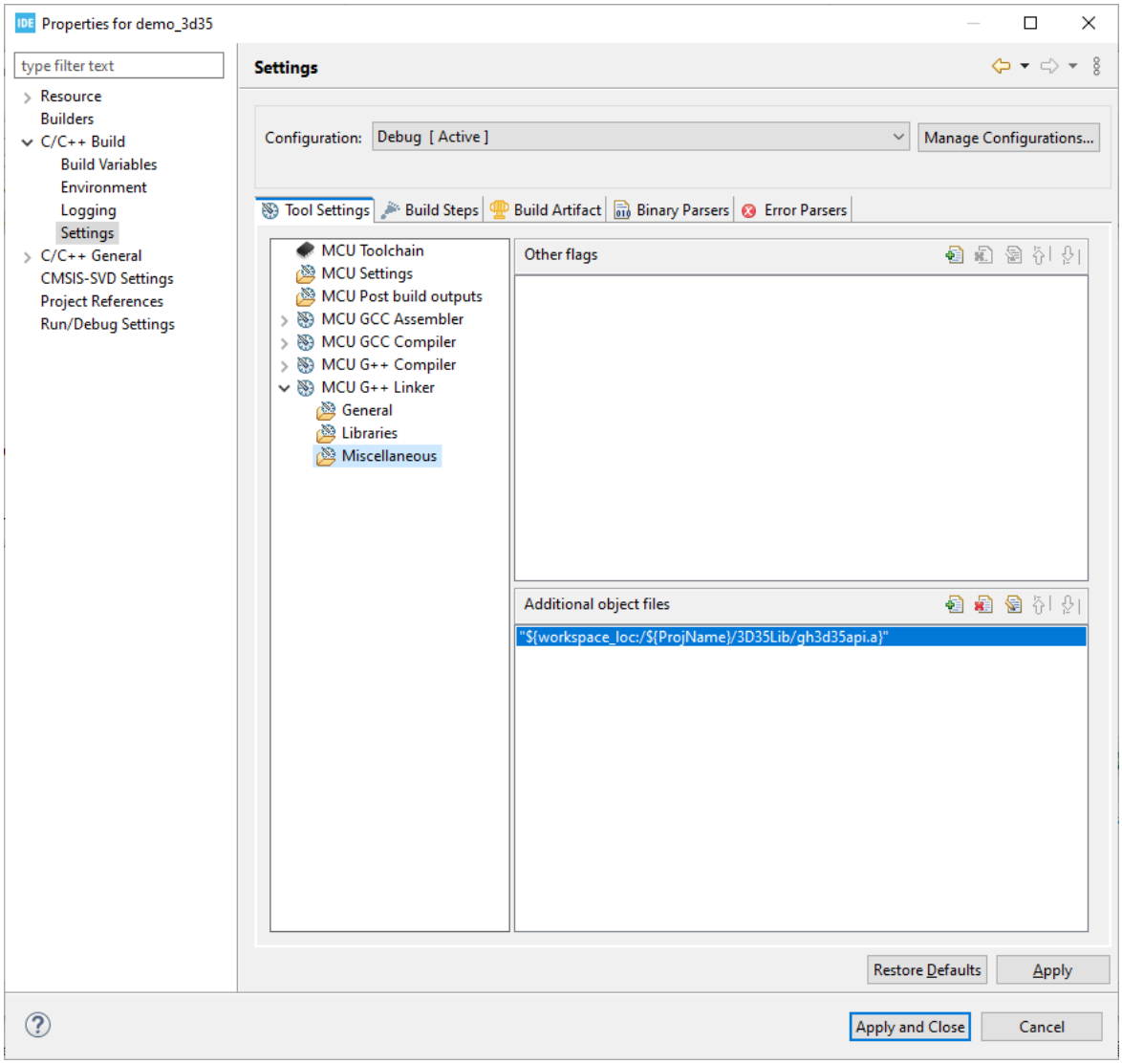
NOTE
The software library and header file have already been incorporated into the demo project and do not need to be added to the configuration. Therefore, the following section is meant for reference only as it has already been performed for the demo project.

10.2 CONFIGURATION

1. To use the library with your project, first copy the library file gh3d35api.a to a desired location within the project workspace.
2. In the STM32CubeIDE, open the Properties for your project. Under C/C++ Build, click on **Settings** and select the **Tool Settings** tab.
   - Under MCU G++ Linker, click on **Libraries**.
   - In the Libraries box, click on **Add** and enter the library file :gh3d35api.a
   - Click **OK**. The library entry should be added in the Libraries box.
   - In the Library search path box, click on **Add** and enter the path to your library file.
     For example: ..\STM32CubeIDE\workspace_1.13.2\3D35Lib\
   - Click **OK**. The library search path entry should be added in the Library search path box.



- Under MCU G++ Linker, click on **Miscellaneous**.
- In the Additional object files box, click on **Add** and enter the library file.
- Click **OK**. The library file entry should be added in the Additional object files box.



10.3 API LIBRARY INTERFACE

The Grayhill 3d35_api.h header file contains the API function declarations as well as some constant values. Include this header file for using the API library with an application. The following sections outline the API function calls based on the interface type.

API RETURN VALUES

```
typedef enum {
    API_ERR_NONE = 0 ,
    API_ERR_FAIL,
    API_ERR_CAN_INVALID_BAUD,
    API_ERR_CAN_INIT,
    API_ERR_CAN_START,
    API_ERR_NO_MSG,
    API_ERR_UNSUPPORTED_MSG,
    API_ERR_LAST
} api_err_t;
```

REQUIRED API FUNCTION CALLS

The following system-level API functions **must** be called by an application:

- API_SystemInit() — Call at startup to initialize the 3D35 hardware
- API_Task() — Call periodically in a task
- API_1msTick() — Call once every millisecond; can be called from a timer

10.3.1 SYSTEM FUNCTIONS

API LIST

API Function	Description
API_SystemInit	Initialize 3D35 hardware peripherals **(required)**
API_Task	API Task routine **(required)** — must be called periodically by application
API_1msTick	API 1 ms tick routine **(required)** — must be called once per millisecond
API_GetVersion	Read Library version (Major, Minor, Revision)
API_ResetChip	Reset the unit processor
API_DelayedReset	Reset the unit processor after specified delay time

C SYNTAX

void API_SystemInit (*void*)

Parameters
none

Return Value
none

void API_Task (*void*)

Parameters
none

Return Value
none

void API_1msTick (*void*)

Parameters
none

Return Value
none

uint32_t API_GetVersion (*void*)

Parameters
none

Return Value	Description	Value Range
uint32_t	32-bit value containing 8-bit fields for Major, Minor, and Revision	Major: bits 16-23 Minor: bits 8-15 Revision: bits 0-7

void API_ResetChip (*void*)

Parameters
none

Return Value
none

void API_DelayedReset (*void*)

Parameters
none

Return Value	Description	Value Range
uint32_t	32-bit value containing 8-bit fields for Major, Minor, and Revision	Major: bits 16-23 Minor: bits 8-15 Revision: bits 0-7

void API_DelayedReset (*void*)

Parameters
delay_ms

Return Value
none

10.3.2 BACKLIGHT INTERFACE

The backlight interface contains API functions for controlling the LCD and keypad button LEDs.

API LIST

API Function	Description
API_SetLCDBacklight	Set the brightness of the screen
API_SetButtonBacklight	Set the brightness of the button LED's
API_SetRedLED	Set the brightness of the Red LED
API_SetAmberLED	Set the brightness of the Amber LED

C SYNTAX

***void* API_SetLCDBacklight (*uint8_t* level)**

Parameters	Description	Value Range
level	LCD backlight level (%)	0 – 100

Return Value

none

***void* API_SetButtonBacklight (*uint8_t* level)**

Parameters	Description	Value Range
level	Button backlight level (%)	0 – 100

Return Value

none

***void* API_SetRedLED (*uint8_t* level)**

Parameters	Description	Value Range
level	Red LED level (%)	0 – 100

Return Value

none

***void* API_SetAmberLED (*uint8_t* level)**

Parameters	Description	Value Range
level	Amber LED level (%)	0 – 100

Return Value

none

10.3.3 BUTTON INTERFACE

The button interface contains API functions for reading the state of each keypad button.

API LIST

API Function	Description
API_GetBtn1	Get button 1 state
API_GetBtn2	Get button 2 state
API_GetBtn3	Get button 3 state
API_GetBtn4	Get button 4 state

C SYNTAX

***uint8_t* API_GetBtn1 (*void*)**
***uint8_t* API_GetBtn2 (*void*)**
***uint8_t* API_GetBtn3 (*void*)**
***uint8_t* API_GetBtn4 (*void*)**

Parameters

none

Return Value	Description	Value Range
uint8_t	Button state	0 (Up) 1 (Down)

10.3.4 ANALOG INTERFACE

API LIST

API Function	Description
API_ReadAnalogIn1	Read the value of analog input 1
API_ReadAnalogIn2	Read the value of analog input 2
API_ReadAnalogIn3	Read the value of analog input 3
API_ReadVPower	Read the input power voltage
API_ReadVRef	Read the reference voltage

C SYNTAX

***uint16_t* API_ReadAnalogIn1 (*void*)**
***uint16_t* API_ReadAnalogIn2 (*void*)**
***uint16_t* API_ReadAnalogIn3 (*void*)**

Parameters

none

Return Value	Description	Value Range
uint16_t	Analog voltage	0 – 10000 (millivolts)

***uint16_t* API_ReadVPower (*void*)**

Parameters

none

Return Value	Description	Value Range
uint16_t	Input power voltage	0 – 32000 (millivolts)

***uint16_t* API_ReadVRef (*void*)**

Parameters

none

Return Value	Description	Value Range
uint16_t	Input reference voltage	0 – 32000 (millivolts)

10.3.5 DIGITAL I/O INTERFACE

API LIST

API Function	Description
API_ReadDigitalIn1	Read the state of digital input 1
API_ReadDigitalIn2	Read the state of digital input 2
API_ReadDigitalIn3	Read the state of digital input 3
API_SetDigitalOut1	Set the state of digital output 1
API_SetDigitalOut2	Set the state of digital output 2

C SYNTAX

***uint8_t* API_ReadDigitalIn1 (*void*)**
***uint8_t* API_ReadDigitalIn2 (*void*)**
***uint8_t* API_ReadDigitalIn3 (*void*)**

Parameters
none

Return Value	Description	Value Range
uint8_t	Input State	0 (OFF) 1 (ON)

***void* API_SetDigitalOut1 *(uint8_t* state)**
***void* API_SetDigitalOut2 *(uint8_t* state)**

Parameters	Description	Value Range
state	Output State	0 (OFF) 1 (ON)

Return Value
none

10.3.6 CAN BUS INTERFACE

API LIST

API Function	Description
API_Init_CAN1	Initialize CAN 1
API_Init_CAN2	Initialize CAN 2
API_IsBitrateValid	Check if specified bit rate is valid
API_IsCAN1Initialized	Check if CAN 1 is initialized
API_IsCAN2Initialized	Check if CAN 2 is initialized
API_CAN1Get	Read a message from the CAN 1 RX buffer
API_CAN2Get	Read a message from the CAN 2 RX buffer
API_CAN1Send	Send a message to the CAN 1 TX buffer
API_CAN2Send	Send a message to the CAN 2 TX buffer
API_IsCAN1Ready	Check if a message is ready to transmit on CAN 1
API_IsCAN2Ready	Check if a message is ready to transmit on CAN 2
API_IsCAN1FDEnabled	Check if CAN FD protocol is enabled for CAN 1
API_IsCAN2FDEnabled	Check if CAN FD protocol is enabled for CAN 2
API_GetCAN1ActiveBitTiming	Get the bit rate currently assigned for CAN 1
API_GetCAN2ActiveBitTiming	Get the bit rate currently assigned for CAN 2

C SYNTAX

***api_err_t* API_Init_CAN1 (*uint16_t* nominal bitrate, *uint16_t* data bitrate)**
***api_err_t* API_Init_CAN2 (*uint16_t* nominal bitrate, *uint16_t* data bitrate)**

Parameters	Description	Values
nominal bitrate	Bit rate for CAN Classic frame and arbitration phase of CAN FD frame	125 = 125 kBit/s 250 = 250 kBit/s 500 = 500 kBit/s 800 = 800 kBit/s 1000 = 1 Mbit/s
data bitrate	Bit rate for the data phase of CAN FD frame. Applies to CAN FD only; must be 0 for Classic CAN	[CAN Classic] 0 [CAN FD] 2000 = 2 Mbit/s 4000 = 4 Mbit/s 5000 = 5 Mbit/s
Return Value	**Description**	**Values**
api_err_t	Error Code	API_ERR_NONE API_ERR_FAIL API_ERR_CAN_INVALID_BAUD API_ERR_CAN_INIT API_ERR_CAN_START

***api_err_t* API_IsBitrateValid (*uint16_t* nominal bitrate, *uint16_t* data bitrate)**

Parameters	Description	Values
nominal bitrate	Bit rate for CAN Classic frame and arbitration phase of CAN FD frame	125 = 125 kBit/s 250 = 250 kBit/s 500 = 500 kBit/s 800 = 800 kBit/s 1000 = 1 Mbit/s
data bitrate	Bit rate for the data phase of CAN FD frame. Applies to CAN FD only; must be 0 for Classic CAN	[CAN Classic] 0 [CAN FD] 2000 = 2 Mbit/s 4000 = 4 Mbit/s 5000 = 5 Mbit/s
Return Value	**Description**	**Values**
uint8_t	Valid/Invalid	0 = Not Valid 1 = Valid

***uint8_t* API_IsCAN1Initialized (*void*)**
***uint8_t* API_IsCAN2Initialized (*void*)**

Parameters		
none		
Return Value	**Description**	**Values**
uint8_t	CAN initialization status	0 = Not Initialized 1 = Initialized

***api_err_t* API_CAN1Get (*uint32_t* id, uint8_t* dlc, uint8_t* type, uint8_t* dta*)**
***api_err_t* API_CAN2Get (*uint32_t* id, uint8_t* dlc, uint8_t* type, uint8_t* dta*)**

Parameters	Description	Values
id	Pointer to variable to receive CAN Id	pointer to: [0 – 1FFFFFFFh]
dlc	Pointer to variable to receive CAN data length	pointer to: [0 – 255]
type	Pointer to variable to receive CAN message type	pointer to: [API_MSG_TYPE_STD = Standard API_MSG_TYPE_EXT = Extended]
dta	Pointer to CAN data buffer	
Return Value	**Description**	**Values**
api_err_t	Error Code	API_ERR_NONE = Message API_ERR_NOMSG = No message API_ERR_UNSUPPORTED_MSG = Error

***api_err_t* API_CAN1Send (*uint32_t id, uint8_t dlc, uint8_t* dta*)**
***api_err_t* API_CAN2Send (*uint32_t id, uint8_t dlc, uint8_t* dta*)**

Parameters	Description	Values
id	CAN Id	0 – 1FFFFFFFh
dlc	CAN data length	0 – 255
dta	Pointer to CAN data buffer	
Return Value	**Description**	**Values**
api_err_t	Error Code	0 = OK Non-zero = Error

***uint8_t* API_IsCAN1Ready (*void*)**
***uint8_t* API_IsCAN2Ready (*void*)**

Parameters
none

Return Value	Description	Values
uint8_t	message ready to transmit	0 = No 1 = Yes

***uint8_t* API_IsCAN1FDEnabled (*void*)**
***uint8_t* API_IsCAN2FDEnabled (*void*)**

Parameters
none

Return Value	Description	Values
uint8_t	CAN FD status	0 = Not Enabled 1 = Enabled

***void* API_GetCAN1ActiveBitTiming (*uint16_t* *p_bt_nom, *uint16_t* *p_bt_dta)**
***void* API_GetCAN2ActiveBitTiming (*uint16_t* *p_bt_nom, *uint16_t* *p_bt_dta)**

Parameters	Description	Values
p_bt_nom	Pointer to nominal bit rate	[refer to API_InitCANx values]
p_bt_dta	Pointer to data phase bit rate	[refer to API_InitCANx values]

Return Value
none

10.3.7 RTC (REAL-TIME CLOCK) INTERFACE

API LIST

API Function	Description
API_GetRTC	Get date and time from RTC
API_SetRTC	Set RTC date and time

DATA TYPES

The time structure used for the RTC API functions is defined in time.h.

Note that the last 3 parameters `tm_wday, tm_yday, tm_isdst` are not supported by the API.

```
struct tm {
  int tm_sec;     /* seconds */
  int tm_min;     /* minutes */
  int tm_hour;    /* hours */
  int tm_mday;    /* day of the month */
  int tm_mon;     /* month */
  int tm_year;    /* year */
  int tm_wday;    /* day of the week */
  int tm_yday;    /* day in the year */
  int tm_isdst;    /* daylight saving time */
}
```

For the RTC API functions, the supported parameters for the tm structure are shown in the following table.

tm Parameter	Values
tm_sec	0 – 59
tm_min	0 – 59
tm_hour	0 – 23
tm_mday	1 – 31
tm_mon	1 – 12
tm_year	(year – 2000)
tm_wday	NOT SUPPORTED
tm_yday	NOT SUPPORTED
tm_isdst	NOT SUPPORTED

C SYNTAX

***api_err_t* API_GetRTC (*struct tm *t*)**

Parameter	Description	Values
t	Pointer to time structure variable; Refer to time structure tm	
Return Value	**Description**	**Values**
api_err_t	Error Code	API_ERR_NONE = OK API_ERR_FAIL = Error

***api_err_t* API_SetRTC (*struct tm *t*)**

Parameter	Description	Values
t	Pointer to time structure variable; Refer to time structure tm	
Return Value	**Description**	**Values**
api_err_t	Error Code	API_ERR_NONE = OK API_ERR_FAIL = Error

10.3.8 EEPROM

API LIST

API Function	Description
API_EERead	Read from EEPROM
API_EEWrite	Write to EEPROM

C SYNTAX

***uint8_t* API_EERead (*void* * buf, *uint16_t* addr, *uint16_t* sz)**

Parameters	Description	Values
buf	Pointer to buffer that will receive data	
addr	EEPROM address to read	0 – 65535
sz	Number of bytes to read	0 – 65535
Return Value	**Description**	**Values**
api_err_t	Error Code	API_ERR_NONE = OK API_ERR_FAIL = Error

***uint8_t* API_EEWrite (*uint16_t* addr, *void* * buf, *uint16_t* sz)**

Parameters	Description	Values
addr	EEPROM address to write	0 – 65535
buf	Pointer to buffer containing data to write	
sz	Number of bytes to write	0 – 65535
Return Value	**Description**	**Values**
api_err_t	Error Code	API_ERR_NONE = OK API_ERR_FAIL = Error

10.3.9 BOOTLOADER AND SHARED RAM

API LIST

API Function	Description
API_IsSharedRamValid	Check if shared RAM is valid
API_ClearSharedRam	Clear the shared RAM
API_ValidateSharedRam	Validate the shared RAM
API_GetFBLVersion	Get the Flash Bootloader version
API_GetFBLString	Get the Flash Bootloader string
API_GetFBLCompileDate	Get the Flash Bootloader compile date

C SYNTAX

***uint8_t* API_IsSharedRamValid (*void*)**

Parameters		
none		
Return Value	**Description**	**Values**
uint8_t	Shared RAM status	0 = Not Valid 1 = Valid

***void* API_ClearSharedRam (*void*)**

Parameters
none
Return Value
none

***void* API_ValidateSharedRam (*void*)**

Parameters
none
Return Value
none

***uint32_t* API_GetFBLVersion (*void*)**

Parameters

none

Return Value	Description	Values
uint32_t	Bootloader version	

***uint32_t* API_GetFBLString (*uint8_t* **pstr)**

Parameters	Description	Values
pstr	Pointer to buffer that will receive data string	[64 bytes]
Return Value	**Description**	**Values**
uint16_t	Flash Bootloader string	API_SHARED_STRING_LENGTH

***uint32_t* API_GetFBLCompileDate (*uint8_t* **pstr)**

Parameters	Description	Values
pstr	Pointer to buffer that will receive version string	[64 bytes]
Return Value	**Description**	**Values**
uint16_t	Flash Bootloader compile date	API_SHARED_STRING_LENGTH

10.3.10 UDS

API LIST

API Function	Description
API_ResetUDS	Called by the UDS stack when command received from tool to initiate a device reset
API_InitiateBootloaderUDS	Called by the UDS stack when reflashing via tool; transfers device address and tool source address to bootloader.

C SYNTAX

void API_ResetUDS (*void*)

Parameters

none

Return Value

none

void API_InitiateBootloaderUDS (*void*)

Parameters

none

Return Value

none

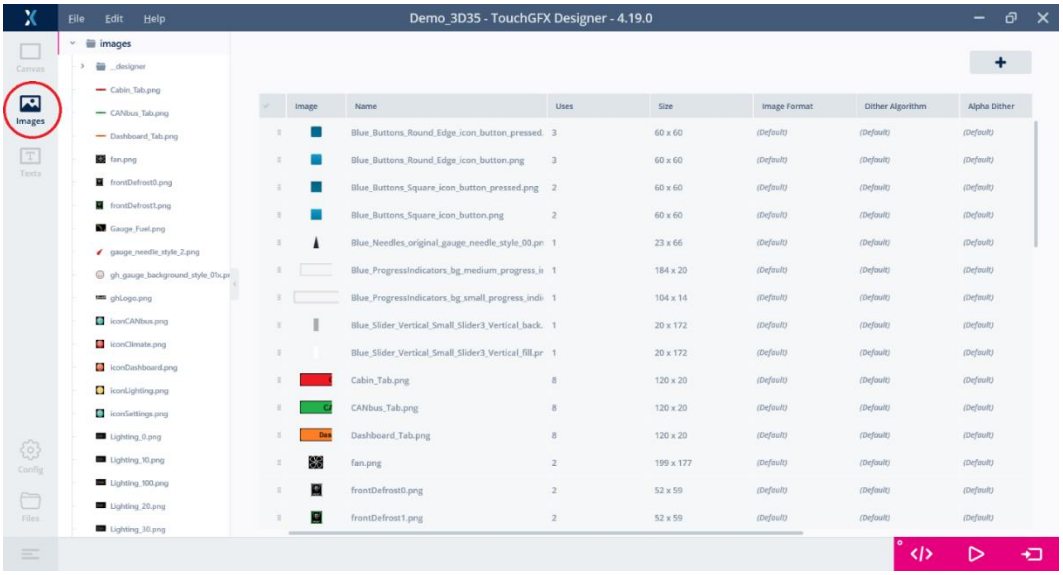
11. TOUCHGFX PROGRAMMING TIPS

11.1 IMAGE FORMAT

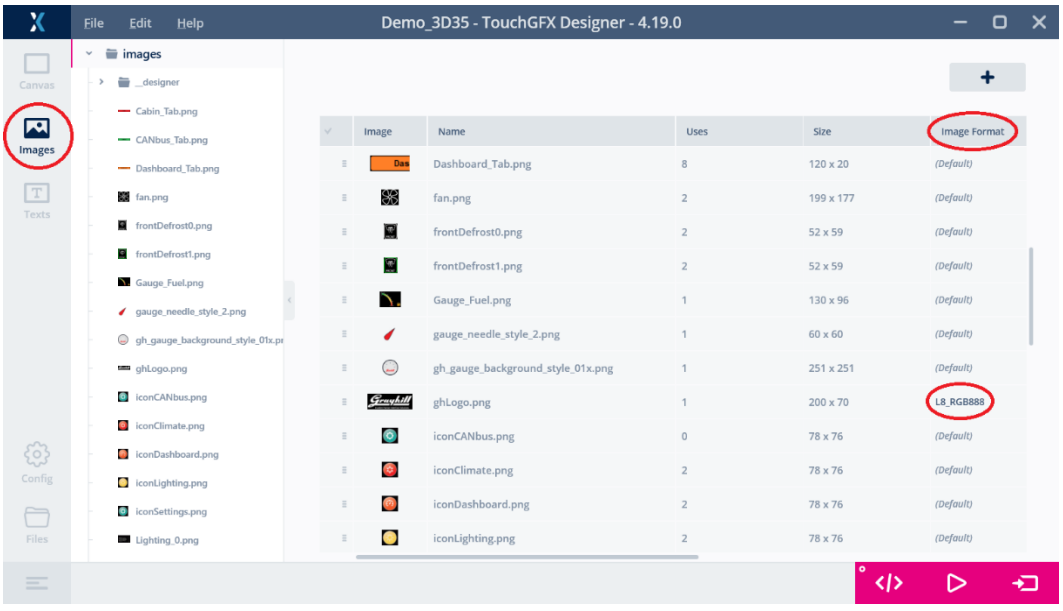
When developing an application for the 3D35 display, it is important to keep in mind the size (bytes) of the binary file that is produced. One way of doing this is to minimize the size of the images being used.

TouchGFX Image Converter converts .png files to bitmaps and generates source (cpp/hpp) files.

In TouchGFX Designer, click on the **Images** icon on the left side of the screen.



In this screen, you can change the Format in which the image is saved.



24-bit color would be RGB888 image format.

If the image data contains transparency information, then the image format needs to include alpha channel. To include the alpha channel, save the image format as ARGB8888.

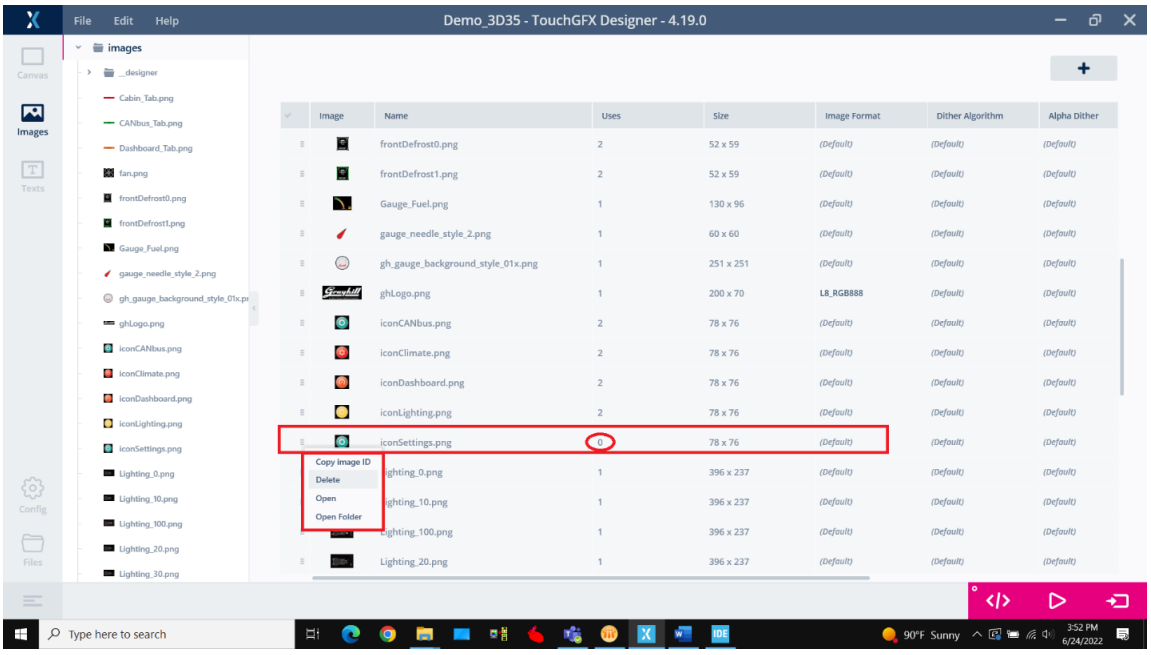
To reduce storage size, the image can also be saved using L8 compression (L8_RGB888, L8_ARGB8888).

EXAMPLE
ghLogo.png

Image File Format	Approx. Storage Size (Bytes)
.png	7K
256 color .bmp	15K
24-bit .bmp	42K

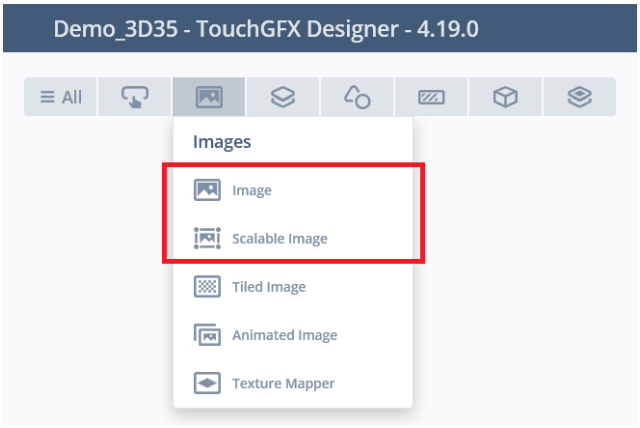
TouchGFX Image Format	Approx. Storage Size (Bytes)
L8_RGB888	17K
RGB888	44K
Default	44K
ARGB8888	57K
L8_ARGB8888	17K

Images that are no longer used in the project should be removed. If ‘Uses’ shows 0, click on the **menu** icon in the left-hand check-box column, and click **Delete**.



11.2 IMAGE VS. SCALABLE IMAGE WIDGETS

Suppose you have a .png image file that you would like to use with a widget, but the image resolution is too large. A Scalable Image widget can be used to reduce the image to the desired size. However, from a storage standpoint, it may be a better option to first scale the image down to the desired size and instead use an Image widget. Using a Scalable Image widget would require more flash space because the image is saved in its original size.



12. STM32CUBE IDE PROGRAMMING TIPS

12.1 SIMULATOR

For building and running the application in the Simulator, it may be necessary to exclude hardware interface references. Use the SIMULATOR #define directive for conditional compiling.

For example, to exclude code that is not supported in the simulator:

```
#ifndef SIMULATOR
      SetLCDBacklightLevel(level * 10) ;
#endif
```

12.2 CODE ARCHITECTURE

Main <-> Model <-> Presenter <-> View

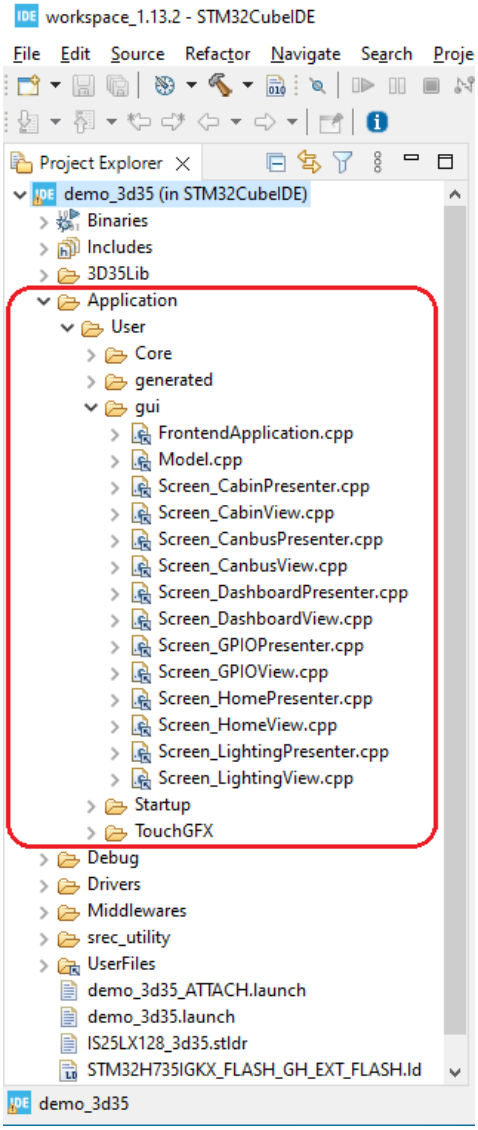
The code that is generated with TouchGFX follows a *Model* <-> *Presenter* <-> *View* architecture.

In the IDE, under Application→User→gui, there are Presenter and View .cpp source files for each Screen.

For example, the Home screen has:

- Screen_HomePresenter.cpp
- Screen_HomeView.cpp

These are automatically generated by TouchGFX. The View files are where most user code will likely be added.



APPENDIX A: STM32CUBE PROGRAMMER SETUP

To use the STM32Cube Programmer tool with the 3D35 display, the external loader file for the Grayhill 3D35 needs to be copied into the appropriate folder. This setup only needs to be done once.

Follow these steps:

1. In the demo_3d35\STM32CubeIDE folder, find file IS25LX128_3d35.stldr and copy it to the ExternalLoader folder for the STM32CubeProgrammer. For a typical installation, this would be:

   C:\Program Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\ExternalLoader

2. Launch the STM32CubeProgrammer.

   On the left side of the window, click the **ExternalLoader** button. In the Search box, start typing the IS25LX128_3d35.stldr filename. The file should pop up in the window.

3. Click the **Select** checkbox.