Quest 5 Material

STMicroelectronics Korea

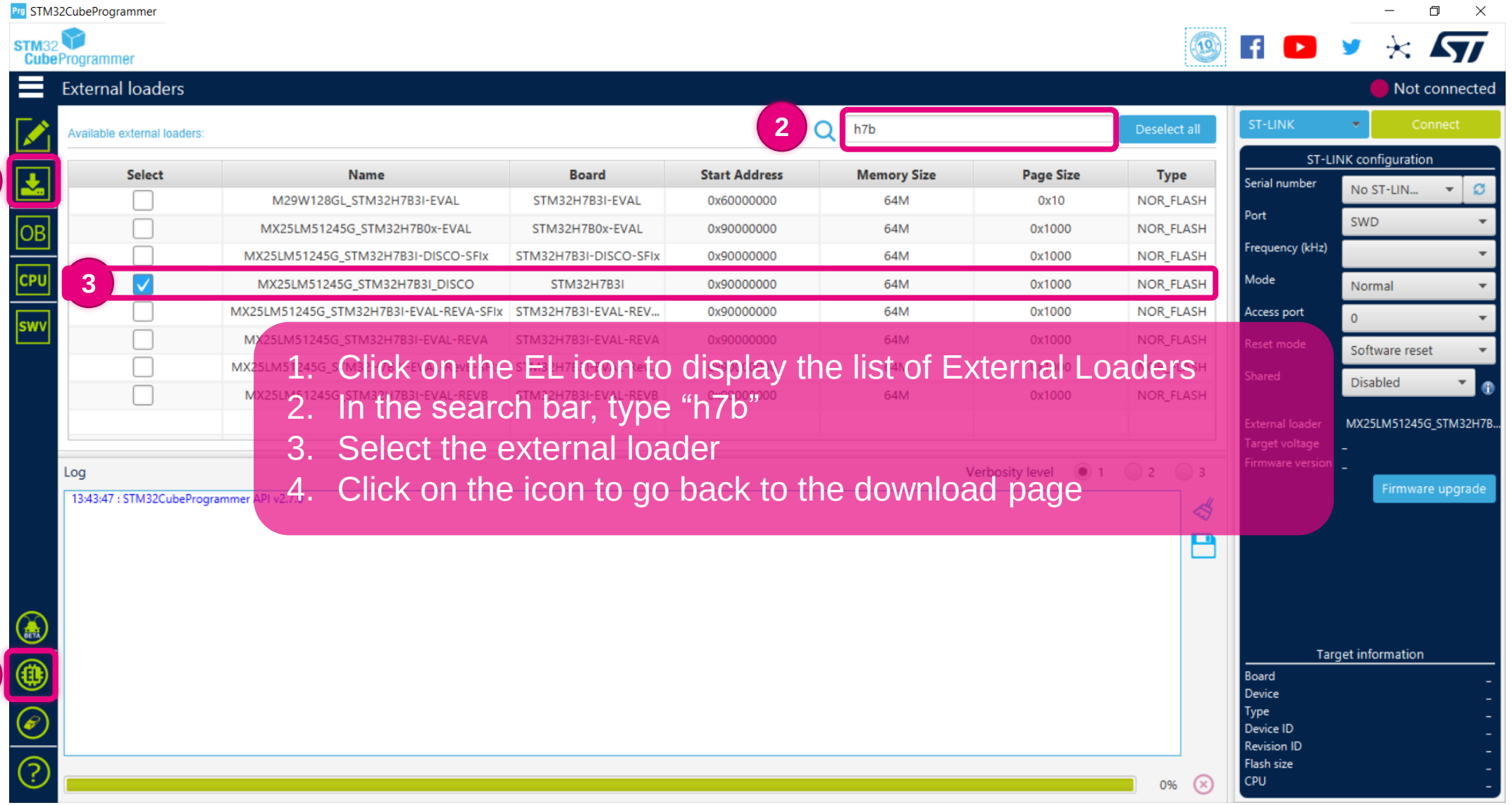
2021

Prerequisite and setup

Software you need to install

- CubeIDE 1.6.1
- CubeMX 6.3.0
- TouchGFX 4.18
- CubeProgrammer 2.8.0
  - In graphic applications, it is common to use external memories to store images, texts, etc. therefore, you cannot download the binary without a dedicated bootloader. In other words, downloading with IAR will not work since IAR will only flash the internal memory. So, you need to use CubeProgrammer. To know how, see next slide.

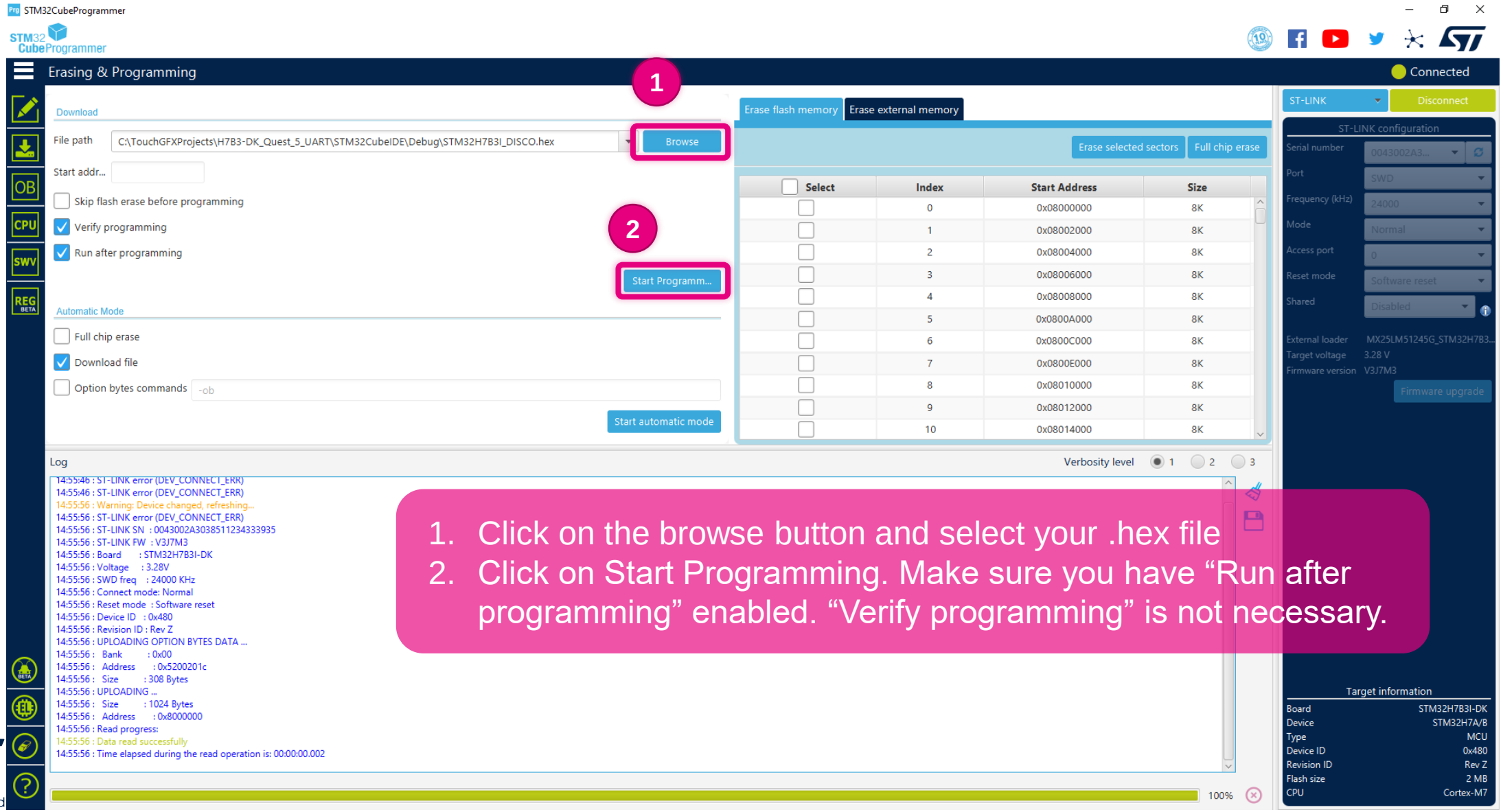
Download binary using CubeProgrammer (1/3)



1. Click on the EL icon to display the list of External Loaders
2. In the search bar, type “h7b”
3. Select the external loader
4. Click on the icon to go back to the download page

life.augmented

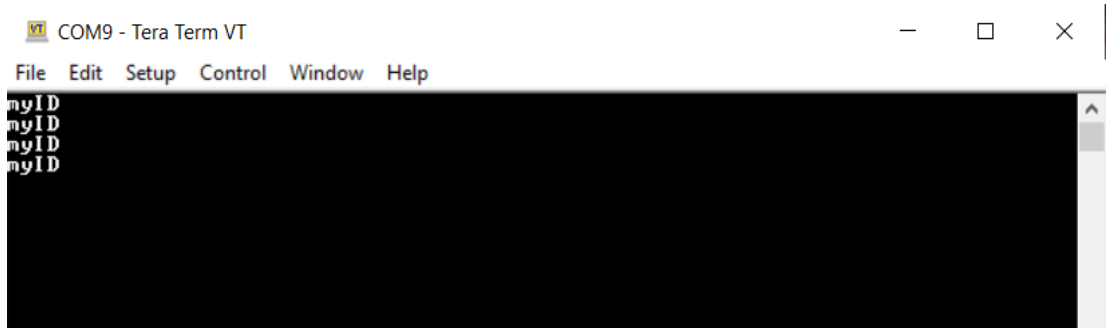
Download binary using CubeProgrammer (2/3)



1. Click on the browse button and select your .hex file
2. Click on Start Programming. Make sure you have “Run after programming” enabled. “Verify programming” is not necessary.

Download binary using CubeProgrammer (3/3)

- Here is what you should see on your board.



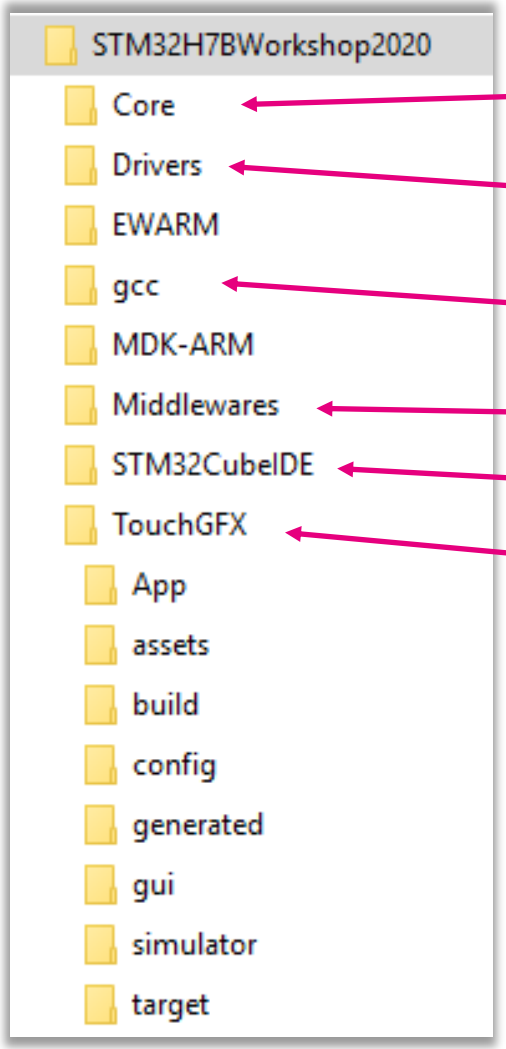
life.augmented

Folder architecture of a TouchGFX project

life.augmented

Explore generated code

In directory : **`<your project folder>\`**



Application common config (main.c, HAL and FreeRTOS conf)

ST MCU Drivers (BSP, HAL)

gcc toolchain files

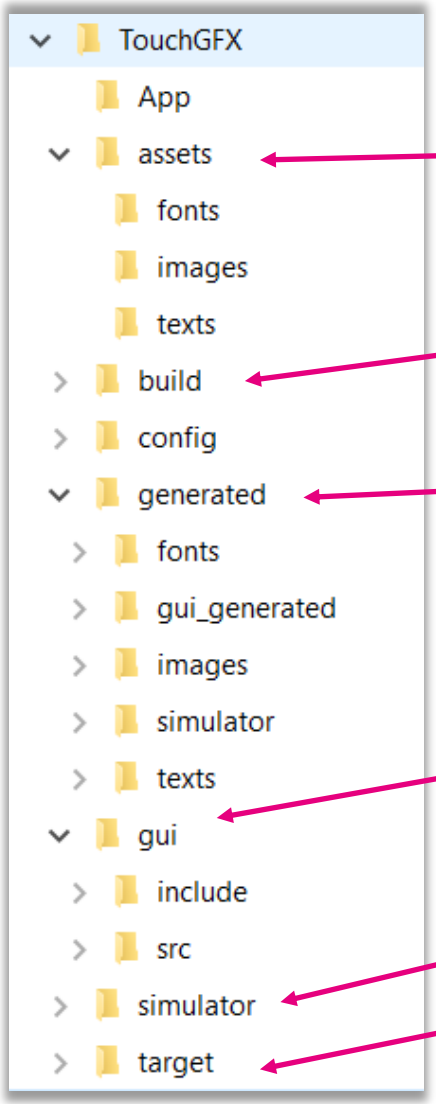
FreeRTOS, TouchGFX framework

STM32CubeIDE configuration files

TouchGFX Designer common files

life.augmented

Explore generated TouchGFX directory



GUI binary inputs (images, texts, fonts)

Working directory for simulator and target compilation

All the cpp files generated from the assets directory + GUI files that should not be modified (in gui_generated)

GUI files that can be modified to implement interactions and customize

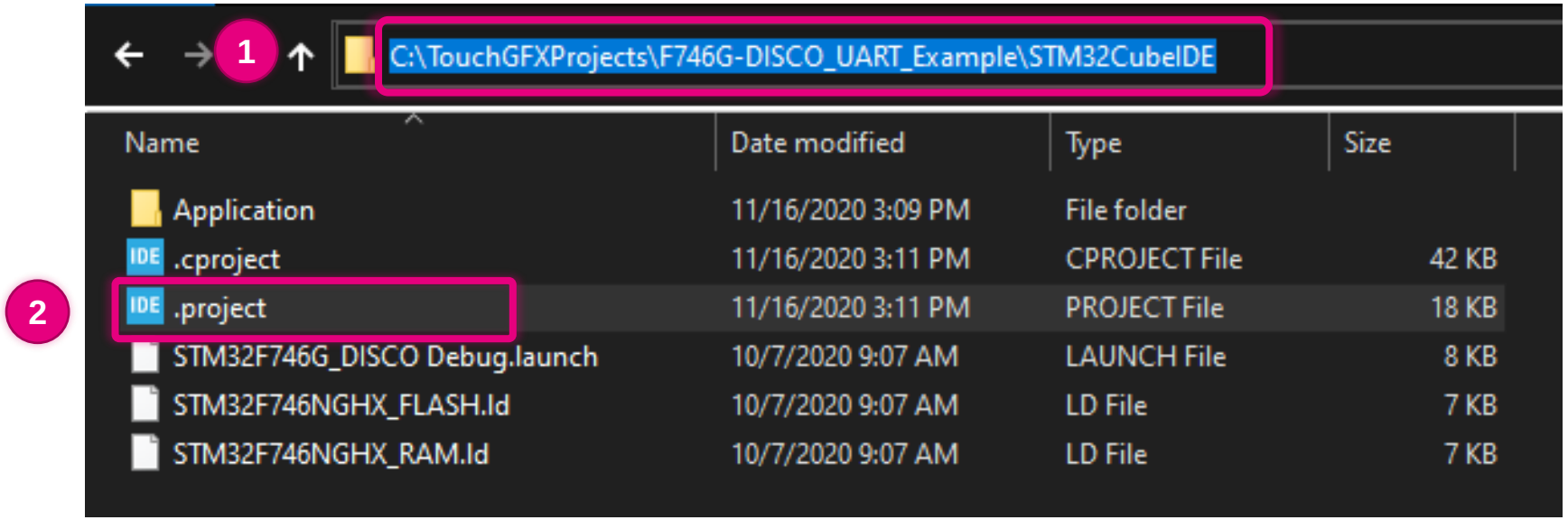
Simulator compilation scripts

Display HW initialisation (LTDC, DMA2D, …)

life.augmented

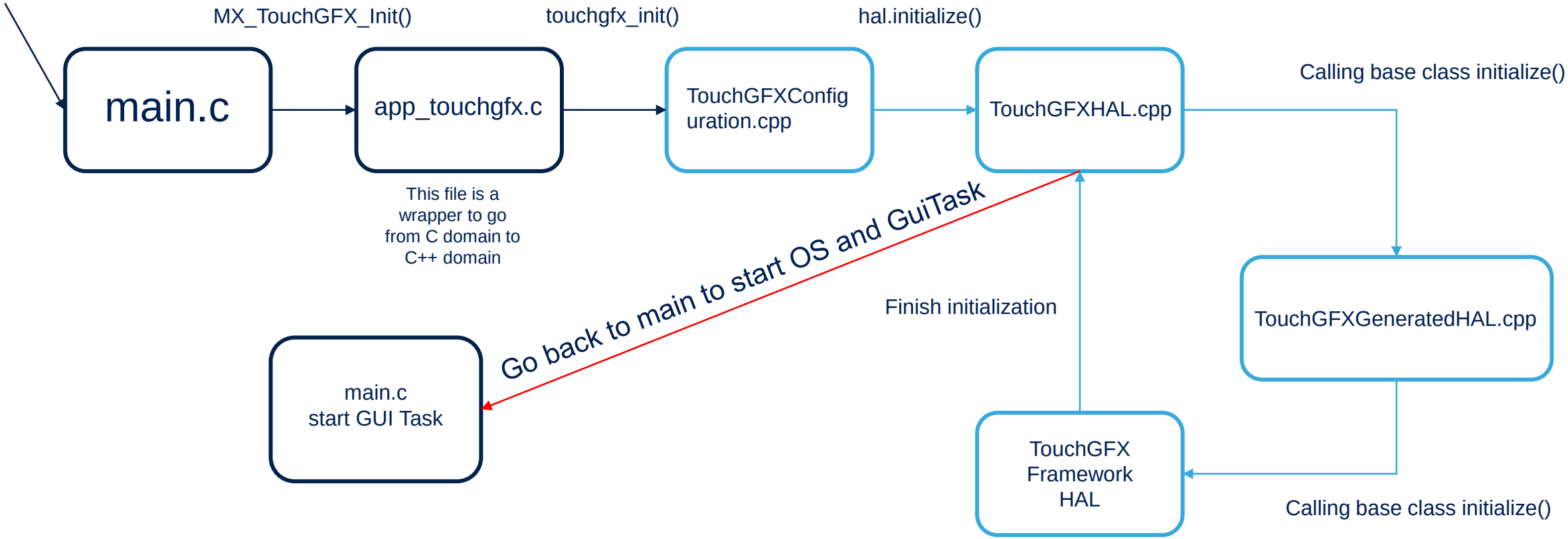
Start STM32CubeIDE

1. Go to **`<your project folder>\STM32CubeIDE\`**
2. Double click on ***.project***, which will bring up STM32CubeIDE



TouchGFX Framework Startup Workflow

TouchGFX Framework Startup Workflow



C++ domain

C domain

life.augmented

TouchGFX Framework Startup Workflow

```
 /* In main.c */
void StartGuiTask(void *argument)
{
  /* USER CODE BEGIN 5 */
  MX_TouchGFX_Process();
  /* Infinite loop */
  for(;;)
  {
    osDelay(1);
  }
  /* USER CODE END 5 */
}
```

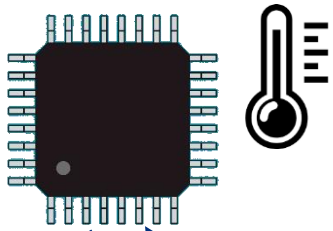
```
/**
 * TouchGFX application entry function (in app_touchgfx.c)
 */
void MX_TouchGFX_Process(void)
{
  // Calling forward to touchgfx_init in C++ domain
  touchgfx_taskEntry();
}
```

```
 /* In TouchGFXConfiguration.cpp */
void touchgfx_taskEntry()
{
 /*
  * 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
  */
  hal.taskEntry();
}
```

life.augmented

MVP system

Model-View-Presenter



MainPresenter.cpp/hpp

Presenter

External event propagation

UI event propagation

External event propagation

Main Application

main.c

C++ to C domain

C to C++ domain

Model

model.cpp/hpp

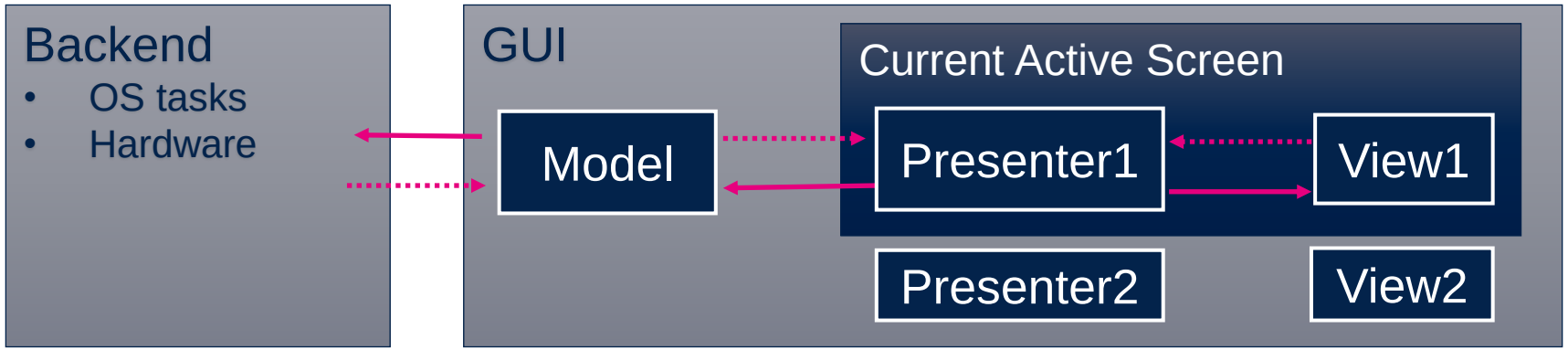
View

MainView.cpp/hpp

Model::tick() is called on each frame rendering (TGFX framework)

TouchGFX Application (1/2)

- Composed of one or several SCREENS
  - Group of widgets (VIEW)
  - Their logic regarding user/system/widgets events (PRESENTER)
  - Only one SCREEN is active at a time (for **lower memory usage**)
- In the MVP architecture:
  - Events flow from Model and View to the presenter
  - Presenter processes events and update accordingly Model and View
  - Model is the only interface with the hardware and backend application

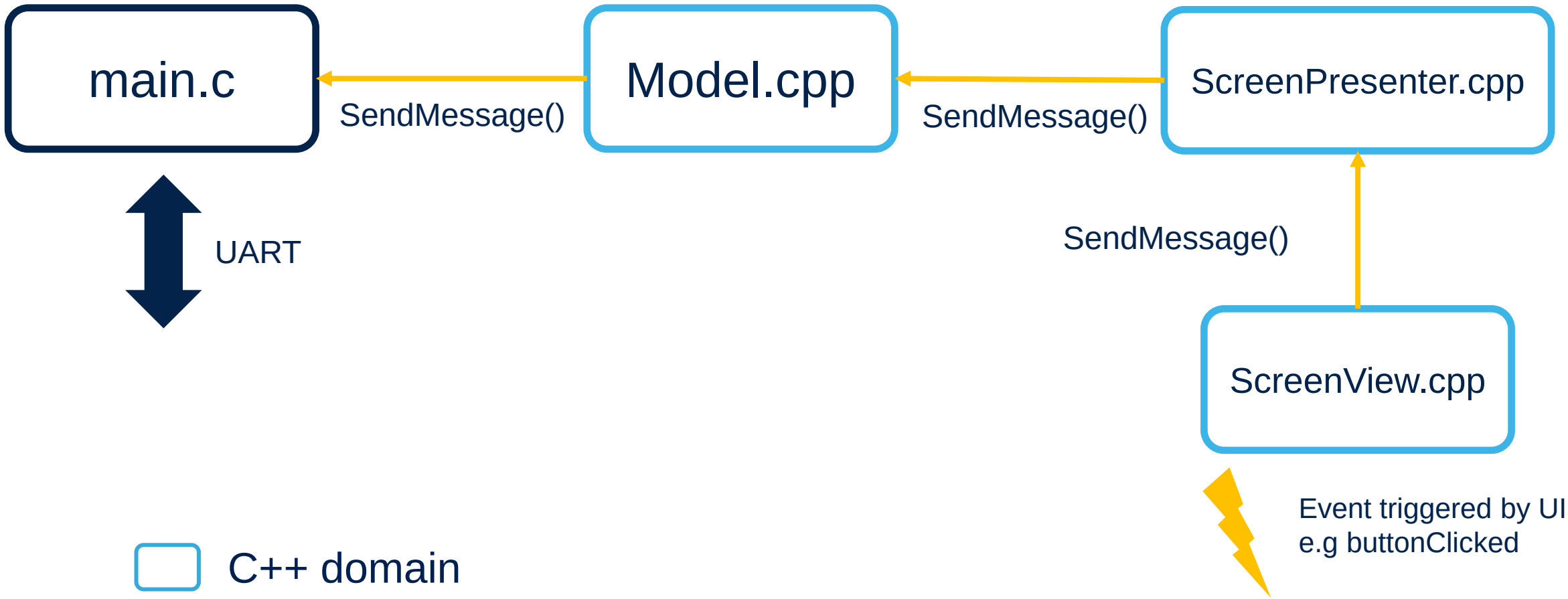


TouchGFX Application (2/2)

- Model
  - Only one in an application
  - Stores state of non-active screens
  - Interface with the hardware, relaying events to and from it
  - Has a pointer to the active presenter
- View
  - Holds and configure the widgets of one screen
  - Has a pointer to it associated presenter to communicate events
- Presenter
  - Receives events from both associated View and Model
  - Decides which action to take

UI to Backend – UART example (no OS)

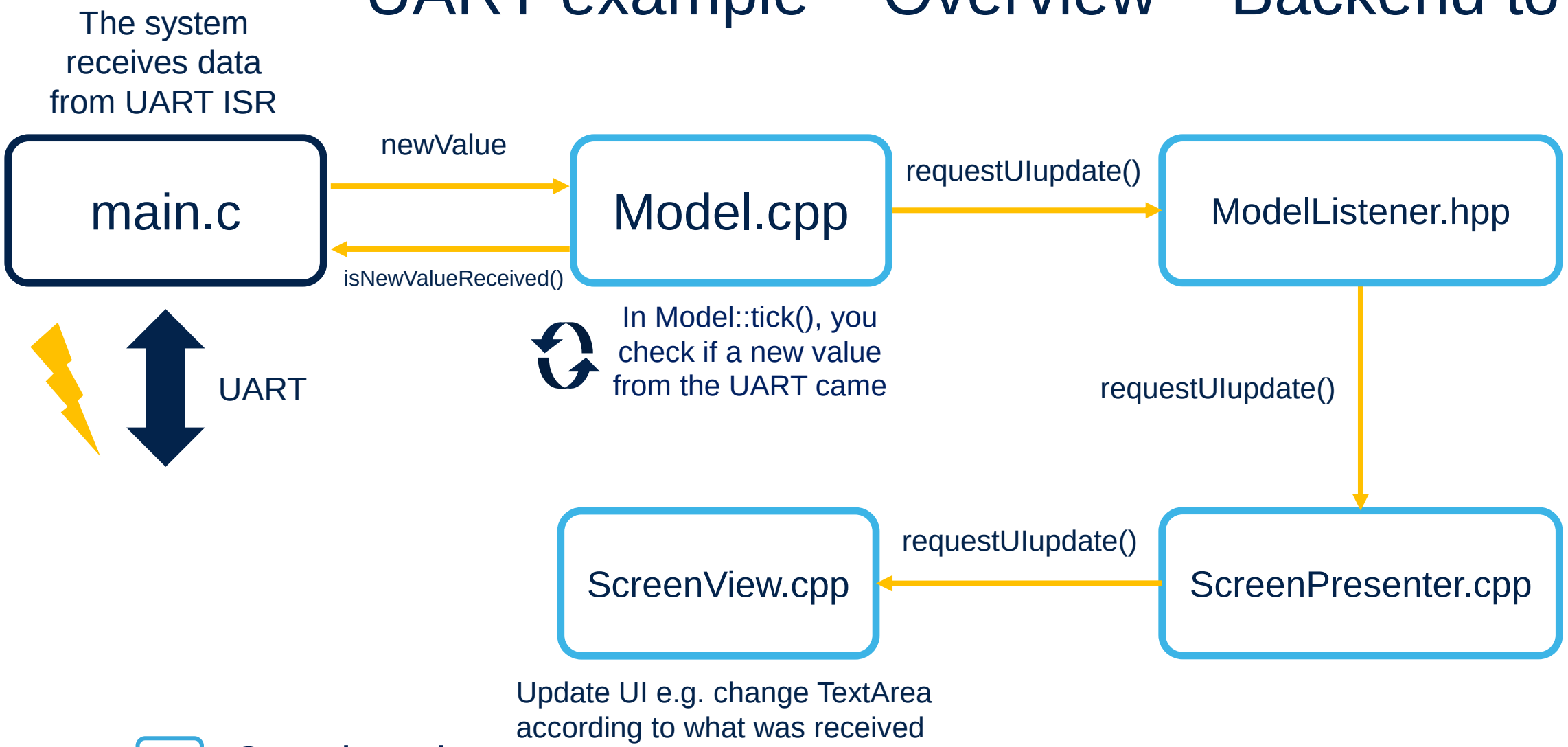
UART example – Overview - UI to Backend



life.augmented

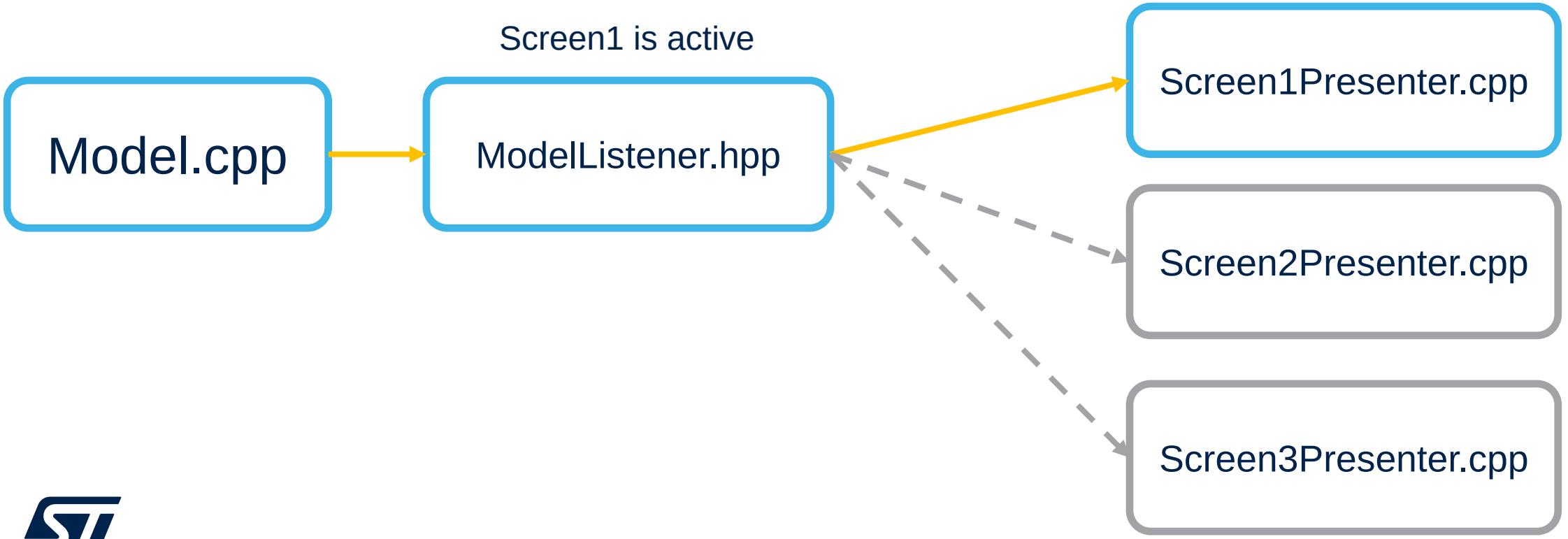
Backend to UI (no OS)

UART example – Overview – Backend to UI



What’s the ModelListener ?

- The Model has a pointer to the currently active Presenter. The type of this pointer is an interface (ModelListener) which you can modify to reflect the application-specific events that are appropriate.

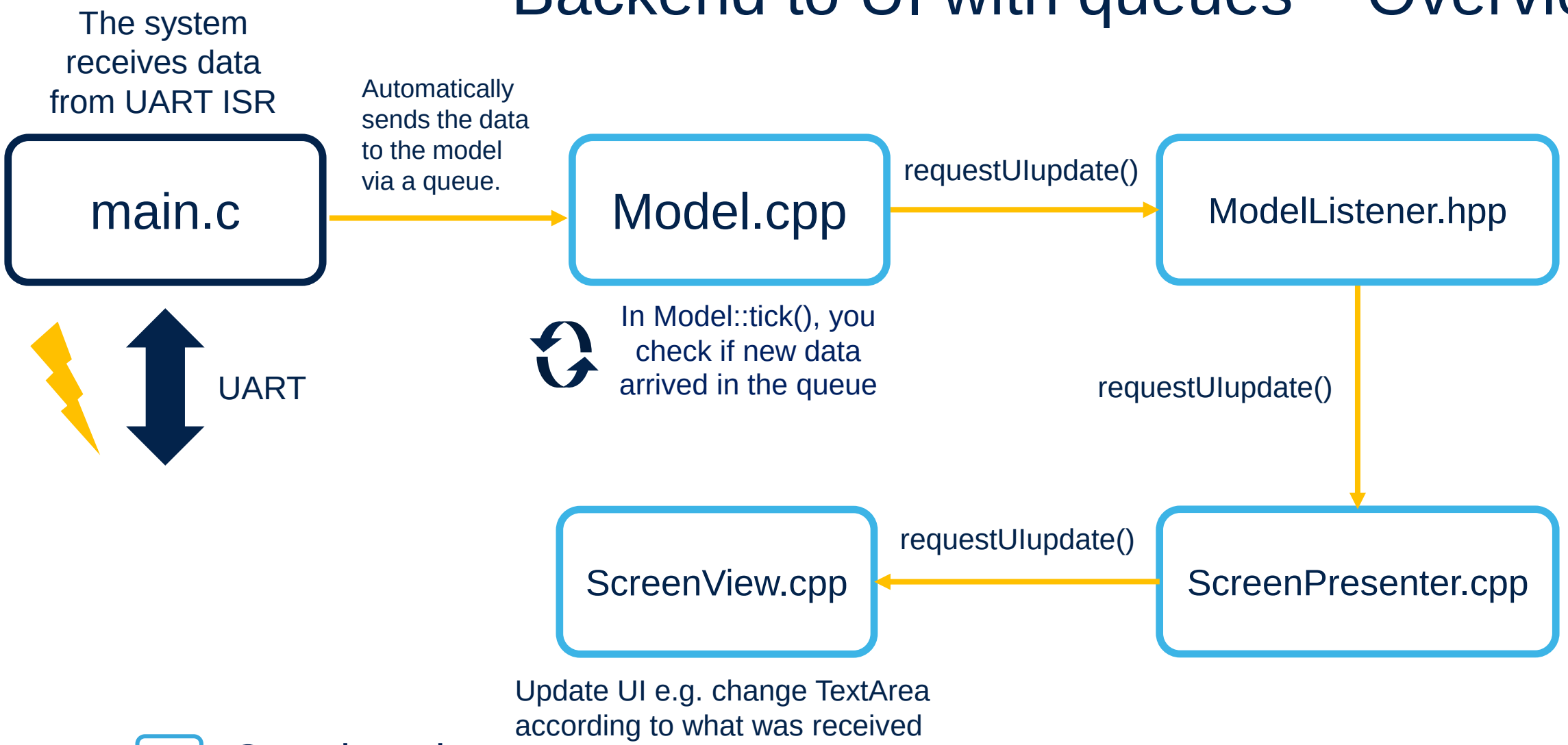


Backend to UI (with OS)

When using FreeRTOS - Queues

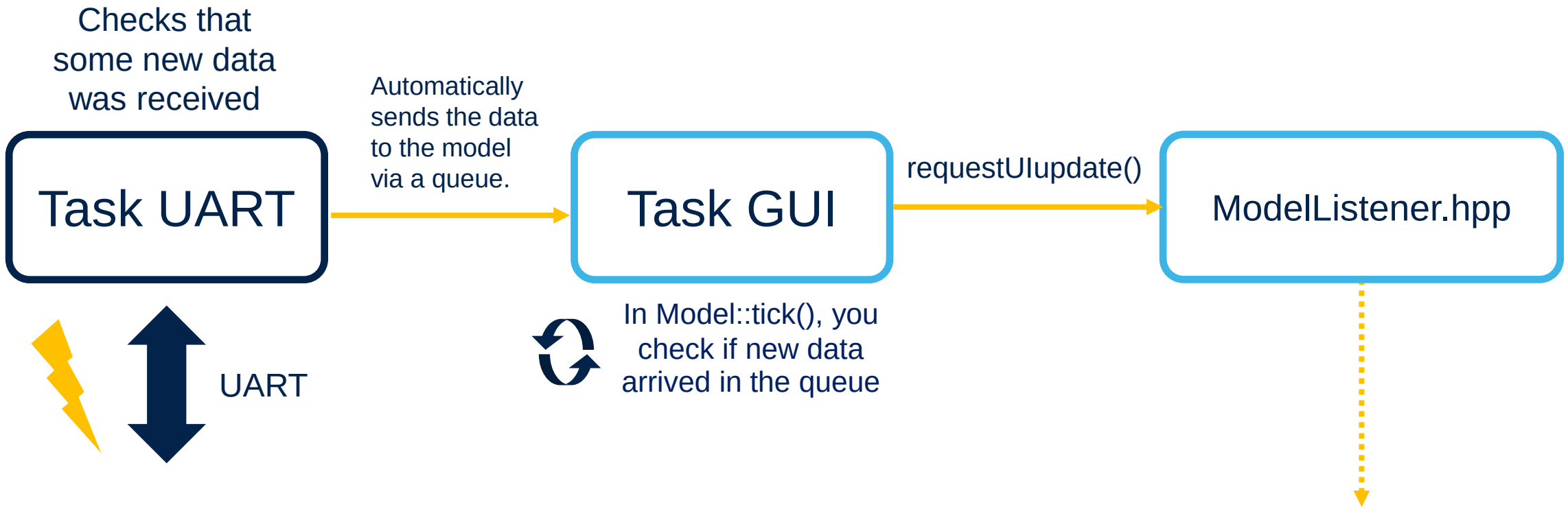
- Prerequisite : None. No need to learn extensively how FreeRTOS works.
- When using FreeRTOS, or any Embedded OS, you most likely use different tasks.
- To send information from one task to the other, you need something called a **queue**.
- Queues have 2 main benefits :
  - Provide a way to communicate between tasks.
  - A non-blocking communication system.

Backend to UI with queues – Overview



life.augmented

Queues for multi-tasks communication



C++ domain

C domain

FreeRTOS Queue API

- For using queues with FreeRTOS you only need to know the following elements.
  - A queue is declared like this :
    - xQueueHandle myQueue;
  - A queue is created as follows :
    - myQueue = xQueueCreate(nbElements, sizeof(element));
  - To add an element in a queue :
    - xQueueSendFromISR(myQueue, &element, 0); // When call inside an interrupt handler
    - xQueueSendToBack(myQueue, &element, 0); //When called from a task
  - To check if an element is in the queue :
    - if (uxQueueMessagesWaiting(myQueue) > 0) { /* Retrieve new data */ }
  - To take the element from the queue :
    - xQueueReceive(myQueue, &newValue, 0); // newValue is the new value received from the queue

Important side notes

Important side notes

- Everything done in the Designer, can also be done in User Code
  - But the Designer can help you with a lot of things
- Avoid going back and forth between User Code and Generated Code
- Remember to utilize the MVP Pattern
- Inspect Generated code
- Reuse code from examples
- The TouchGFX API: Button
- Suggestion :
  - Use the code editor Visual Studio Code.
  - Use paint.NET for image editing

Documentation Link: Model-View-Presenter Design Pattern

Common questions answered

How to go from C to C++ domain and vice-versa ?

```
/**
 * Declaration of a C function in a C++ file
 */
extern “C”
{
   void myFunctionInCDomain();
}
```

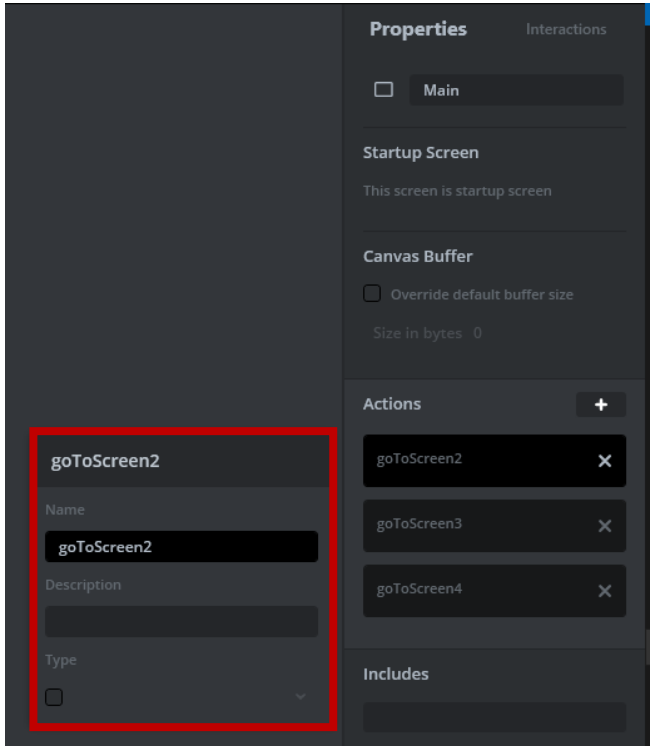
```
/* The function previously declared
 * can be called anywhere in the file
 */
void Model::func1()
{
    myFunctionInCDomain();
}
```

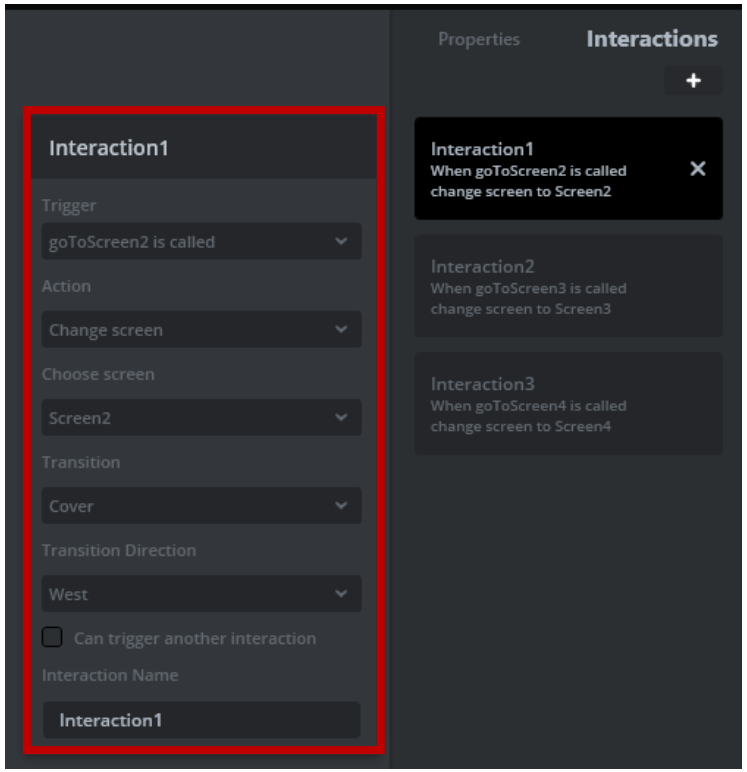
```
/**
 * Implement the function in your C file
 */
void myFunctionInCDomain()
{
   // Code executed
}
```

life.augmented

How to add a screen change upon external event ?

- Steps :
  - You create a change screen action (you will call it from user code later on)
  - Then you create your interaction that performs a screen transition where the trigger is the call of your previously created action





Thank you

life.augmented