ELLIE software achitecture

Wouter Pennings – April 2024

Table of Contents

Introduction

A student initiative, "The Contour Wall," is part of the "ELLIE" project, which aims to boost engagement within Fontys buildings for students, employees, and visitors. This large-scale, colorful display serves as a focal point showcasing student talent and fostering collaboration while providing hands-on learning through integrating diverse ICT skills. Its open interface allows future students to easily incorporate The Contour Wall into their projects.

This software design document is intended for developers interested in contributing to the Contour Wall. It details the system architecture, from the tile firmware to the control computer software layers. A C4 model provides a high-level view of the system components and their interactions. We'll also explore the custom communication protocol between the computer and the tiles.

For a deeper understanding of the hardware and related terminology, it's recommended to review the Contour Wall design document first. Said document primarily goes over the hardware and construction of the Contour Wall and covers many setups and terms that will be useful to understand the software architecture.

For detailed function or class-level documentation, please refer to the dedicated documentation found within the Github repository. All libraries and wrappers are comprehensively documented with examples for your reference.

GitHub: https://github.com/StrijpT-Ellie/contour-wall

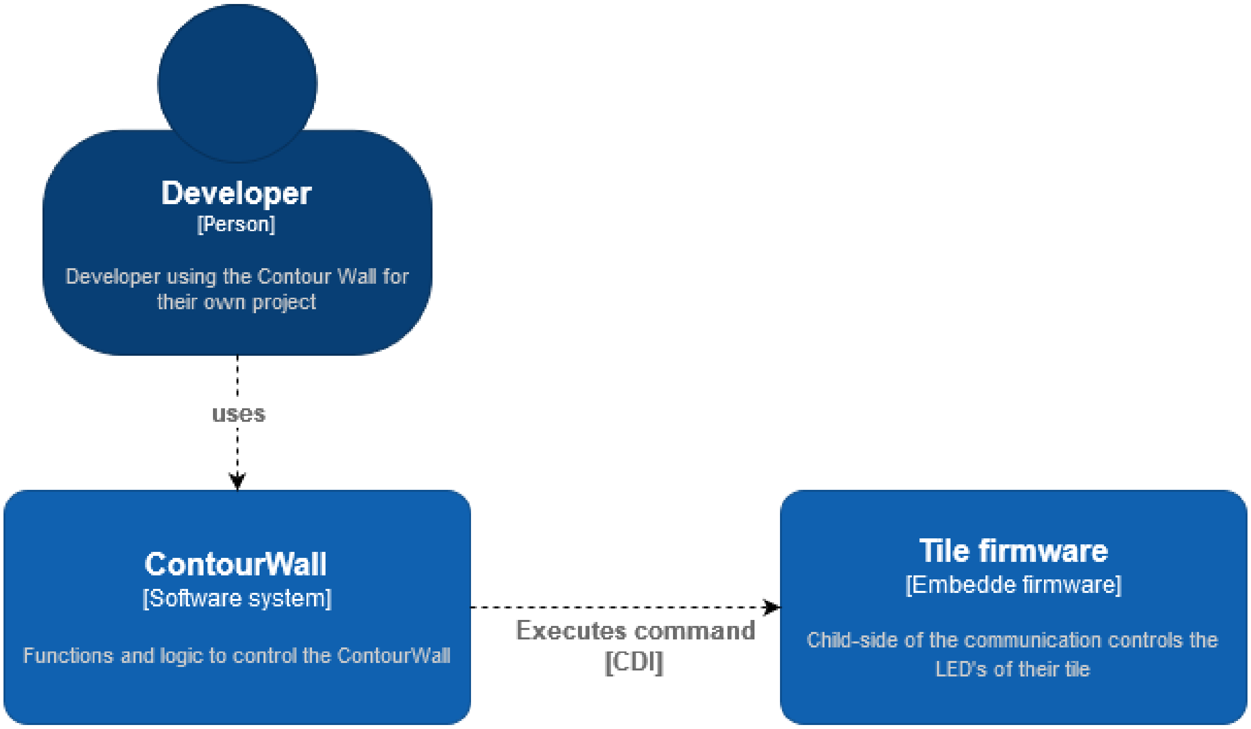
Terminology

Term	**meaning**
Parent	The library which controls the Contour Wall, runs on the parent device, often a laptop or Raspberry Pi. Previously often called “master”.
Child	The child is the tile on which a micro controller (ESP32-S3) is mounted. Previously often called “slave”.
Concurrency	Multiple computations are being executed in parallel.
AOT compiler	Ahead-of-time compilation
JIT compiler	Just-in-time compilation is compilation during execution of a program rather than before execution
ABI	An ABI defines how data structures or computational routines are accessed in machine code, which is a low-level, hardware-dependent format.
Magic number	A list of numbers used by E.G. files so other programs way to recognize them other than a file extension.

C4 Model

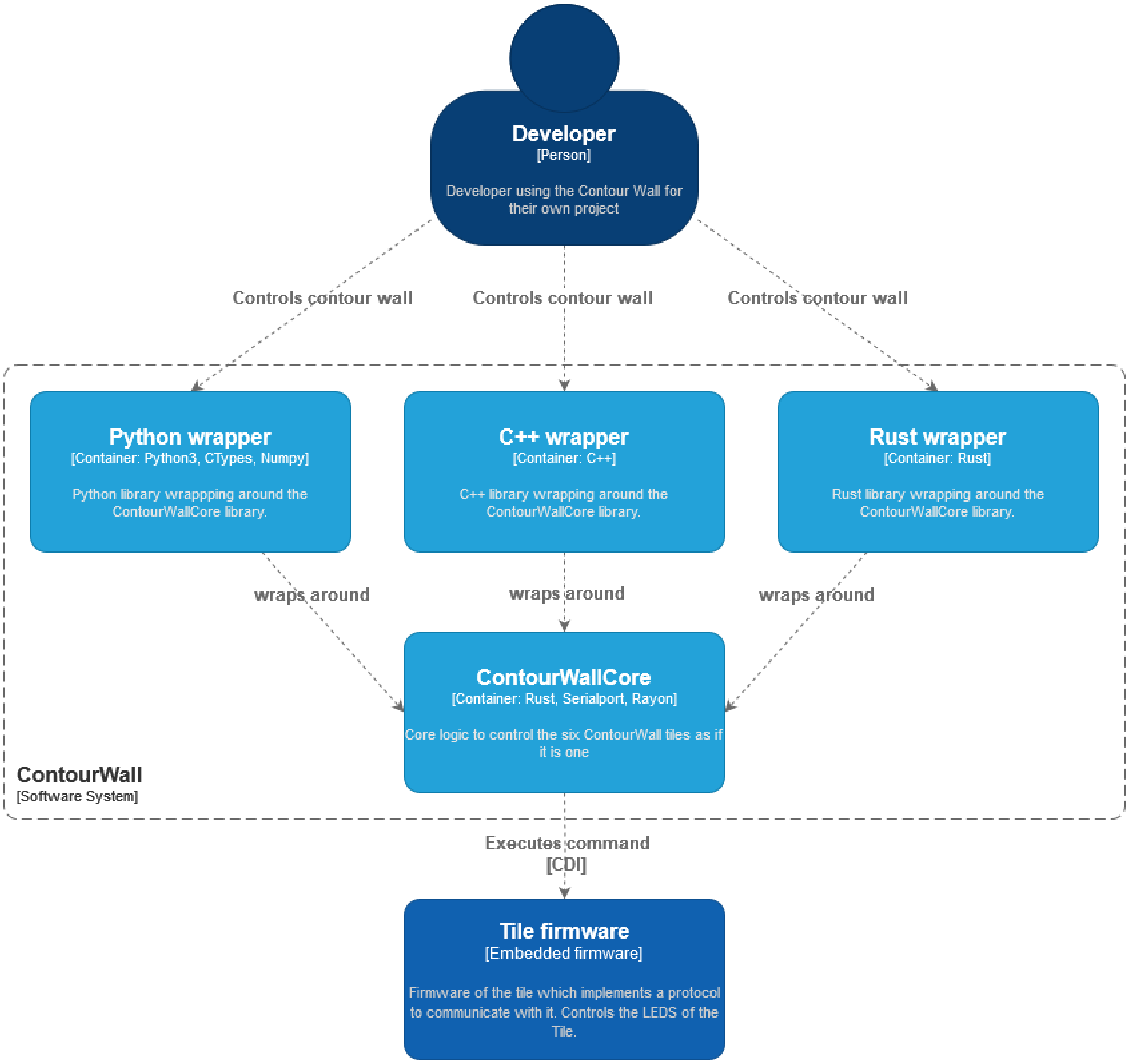
Level 1

The “ContourWall” system runs on the parent device, and the tile firmware runs on the child.

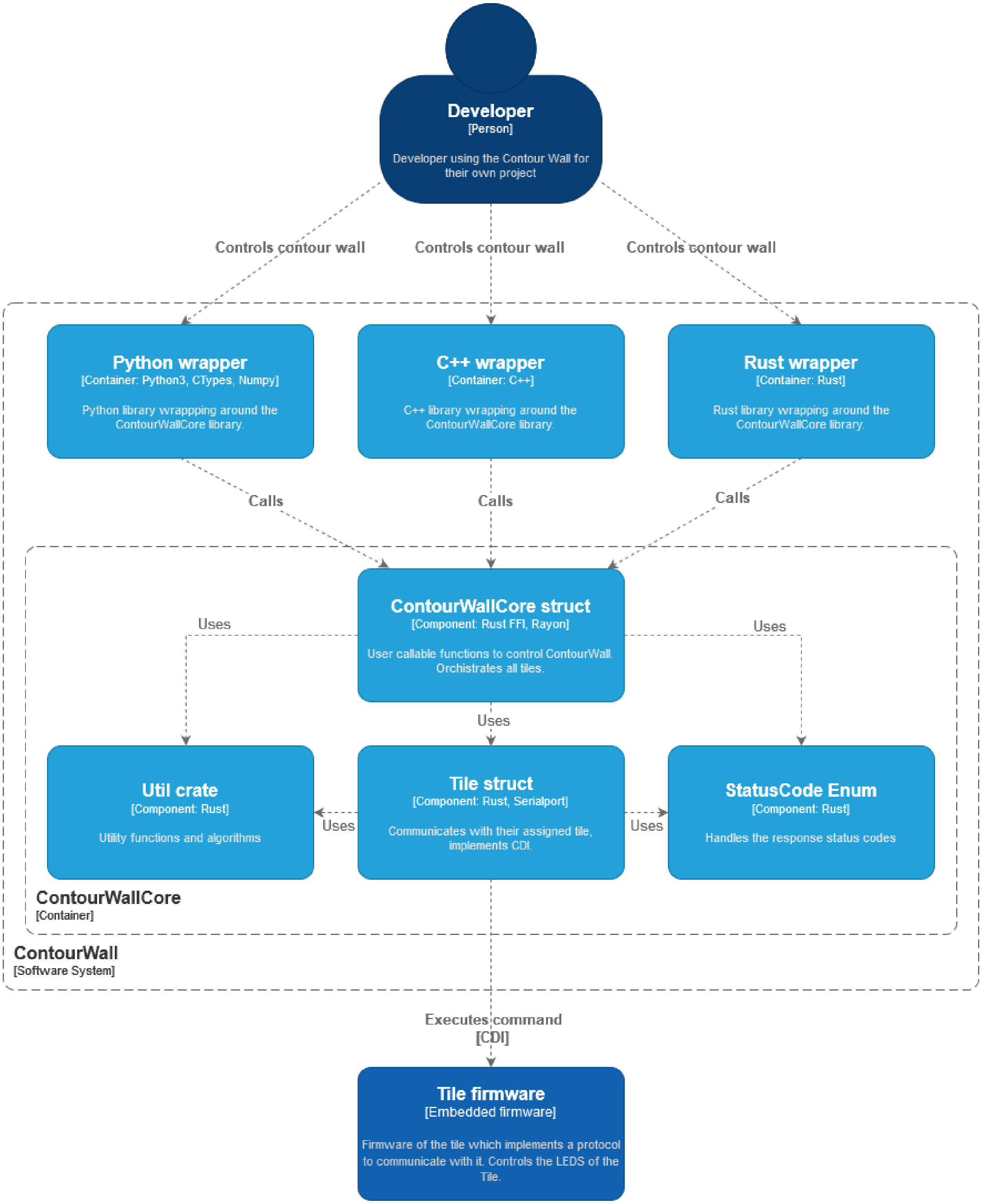

Developer
[Person]
Developer using the Contour Wall for their own project
uses
ContourWall
[Software system]
Functions and logic to control the ContourWall
Executes command
[CDI]
Tile firmware
[Embedde firmware]
Child-side of the communication controls the LED's of their tile

Level 2

The level 2 diagram shows how the ContourWall system contains of 2 layers. The first layer are the wrappers, the developer chooses the wrapped based on the language they want to use or, are already using. However, all these wrappers use the same library in the background to control the Contour Wall; the “ContourWallCore” library.


Developer
[Person]
Developer using the Contour Wall for their own project
Controls contour wall
Controls contour wall
Controls contour wall
Python wrapper
[Container: Python3, CTypes, Numpy]
Python library wrappping around the ContourWallCore library.
C++ wrapper
[Container: C++]
C++ library wrapping around the ContourWallCore library.
Rust wrapper
[Container: Rust]
Rust library wrapping around the ContourWallCore library.
wraps around
wraps around
wraps around
ContourWallCore
[Container: Rust, Serialport, Rayon]
Core logic to control the six ContourWall tiles as if it is one
ContourWall
[Software System]
Executes command
[CDI]
Tile firmware
[Embedded firmware]
Firmware of the tile which implements a protocol to communicate with it. Controls the LEDS of the Tile.

Level 3


Developer
[Person]
Developer using the Contour Wall for their own project
Controls contour wall
Controls contour wall
Controls contour wall
Python wrapper
[Container: Python3, CTypes, Numpy]
Python library wrappping around the ContourWallCore library.
C++ wrapper
[Container: C++]
C++ library wrapping around the ContourWallCore library.
Rust wrapper
[Container: Rust]
Rust library wrapping around the ContourWallCore library.
Calls
Calls
Calls
ContourWallCore struct
[Component: Rust FFI, Rayon]
User callable functions to control ContourWall. Orchistrates all tiles.
Uses
Uses
Uses
Util crate
[Component: Rust]
Utility functions and algorithms
Uses
Tile struct
[Component: Rust, Serialport]
Communicates with their assigned tile, implements CDI.
Uses
StatusCode Enum
[Component: Rust]
Handles the response status codes
ContourWallCore
[Container]
ContourWall
[Software System]
Executes command
[CDI]
Tile firmware
[Embedded firmware]
Firmware of the tile which implements a protocol to communicate with it. Controls the LEDS of the Tile.

Level 4

STILL HAVE TO MAKE THIS ONE

Communications protocol

For the communications between the parent and child a custom communications protocol is made, it is built upon UART. This protocol is called: "**C**ontourwall **D**isplay **I**nterface" or **CDI** for short. The protocol is based on the execution of commands on the child, by the parent. The child can send a response back to the parent, but it is impossible for the child to execute a command on the parent.

Like mentioned before, in the case of the Contour Wall, a parent is the ContourWall library, often run on E.G. a laptop or Raspberry Pi. The children are the tiles of which the Contour Wall is comprised. The tiles have an ESP32-S3 as a controller.

CDI has a default UART communications speed of 2MHz. Lower speeds should work without a problem and will increase stability but with lower framerate, important to know that a speed of 2MHz has never caused any stability problems. A higher communication speed has not been tested yet, but should work, at worse stability depending on the speed. UART has been configurated in an 8N1 configuration.

- A data frame has size of 8 bits.
- No parity bit. This is unnecessary as a custom CRC will be added, which is more efficient than having each data frame have its own parity bit.
- One stop bit instead of the optional two. A second stop bit could be added if there is signal noise.

This results in a protocol efficiency of 80%, meaning that if I want to send 1200 bytes, or 9600 bits, the actual number of bits transferred would be 12000.

Commands

To execute a command, the parent sends the command code to the child. If a command body is expected the child will wait. Otherwise, the child continues, does its thing and returns its response after it is done.

In total there are seven different commands, that a parent can execute on the child:

Command	**Description**	**Command body**	**Response**
0: show	Signals update their pixels based on their local framebuffer	**0 bytes**	**0 bytes**
1: solid color	This sets all the pixels of the local framebuffer of the tile to these colors	**4 bytes**, one byte for red, green, blue and a CRC.	**1 byte**: Status code
2: update all	Send a RGB values for each LED of the 400 LEDs of the tile.	**1201 bytes:** 400 RGB values + a CRC. Each RGB is 3 bytes. One RGB value per pixel.	**1 byte**: Status code
3: update specific	Updates specific LEDs on the tile. This command should be used in the cases where only a couple of pixels need to be changed. This has worse data efficiency than "update all" but is allows for specific pixels changes if needed.	**n bytes:** The first is number of pixels which are going to be updated. Then there are 5 (2 bytes for pixel index[1], 3 bytes for RGB code) bytes for each pixel and an CRC. Example: if 8 pixels will be updated then there are 8*5+1 = 41 bytes.	**1 byte**: Status code
4: get tile identifier	Requests the unique identifier stored in the EEPROM. This identifier is used to know what COM port which tile is.	**0 bytes**	**3 bytes**: identifier, CRC, Status code
5: set tile identifier	Sets the unique identifier of an ELLIE tile.	**2 bytes**: identifier and a CRC	**1 Byte**: Status code
6: Magic numbers	Asks for the magic numbers of the tile. This is to identify that the COM port is a tile for the Contour Wall	N/A	**5 bytes:** which are the ASCII values of "Ellie"

[1] There are 400 pixels, one byte has a maximum value of 255, therefore if you want to update pixel on pixel E.G. 311, you need to split the index into two bytes.

Status code

To communicate the status to the parent after the child has executed a command a status code is returned. These are like the HTTP status codes, but customized for this context:

Status code	Value	Description
Error	0	Generic error, something went wrong on the child-side
Too slow	1	The data on the child side has not been received with 10ms
Non-matching CRC	2	The CRC does not match the actual data
Unknown command	3	This command identifier is not known
Ok	100	Everything went okay, no problems
Next	101	Data is received, and child is ready for the next chunk
Reset	255	Something went wrong, do not send anything for 100ms. Child is clearing UART buffers, parent should clear UART buffers too.

Any status code value that is received, but that is not defined will be ignored.

Software

Contour Wall System

The ContourWall system runs on the parent. It consists of two parts; the first part is the “ContourWallCore” library of implemented in Rust. This contains all the algorithmic and communication logic of the system. The core library should never be used directly by developers incorporating the ContourWall into their project.

The implementation of ContourWallCore follows the C ABI, which makes its functions callable from other programming languages when compiled to a library; that is what the second part of the ContourWall system is for. The second part is a wrapper around ContourWallCore library, the wrapper abstracts away the difficulties of ContourWallCore to increase developer productivity while keeping the same performance.

ContourWallCore library

The main requirements for the implementation language of ContourWallCore are threefold:

1. **Performance.** This ruled out all non-JIT or AOT compiled languages like Python and Lua.
2. **Native language without a runtime; coequally a systems language**. This immediately ruled out lots of languages: C#, Go, Kotlin, JavaScript, etc.
3. **A safe language, mainly from a memory perspective.** This ruled out C, C++. Memory safety is very important in this case, as this an application that could run for weeks. Any minor memory leak would compound for a long time before surfacing as a problem, which would stay under the radar while testing.

This left Rust as the most popular option who fitted the requirements. Some additional upsides of Rust are its excellent documentation, first-party development tools, package manager (Cargo) and vibrant community. We are aware that Rust is a relatively young language with a tiny number of students having noteworthy experience. However, it is growing rapidly and trusted by influential organization and companies such as Linux, Microsoft, Amazon and Google.

ContourWallCore has two major dependencies, Serialport for serial communication and Rayon for concurrency. Both Serialport and Rayon are major libraries that are the de facto choices in their field; Rayon is even used in the Rust compiler itself.

The Contour Wall consists of six tiles to which data needs to be send to, and that data also need preprocessing before it is ready to be send to the tiles. This processing and communicating with the tiles are 6 separate processes which can be parallelized to increase performance. Using Rayon, 6 threads are spawned, each thread will execute the command with their assigned tile. The concurrent solution, compared to the sequential solution is about 5 times faster, some execution time is lost due to the extra overhead. There are two improvements because of the faster parallel execution:

1. Users can push five times more frames to the Contour Wall, *or* the user has five times more time to generate content per frame.
2. The synchronization of the six panels is better, because the time between when the first and last tile will be updated is shorter.

MAYBE MAKE FLOWCHART TO EXPLAIN BETTER HOW THE CONCURRENCY MODEL WORKS????

ContourWallCore wrappers

Officially there are three supported languages where a wrapper is implemented: Python, C++ and Rust. These three languages should cover most uses-cases that will be come across. However, third parties can create their own wrapper in the language of their choice when the need arises. The official wrappers are written with these rules or desires in mind:

1. **Contain as little logic as possible.** Especially if all the wrappers have the same logic, it should be considered to move this logic into ContourWallCore.
2. **Follow existing conventions of packages often used in combination with the wrapper.** This reduces the complexity of incorporating the Contour Wall into existing projects. E.G. OpenCV will be used to display content on the ContourWall, make sure that the wrapper uses OpenCV conventions. This makes sure that little conversions between data structures is needed to improve performance and reduce complexity.
3. **Keep the API of the wrappers similar.** When the API of all wrappers is identical is it easy to switch between them, streamlines wrapper development and reduces the learning curve.

It is advised that future (third-party) wrappers abide by these rules.

Tile firmware

The firmware of the tile is implemented in C++ (Arduino). FastLED is the sole dependency, it is required to control the WS2812B LEDs on the display panels.