OpenLcbCLib

Quick Start Guide

Get an OpenLCB/LCC node running in minutes

ESP32 + Arduino IDE edition

OpenLcbCLib supports many platforms — this guide covers one specific path.

Table of Contents

1. What is OpenLCB/LCC?

Note: This guide uses an ESP32 development board with Arduino IDE as a concrete, working example. OpenLcbCLib itself is platform-agnostic C — it can be adapted to any microcontroller with a C compiler. See the *Developer Guide* for other platforms and IDEs.

OpenLCB (Open Layout Control Bus) is an open-source project that produced a suite of specifications and technical notes for connecting model railroad devices. The NMRA adopted these specifications and technical notes as the LCC (Layout Command Control) standard. LCC and OpenLCB refer to the same protocol — LCC is simply the NMRA's official name for it.

Throttles, signal controllers, turnout drivers, occupancy detectors, and other accessories all talk to each other over a shared CAN bus (or WiFi) using a common language. OpenLcbCLib is a portable C library that implements these protocols for microcontrollers, handling all low-level protocol work so you can focus on what your node actually does.

Key Terms

- **Node:** Any device on the network. Your ESP32 board is a node.
- **Node ID:** A unique 48-bit number that identifies your node globally.
- **Event ID:** A 64-bit number representing something that happened or a command to perform.
- **Producer:** A node that sends an event when something happens.
- **Consumer:** A node that reacts to an event.
- **CDI:** Configuration Description Information — an XML file describing what settings your node exposes.
- **CAN Bus:** The physical wiring (two wires) that connects all nodes on the layout.

2. What You Need

Hardware

- ESP32 development board (any common 38-pin or 30-pin board)
- CAN bus transceiver module (SN65HVD230 or MCP2551 recommended)
- USB cable for programming
- LCC/OpenLCB layout bus, OR two ESP32 boards to test with

Note: The ESP32 has a built-in CAN controller called TWAI (Two-Wire Automotive Interface). You only need an external transceiver chip — you do NOT need a separate CAN controller.

CAN Transceiver Wiring

ESP32 Pin	Transceiver Pin
GPIO 21	TX (CAN transmit)
GPIO 22	RX (CAN receive)
3.3V or 5V	VCC (check your module datasheet)
GND	GND
(layout bus)	CANH and CANL to bus wires

Software

- Arduino IDE 2.x or PlatformIO (VSCode)
- ESP32 Arduino core installed in Arduino IDE (search "esp32" in Boards Manager)
- A modern web browser (Chrome, Firefox, or Edge) for the Node Wizard
- The OpenLcbCLib source code

---

3. Project Setup for Arduino IDE

The library is NOT installed as an Arduino Library. Instead, the source files live directly inside your sketch folder. This keeps every project self-contained with its own copy of the library.

Important Arduino IDE rule: Arduino IDE will only automatically compile C/C++ files that are in the sketch folder itself *or* in a subfolder named exactly **src**. Files in any other subfolder are ignored. The library is therefore structured so all source files sit under a folder named src/.

Starting From the ESP32 Example

The library includes a ready-to-run ESP32 example at:

```
OpenLcbCLib/src/applications/arduino/esp32/BasicNode/
```

1. Copy the entire BasicNode folder to your Arduino sketches folder.
2. Rename the folder to your project name, then rename the .ino file inside it to the same name. Arduino IDE requires the folder and .ino file to have exactly the same name.
3. Open the renamed .ino file in Arduino IDE.

Complete Project File Structure

After copying, your sketch folder will look like this. The src/ subfolder is what Arduino IDE compiles as library code.

```
BasicNode/                          <- your sketch folder
|-- BasicNode.ino                   <- main sketch
|-- openlcb_user_config.h           <- buffer sizes & feature flags
|-- openlcb_user_config.c           <- node parameter data
`-- src/                            <- Arduino IDE compiles this subtree
    |-- application_callbacks/      <- your application logic
    |   |-- callbacks_can.h / .cpp
    |   |-- callbacks_config_mem.h / .cpp
    |   `-- callbacks_olcb.h / .cpp
    |-- application_drivers/        <- hardware interface
    |   |-- esp32_can_drivers.h / .cpp
    |   `-- esp32_drivers.h / .cpp
    `-- openlcb_c_lib/              <- library (do not edit)
        |-- openlcb/                <- protocol engine
        |   `-- ... (library files)
        |-- drivers/canbus/         <- CAN state machines
        |   `-- ... (driver files)
        `-- utilities/
            `-- ... (helper files)
```

Note: Never rename the src/ subfolder. Arduino IDE will stop compiling the library files if the folder has any other name.

4. Choosing a Node ID

Every node on a layout must have a unique 48-bit Node ID — think of it like a MAC address. If two nodes share an ID the network will have conflicts.

The NMRA does not maintain individual Node IDs. Instead, it assigns **ranges** to manufacturers and members. Each range owner then assigns individual IDs from their own range. The full list of allocated ranges is published at registry.openlcb.org/uniqueidranges.

Option A: NMRA Member Range

If you are an NMRA member, you automatically have a personal Node ID range derived from your member number. Log in to the NMRA member portal to look up your assigned range. This guarantees globally unique IDs for any nodes you build and share with others.

Option B: Request a Manufacturer Range

If you are producing nodes commercially or distributing firmware publicly, request a dedicated range through the OpenLCB group. See the registry at registry.openlcb.org/uniqueidranges for details.

Option C: Development / Test Range

For personal testing on your own layout, you can use Node IDs from the reserved development range. These are safe to use privately but should not be used in nodes you distribute to others.

Setting the Node ID

BasicNode demo — the Node ID is defined as a constant at the top of the sketch. Find this line and change it to your chosen value:

```
// In BasicNode.ino, find and change this line:
#define NODE_ID  0x050101010777

// Use any value in the development range:
#define NODE_ID  0x050101010001   // node 1
#define NODE_ID  0x050101010002   // node 2
// Keep a list - every node needs a different ID
```

Wizard-generated projects — enter your Node ID in the **Project Options** panel before you generate the code. The Wizard accepts the ID in dotted hex format (**xx.xx.xx.xx.xx.xx**, for example 05.01.01.00.00.01) and writes it into the generated files automatically. There is no `#define NODE_ID` to edit by hand.

> **Note:** Every node on your layout must have a DIFFERENT Node ID, even when using the development range. Keep a list so you do not accidentally reuse one.

---

5. Starting a Project From Scratch

Once you are comfortable with the BasicNode demo, you will want to start your own project. Open the browser-based **Node Wizard** at **tools/node_wizard/node_wizard.html** in any modern browser — no internet connection required. Work through the sidebar from top to bottom:

Sidebar Section	What You Do There
Node Type	Pick the role of your node: Basic (events only), Typical (events + configurable settings), Train (locomotive decoder), or Train Controller (throttle).
CDI	Describe the settings your node exposes to JMRI and other tools. Required for Typical, Train, and Train Controller nodes.
FDI	Describe the DCC functions your decoder supports. Train nodes only.
Platform Drivers	Select your target hardware. The Wizard pre-fills working driver code for the chosen platform and sets Arduino mode automatically.
Callbacks	Choose which protocol events your application needs to respond to.
Generated Files	Preview the output file tree and click Download ZIP to get your project.

The **Platform Drivers** step controls whether the output uses Arduino or non-Arduino layout. Arduino-based platforms generate .cpp source files and put all library code under src/; non-Arduino platforms generate plain .c files in a flat folder structure.

Platform	Framework	Layout
ESP32 + TWAI	Arduino / PlatformIO	Arduino (.cpp, src/)
ESP32 + WiFi GridConnect	Arduino / PlatformIO	Arduino (.cpp, src/)
RP2040 + MCP2517FD	Arduino (Earle Philhower core)	Arduino (.cpp, src/)
STM32F4xx + CAN	STM32 HAL (CubeIDE)	Non-Arduino (.c, flat)
TI MSPM0 + MCAN	TI DriverLib (CCS / Theia)	Non-Arduino (.c, flat)
None / Custom	Your own toolchain	Non-Arduino (.c, flat)

Note: The Developer Guide covers the Node Wizard in full detail, including every section, the CDI editor, and the difference between Arduino and non-Arduino folder layouts.

6. Understanding openlcb_user_config.h

This is the most important file in your project. It tells the library how much memory to allocate and which features to compile in. Every value is mandatory.

```
// --- Feature Flags -------------------------------------------------
// Uncomment to enable the protocols your node uses.
#define OPENLCB_COMPILE_EVENTS           // event producer/consumer
#define OPENLCB_COMPILE_DATAGRAMS        // needed for config memory
#define OPENLCB_COMPILE_CONFIG_MEMORY    // CDI/settings support

// --- Buffer Pool ---------------------------------------------------
#define USER_DEFINED_BASIC_BUFFER_DEPTH    32   // 16 bytes each
#define USER_DEFINED_DATAGRAM_BUFFER_DEPTH  4   // 72 bytes each
#define USER_DEFINED_SNIP_BUFFER_DEPTH      4   // 256 bytes each
#define USER_DEFINED_STREAM_BUFFER_DEPTH    1   // 512 bytes each

// --- Node Count ----------------------------------------------------
#define USER_DEFINED_NODE_BUFFER_DEPTH     4

// --- Events --------------------------------------------------------
#define USER_DEFINED_PRODUCER_COUNT        64
#define USER_DEFINED_CONSUMER_COUNT        32
```

Note: The Node Wizard generates this file for you. The BasicNode example already includes it with sensible defaults for ESP32.

7. Uploading and Running

In Arduino IDE

1. Open BasicNode.ino in Arduino IDE.
2. From Tools, select your board: "ESP32 Dev Module".
3. Select the correct COM port.
4. **BasicNode demo:** change the `#define NODE_ID` at the top of the sketch to your chosen value (see Section 4). **Wizard-generated project:** the Node ID was already set in Project Options when you generated the code — nothing to change here.
5. Click Upload.

What Happens at Power-On

When the node starts it goes through the LCC Alias Negotiation process. After a few seconds it becomes active and is visible to JMRI and other LCC tools on the bus.

Testing With JMRI

- Open JMRI and connect to your LCC network.
- Go to Tools > LCC > LCC Nodes. Your node should appear in the list.
- Right-click the node and choose "Configure". JMRI reads the CDI and shows your settings.
- You can change the node name and description from here.

> **Note:** The BasicNode demo does not implement real persistent storage. Changes you make to the node name and description through JMRI will not survive a power cycle. How you store configuration data is up to you — common choices include external EEPROM, an SD card, or the ESP32's internal flash (NVS/Preferences). The Developer Guide covers how to wire your storage choice into the config memory callbacks.

8. What's Next

You now have a working OpenLCB/LCC node. Here are the typical next steps:

Add Your Own Producer Events

```
// In Callbacks_initialize():
openlcb_node_t *node = OpenLcb_get_node(0);
OpenLcbApplication_register_producer_eventid(
    node, 0x0501010107770001, EVENT_STATUS_CLEAR);

// In Callbacks_on_100ms_timer_callback():
if (button_is_pressed())
    OpenLcbApplication_send_event_pc_report(
        node, 0x0501010107770001);
```

React to Consumer Events

```
// Register the event to listen for:
OpenLcbApplication_register_consumer_eventid(
    node, 0x0501010107770002, EVENT_STATUS_UNKNOWN);

// In your event consumer callback:
void Callbacks_on_event_consumed(
        openlcb_node_t *node, event_id_t event_id)
{
    if (event_id == 0x0501010107770002)
        turn_on_led();
}
```

Read the Developer Guide

The companion document, *OpenLcbCLib Developer Guide*, walks through every part of the system in detail: the Node Wizard, drivers, callbacks, CDI configuration, and more.

Online Help

Full documentation: https://jimkueneman.github.io/OpenLcbCLib/documentation/help/html/