Transports & Platform Notes

This page explains how MIDI data is transported across platforms and network protocols in `Assets/MIDI`.

For a per-platform feature matrix and OS/version requirements, see:

- Platforms & Limitations

How the managers choose transports (important)

On most platforms, `MidiManager` / `Midi2Manager` **initialize multiple backends in parallel** (e.g., platform MIDI + RTP-MIDI, and optionally Nearby when enabled).

Practical implications:

- **Sending may fan out**: a single `SendMidi...()` call can be forwarded to *each initialized backend* that supports it.
- **Targeting matters**: always send to a specific `deviceId` you intend to reach, and be careful if the same logical endpoint appears in multiple backends.
- **Device attach/detach events** are aggregated: device IDs from all active backends contribute to the manager's device sets.

This is by design: the transports are treated as "providers" that share one API surface.

Platform-native MIDI (Plugins/ + MidiPlugin.*.cs)

Most platforms use a native plugin (or WebGL JS plugin) plus a C# wrapper implementing `IMidiPlugin`.

Location:

- Native binaries: `Assets/MIDI/Plugins/<Platform>/...`
- C# wrappers: `Assets/MIDI/Scripts/MidiPlugin.<Platform>.cs`

Windows

- Wrapper: `MidiPlugin.Windows.cs`
- Additional helpers: `Assets/MIDI/Scripts/win32/` (P/Invoke and port abstractions)

macOS / iOS / Android / Linux

- Wrappers exist per platform and call into their respective native libraries.

Android devices (including Meta Quest)

Android-based headsets (e.g., Meta Quest) use the same Android transport layer, but there are a couple of practical setup items:

- **USB MIDI on Meta Quest** may require enabling a USB device intent filter during build postprocessing.
- **Bluetooth MIDI device pairing on Meta Quest** is commonly done via Android's Companion Device workflow (optional feature flag).

See:

- Build PostProcessing & Scripting Define Symbols (Meta Quest USB + `FEATURE_ANDROID_COMPANION_DEVICE`)
- Platforms & Limitations (Android API level notes)

WebGL

- Wrapper: `MidiPlugin.WebGL.cs`
- JS libraries: `Assets/MIDI/Plugins/WebGL/*.jslib`

Notes (important):

- Browser permissions and WebMIDI support affect availability.
- WebGL can't use raw UDP/TCP sockets, so **RTP-MIDI** and **UDP MIDI 2.0** are not available.
- WebGL currently can't handle USB MIDI 2.0 devices nor network MIDI 2.0; MIDI 2.0 is effectively limited to file workflows (e.g., MIDI Clip read/write).
- WebGL may not access other servers' resources via `UnityWebRequest` depending on browser/CORS rules; put files such as `.mid` into `StreamingAssets`.

WebGL Template requirement: expose `unityInstance`

Some WebGL flows expect the Unity runtime instance to be reachable from JS as `unityInstance`.

You should modify your WebGL template `index.html` to assign the created instance into a global variable:

```
var unityInstance = null;
script.onload = () => {
    createUnityInstance(canvas, config, (progress) => {
        progressBarFull.style.width = 100 * progress + "%";
    }).then((unityInst) => {
        unityInstance = unityInst;
    });
};
```

Alternatively:

- copy `Assets/MIDI/Samples/WebGLTemplates` into `Assets/WebGLTemplates`
- then select `Default-MIDI` or `Minimal-MIDI` in: `Project Settings > Player > Resolution and Presentation > WebGL Template`

(Unity manual reference: WebGL Templates.)

RTP-MIDI (Network MIDI 1.0)

RTP-MIDI (Apple Network MIDI style) is supported via a pure .NET implementation:

- Module: `Assets/MIDI/Scripts/RTP-MIDI-for-.NET/`
- Adapter: `Assets/MIDI/Scripts/MidiPlugin.RtpMidi.cs` (implements `IMidiPlugin`)

Key capabilities:

- Start one or more RTP-MIDI servers listening on UDP ports.
- Connect/disconnect remote endpoints.
- Receive and send MIDI 1.0 messages.

Limitations:

- The error correction protocol (**RTP MIDI Journaling**) is not supported.

Threading:

- Inbound RTP-MIDI events are forwarded to the manager using a main-thread-safe dispatch mechanism.

UDP MIDI 2.0 (Network UMP)

UDP MIDI 2.0 support is provided by:

- `Assets/MIDI/Scripts/Midi2Plugin.Udp.cs`
- Discovery helpers/dependencies: `Assets/MIDI/Scripts/UdpMidi2Discovery/`

Capabilities exposed (through `Midi2Manager`):

- Run UDP MIDI 2.0 servers.
- Discover servers on the LAN.
- Connect/disconnect clients.
- Send/receive UMP messages.
- Handle session-level commands (ping/retransmit/reset/NAK/bye).

See also:

- MIDI 2.0 / UMP (Midi2Manager)

“Nearby” transport (Google Nearby Connections)

There is a `nearby/` module and a corresponding plugin wrapper:

- `Assets/MIDI/Scripts/nearby/`
- `Assets/MIDI/Scripts/MidiPlugin.Nearby.cs`

This transport is an alternative MIDI link layer surfaced through the same MIDI 1.0 API.

Setup notes:

- Requires adding the Nearby dependency package (see Build PostProcessing & Scripting Define Symbols).
- Requires API level constraints on Android (see Platforms & Limitations).

Runtime note:

- When enabled, the Nearby backend requires periodic updates from Unity’s player loop. The project wires this through the manager when the transport is compiled in.

Source code example implementation (RTP-MIDI transport)

Use:

- `Assets/MIDI/Samples/Scripts/DocumentationExamples/RtpMidiTransportExample.cs`

Attach it to a GameObject and enter Play Mode. It starts an RTP-MIDI server.
Then connect to it using your OS / device RTP-MIDI routing tools (exact UI depends on platform).