Inter-App MIDI — Cross-platform Notes (Android, iOS/macOS, Linux)

This document explains how "**Inter-App MIDI**" works in this Unity MIDI plugin across platforms.

Because "Inter-App MIDI" is not a single universal standard name on every OS, this document uses the term in a **practical** sense:

> **Inter-App MIDI = sending/receiving MIDI between applications on the same machine/device**, usually via OS-provided virtual endpoints/ports or an OS service.

1) Quick summary by platform

Android (explicit Inter-App MIDI integration)

- Implemented via an Android AAR and accessed from Unity through the Android plugin wrappers.
- Location:
  - `Assets/MIDI/Plugins/Android/inter-app-midi-0.0.5.aar`
  - `Assets/MIDI/Scripts/MidiPlugin.Android.cs` (MIDI 1.0)
  - `Assets/MIDI/Scripts/Midi2Plugin.Android.cs` (MIDI 2.0 / UMP)
- You get an "Inter-App" transport in addition to USB MIDI and BLE MIDI.

iOS / macOS (CoreMIDI: inter-app routing via virtual endpoints)

- On Apple platforms, inter-app MIDI typically means **CoreMIDI endpoints** (apps can expose virtual MIDI destinations/sources, and the OS routes messages between them).
- In this project, Apple platform support is implemented in:
  - `Assets/MIDI/Scripts/MidiPlugin.Apple.cs` (MIDI 1.0)
  - `Assets/MIDI/Plugins/iOS/libMIDIPlugin.a` and `Assets/MIDI/Plugins/macOS/MIDIPlugin.bundle`
- From Unity's perspective, you still use `MidiManager` and device IDs; if another app exposes a CoreMIDI port, it can show up as a device and work like "inter-app MIDI".

Linux (ALSA/JACK: inter-app routing via MIDI ports)

- On Linux, inter-app MIDI routing is typically done via **ALSA MIDI ports** and/or **JACK MIDI** (depending on how the native plugin is built/configured).
- In this project, Linux support is implemented in:
  - `Assets/MIDI/Scripts/MidiPlugin.Linux.cs`
  - `Assets/MIDI/Plugins/Linux/MIDIPlugin.so`
- Again, Unity usage is the same: `MidiManager` sends to/receives from device IDs that correspond to ports exposed by the OS/stack.

2) How it fits the plugin architecture

MIDI 1.0

- Your app uses `MidiManager` for:
    - initialization (`InitializeMidi`)
    - device enumeration (via `DeviceIdSet`, `InputDeviceIdSet`, `OutputDeviceIdSet`)
    - event reception (Unity EventSystem handler interfaces)
    - sending MIDI 1.0 messages (Note On/Off, CC, SysEx, etc.)

Inter-app MIDI on iOS/macOS/Linux is effectively “just another set of MIDI endpoints” provided by the OS and surfaced through the platform plugin.

MIDI 2.0 / UMP

- Android has an explicit Inter-App MIDI path for MIDI 2.0 (`Midi2Plugin.Android.cs`).
- For iOS/macOS/Linux, MIDI 2.0 support depends on what the platform plugin and OS stack expose. In this repository, MIDI 2.0 plugin implementations are separate from the MIDI 1.0 Apple/Linux plugins, so “inter-app MIDI 2.0” may not be available in the same way as Android unless the platform’s MIDI 2.0 plugin backend supports it.

Practical takeaway:

- **Inter-app MIDI for MIDI 1.0 is clearly supported on iOS/macOS/Linux via their platform MIDI backends.**
- **Inter-app MIDI for MIDI 2.0 (UMP) is explicitly integrated for Android**; availability on other platforms depends on the MIDI 2.0 backend implementation you’re using there.

3) Usage (same API across platforms)

Receiving inter-app MIDI (MIDI 1.0)

1. Implement one or more MIDI 1.0 handler interfaces (e.g., `IMidiNoteOnEventHandler`).
2. Register the handler object with `MidiManager`.
3. Initialize MIDI (`MidiManager.InitializeMidi(...)`).
4. Connect another app to your app's exposed endpoint (or connect your app to another app's endpoint) using the OS/app routing UI/tools.
5. Your handlers will fire as messages arrive.

Sending inter-app MIDI (MIDI 1.0)

- Use `MidiManager.Instance.SendMidiNoteOn(...)` / `SendMidiControlChange(...)` / `SendMidiSystemExclusive(...)` etc.
- Choose the appropriate `deviceId` corresponding to the target app/port.

4) Platform-specific “how do I connect apps?” (conceptual)

> This section is intentionally conceptual because routing UI/tools vary by OS version and third-party apps.

Android

- Inter-App MIDI routing is handled by the Android Inter-App MIDI service/plugin integration.
- Device IDs correspond to endpoints exposed through that layer.

iOS / macOS

- Other apps expose CoreMIDI ports (virtual sources/destinations).
- You typically connect apps using:
  - the app’s own MIDI settings UI, and/or
  - OS-level MIDI routing tools (varies by device and workflow).
- Once connected, your Unity app sees the endpoints as MIDI devices.

Linux

- Apps expose ports via ALSA/JACK.
- Routing is typically done using external patchbay/connection tools.
- Once routed, your Unity app sends/receives via the corresponding ports.

5) Notes, limitations, and troubleshooting

“Inter-App MIDI” naming differs by platform

- Android: explicit “Inter-App MIDI” plugin/service in this repo.
- iOS/macOS: CoreMIDI endpoints (inter-app by design).
- Linux: ALSA/JACK ports (inter-app by design).

Device IDs may differ per platform

Device identifiers are platform/transport-defined. Do not assume the same `deviceId` format across OSes.

Duplicate sends (Android)

On Android, the MIDI 1.0 backend may send via multiple transports depending on what is initialized (USB/BLE/Inter-App). If you see duplicated messages, verify which endpoints you are targeting and whether multiple transports are active for the same logical device.

MIDI 2.0 inter-app expectations

If your goal is specifically **MIDI 2.0 UMP between apps**, verify that:

- your target platform’s MIDI 2.0 backend supports it, and
- the peer app actually supports MIDI 2.0/UMP on that platform.

6) Related docs

- MIDI 1.0 (MidiManager)
- MIDI 2.0 / UMP (Midi2Manager)
- Transports & Platforms

Source code example implementation (Inter-App MIDI “practical” example)

Inter-App MIDI is mostly about **routing**; the Unity-side code is simply “enumerate devices, pick the right deviceId, send/receive”.

Use the MIDI 1.0 quick-start component:

- `Assets/MIDI/Samples/Scripts/DocumentationExamples/Midi1QuickStartExample.cs`

How to validate Inter-App routing:

1. Start Play Mode with the script attached
2. Use your OS / MIDI app to route a virtual source/destination to the Unity app (or to another app)
3. Watch Unity logs:

- device attach events indicate the endpoint is visible
- Note On logs confirm inbound inter-app traffic

4. Send the test note and confirm it arrives in the target app