MPE (MIDI Polyphonic Expression)

This page documents the MPE support provided by:

- `Assets/MIDI/Scripts/MpeManager.cs`
- `Assets/MIDI/Scripts/IMpeEventHandler.cs`

MPE is implemented as a layer on top of MIDI 1.0-style messages and `MidiManager`.

What this layer does

Input (MPE → unified callbacks)

- Watches for **MPE Configuration Messages (MCM)** sent via Control Change on manager channels (0 for lower zone, 15 for upper zone).
- Builds an internal model of **lower/upper zones**, member channels, and zone state.
- When events arrive on member channels, it forwards them as **MPE events** to MPE-specific handler interfaces, reporting the **zone manager channel** as the zone identifier.

Output (unified calls → MPE channel allocation)

- Provides methods like `SendMpeNoteOn` / `SendMpeNoteOff` that:
  - select an appropriate member channel per note,
  - optionally reuse recently used channels,
  - track active notes per channel,
  - apply zone-wide parameters before playing new notes.

Zones and channels (conceptual)

- **Lower zone**: manager channel = 0, member channels = 1..N
- **Upper zone**: manager channel = 15, member channels = (15-N)..14

A device may have one or both zones configured, but the total member channels across both zones must fit the MIDI channel space.

Output API: MpeManager

`MpeManager` is a singleton-style manager (`MpeManager.Instance`) providing:

Zone setup

- `SetupMpeZone(deviceId, managerChannel, memberChannelCount)`

This:

- Validates `managerChannel` is `0` or `15`.
- Limits `memberChannelCount` to 0..15.
- Updates internal zone state.
- Sends the MPE configuration RPN/CC sequence to the device via `MidiManager`.

MIDI mode changes

- `ChangeMidiMode(deviceId, channel, mode)`

Supported modes:

- `3`: Omni Off, Poly
- `4`: Omni Off, Mono

(Other values are ignored.)

Sending events

- `SendMpeNoteOn(deviceId, channel, note, velocity)`
- `SendMpeNoteOff(deviceId, channel, note, velocity)`
- `SendMpePolyphonicAftertouch(deviceId, channel, note, pressure)`
- `SendMpeControlChange(deviceId, channel, function, value)`
- `SendMpeProgramChange(deviceId, channel, program)`
- `SendMpeChannelAftertouch(deviceId, channel, pressure)`
- `SendMpePitchWheel(deviceId, channel, amount)`
- Plus convenience wrappers for SysEx and system messages.

Channel selection behavior (summary)

When MPE is enabled for the target device:

- Note events are routed to a selected **member channel**.
- If the same note is already playing on some member channel, the manager may send Note Off first to stop it before replaying.
- Before starting a note, zone-wide parameters are applied to the selected channel:
  - controllers
  - poly aftertouch cache
  - program
  - channel aftertouch
  - pitch

If MPE is not configured, MpeManager falls back to sending directly to the specified channel.

Input API: MPE event handlers

Incoming MIDI 1.0 messages are interpreted as MPE when zone configuration is known.

Typical MPE handler interfaces include:

- `IMpeNoteOnEventHandler`
- `IMpeNoteOffEventHandler`
- `IMpeControlChangeEventHandler`
- `IMpePitchWheelEventHandler`
- `IMpeZoneDefinedEventHandler`
- etc.

The input layer reports:

- `deviceId`
- the **zone manager channel**
- event parameters (note, velocity, controller/value, etc.)

Practical usage patterns

1. Configure your output device for MPE:

- call `MpeManager.Instance.SetupMpeZone(...)`

2. Send notes using `SendMpeNoteOn/Off` rather than `MidiManager.SendMidiNoteOn/Off`.
3. Implement `IMpeZoneDefinedEventHandler` to react to incoming zone changes (useful if the device config changes dynamically).

Source code example implementation (MPE output)

Use:

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

Attach it to a GameObject and enter Play Mode.
It will:

- initialize MIDI 1.0
- configure a lower-zone MPE layout
- send a test note using MPE channel allocation