ChatGPT

软件架构设计概述

根据 Wwise 官方指南，建议采用**分层架构**，将 WAAPI 调用逻辑封装在后端服务层（如 `WwiseService`），前端采用 MVVM 模式进行数据绑定和交互 [1] [2]。使用 C# + Avalonia（跨平台 .NET XAML）开发 GUI，可以重用 WPF 经验，并利用 Audiokinetic 提供的 WAAPI C# 客户端或开源 WaapiCS 封装简化 JSON/WAMP 调用 [3] [4]。整体架构划分为：**前端/UI 层**、**视图模型层**、**后端服务层**（Wwise WAAPI 交互）和**数据模型层**。前端与后端通过事件或数据绑定松耦合，后端所有 WAAPI 调用采用异步执行，避免阻塞 UI 线程 [2]，保证界面响应流畅。

- **前端/UI 层**：使用 Avalonia 构建窗口界面，按 MVVM 原则将视图与业务逻辑分离。界面包含事件列表（树形结构）、事件详情面板（RTPC 滑块、Switch 下拉、播放/停止按钮等）和操作控件。通过数据绑定，ViewModel 接收后端查询结果并驱动 UI 更新。
- **视图模型层（ViewModel）**：定义数据模型和命令，包含 `EventViewModel`、`RtpcViewModel`、`SwitchViewModel` 等。该层处理用户操作（如滑块拖动、下拉选择、按钮点击）并调用后端服务。
- **后端服务层（WwiseService）**：封装 WAAPI 通信细节，提供事件查询、参数查询和声音控制接口。例如，调用 `ak.wwise.core.object.get` 查询事件及其子动作，从中识别出依赖的 GameParameter（RTPC）和 SwitchGroup [5]；再查询每个参数的 `@Min` / `@Max` 等属性及 SwitchGroup 的子状态列表，用于界面控件范围与选项生成 [5]。同时提供封装好的异步方法：`PostEvent`（播放事件）、`StopAll`（停止播放）、`SetRTPCValue`、`SetSwitch` 等调用 [6] [7] [8]。
- **数据模型层**：定义 `EventModel`、`RtpcModel`、`SwitchGroupModel`、`GameObject` 等 POCO 类。`EventModel` 包含事件 ID、名称、所属路径等，持有对应的 RTPC 和 Switch 对象列表，用于数据绑定。`RtpcModel` 包含参数 ID、名称、最小/最大值和当前值；`SwitchGroupModel` 包含开关组 ID、名称以及可选状态列表和当前选中状态。
- **解耦与扩展**：通过上述分层和 MVVM 设计，实现 UI 与 WAAPI 逻辑严格分离 [2]。各模块以接口或服务形式交互，方便未来扩展（如支持更多 WAAPI 功能、添加3D声源功能等）和单元测试 [9] [2]。

后端事件流处理

后端核心是 `WwiseService` 类（或多个服务类），负责管理 WAAPI 连接和所有对 Wwise 的操作。启动时，程序需先调用 `ak.soundengine.registerGameObj` 注册一个游戏对象，保存返回的 `gameObjectId` [10]，后续对声音引擎的操作（播放、RTPC、Switch）均以此 ID 为目标。WwiseService 包含以下关键方法：

- **连接初始化**：建立 WAAPI 连接（默认 `ws://localhost:8080/waapi`），确保 Wwise 正在运行并开启 WAAPI 功能 [11]。可以使用 Audiokinetic 官方的 `WaapiClient` 或 `WaapiCS` 来简化连接和 JSON 调用过程 [4]。连接成功后自动注册游戏对象。
- **事件查询与依赖提取**：提供方法按名称或路径查询事件对象，示例调用 `ak.wwise.core.object.get` [12]。获取事件后，再使用 `transform select children` 列出该事件下所有动作 [13]。根据返回的动作 `@ActionType` 判别类型：`GameParameter`（或 SetGameParameter）动作对应 RTPC 参数，`SetSwitch` / `SetState` 对应 SwitchGroup/State [14]。识别出依赖的所有 RTPC 和 SwitchGroup 后，再次调用 `object.get` 获取各对象属性——比如对 RTPC 查询其 `@Min` / `@Max`、模拟值 [5]；对 SwitchGroup 查询其子 Switch 容器或状态名列表 [5]。通过这些结果，后端将构造对应的 `RtpcModel` 和 `SwitchGroupModel` 列表，返回给前端 ViewModel 用于 UI 生成。

- **RTPC 控制**：当用户拖动某个 RTPC 滑块时，前端通过绑定触发 `WwiseService.SetRTPCValue(rtpcId, value)`。该方法内部调用 WAAPI 接口 `ak.soundengine.setRTPCValue`，传入参数 ID、数值和游戏对象 ID [6]。调用异步发出后，Wwise 声音引擎实时更新该参数值。应确保所有 WAAPI 调用异步执行，避免 UI 卡顿 [2]。
- **Switch 控制**：用户在 UI 选择某个 SwitchGroup 的状态时，触发 `WwiseService.SetSwitch(switchGroupId, stateId)`，内部调用 `ak.soundengine.setSwitch` [7]。传入 SwitchGroup 的 ID、所选 State 的 ID 以及游戏对象 ID，将对应状态发送给引擎，实时切换声源状态 [7]。
- **播放/停止事件**：在事件详情区提供“播放”和“停止”按钮。点击播放时调用 `WwiseService.PostEvent(eventId)`，内部使用 `ak.soundengine.postEvent` 将事件发送给引擎并开始播放 [8]。点击停止时可调用 `ak.soundengine.stopAll`（或 `stopEvent`）来停止当前所有声音 [8]。所有此类音频引擎控制函数均通过后端服务发起，返回结果后更新 UI（如播放状态指示等）。

以上后端逻辑通过事件或回调通知 ViewModel 更新状态。**所有 WAAPI 调用都在后台线程异步执行，UI 层通过数据绑定或事件监听自动刷新** [2]。例如，EventViewModel 可以定义 `LoadEventDependenciesCommand`，调用后端查询并填充 `RtpcList` 和 `SwitchList`，界面自动生成相应控件（见下文）。后端还应对可能的错误（如连接失败、对象不存在等）进行捕获，并通过错误事件通知 UI 提示用户。

前端 UI/UX 设计方案

前端界面遵循**直观清晰、层级分明**的设计理念。主窗口可分为两大区域：**左侧**是事件目录树（Soundboard 视图），**右侧**是当前选中事件的控制面板。

- **声音事件目录（左侧树状视图）**：左栏使用类似 Wwise 项目视图的**树形列表**，按照 Wwise 中的分类（如工作单元、文件夹）层级组织所有事件。这样用户可像在 Wwise 中浏览事件一般，快速定位或搜索音效事件。界面顶部可放置一个**搜索框**，支持按名称筛选事件。此处每个事件节点可显示名称和一个播放快捷按钮（可选）以直接试听，或仅作为导航。组件可使用 Avalonia 的 `TreeView` 控件，数据源绑定到构建好的事件模型树，支持动态加载（Lazy Loading）以提高性能。

- **事件详情与控制面板（右侧）**：当用户点击左侧树中的某个事件时，右侧显示该事件的详细控件面板，包括：

- **事件信息与控制按钮**：顶部显示事件名称，旁边有“播放”(Play) 和“停止”(Stop) 按钮。点击“播放”触发后端 `PostEvent`；点击“停止”调用 `StopAll`。播放期间按钮可变更状态（如“正在播放”状态），并可在界面某处显示当前播放状态。
- **RTPC 控制区**：列出该事件所依赖的所有 RTPC 参数。每个 RTPC 占一行，包含**参数名称标签**、**滑块控件** 和**数值显示**。滑块范围设置为该参数的最小值到最大值，中间数值对应参数范围 [6]。用户拖动滑块时（或输入数值时）立即调用 `setRTPCValue` 并实时调整声音效果 [6]。数值显示可双向绑定，同步展示当前参数值。控件布局可采用水平 StackPanel：左侧文字、右侧滑块和旁边数字文本框。
- **Switch 控制区**：列出所有相关的 SwitchGroup，每组使用一个**下拉菜单**显示可选状态 [7]。每个 SwitchGroup 显示其名称和一个下拉列表，列表项为该组所有可能的状态名称。用户选择状态后，调用 `setSwitch` 切换该开关状态 [7]。可考虑使用图标或颜色强调当前选中状态。
- **实时反馈**：可以在界面底部或状态栏显示当前的 RTPC 值和 Switch 状态（只读模式），或播放时长等信息。若播放失败或未能连接 WAAPI，应在界面提示错误。

整个 UI 各组件通过数据绑定与后台服务层解耦：例如，`EventViewModel.RtpcList` 是一个可观察集合，UI 自动为其中每个 `RtpcViewModel` 实例生成对应的控件行；同理 `SwitchGroupList` 动态生成对应下拉菜单[5]。这样设计使得前端页面不必硬编码参数控件数量，支持不同事件有不同数量和类型的控制项[5]。

用户交互流程

1. **应用启动**：自动连接到本机 Wwise WAAPI 并注册游戏对象。若连接失败，提示用户检查 Wwise 是否开启 WAAPI。
2. **浏览/搜索事件**：用户可在左侧树视图中展开工作单元、文件夹，浏览音效事件列表；或在搜索框输入事件名称，实时筛选匹配的事件节点。
3. **选择事件**：单击某个事件节点，应用调用后端查询该事件的子动作和依赖（RTPC、SwitchGroup），生成控制面板。界面右侧加载并显示相应的滑块和下拉菜单控件[5]。
4. **调整 RTPC/切换 Switch**：用户拖动 RTPC 滑块或输入数值时，界面调用后端 `setRTPCValue` 接口[6]，声音引擎实时更新效果；切换下拉菜单时调用 `setSwitch`[7]，即时改变声音状态。用户能听到效果的瞬时反馈，便于调试与预览音效。
5. **播放与停止**：在控制面板点击"播放"后端发送 `postEvent` 播放声音[8]；点击"停止"后端调用 `stopAll` 停止所有声音[8]。界面可在播放时禁用其他操作或显示动画提示。
6. **层级切换**：如需预览其他事件，用户返回左侧选择其它节点，步骤重演。应用持续保留 WAAPI 连接和已注册的游戏对象 ID，不需要多次注册。

通过上述设计，用户可以像使用 SoundCaster 一样，轻松搜索并试听项目中的任意事件，实时调节参数和开关，而无需离开 Wwise 编辑器。界面布局简洁、操作直观：左侧层级清晰，右侧控件布局有序，尽量减少点击和等待。Avalonia 的数据绑定和命令机制保证了 UI 与业务逻辑的同步和可测试性[9][2]。

开发任务与扩展

建议按阶段推进开发：第一步搭建 C# + Avalonia 项目框架，集成 WAAPI 客户端库，验证与 Wwise 的连接[2]；第二步实现后端事件查询与依赖提取逻辑（事件对象定位、RTPC/Switch 识别）；第三步开发前端界面（树状事件列表、滑块和下拉等控件）并与后端数据绑定；最后完善播放/停止功能、异常处理和用户提示[6][2]。采用 MVVM 和依赖注入等设计，可使项目**易于解耦、维护和扩展**：例如将来可支持多语言、网络访问或新增 WAAPI 功能（如3D 声源位置控制）而不影响现有模块[2][9]。

参考资料： Wwise 官方文档和社区指南详述了通过 WAAPI 查询事件、RTPC、Switch 的方法，以及控制引擎播放的函数（如 `ak.soundengine.postEvent`、`setRTPCValue`、`setSwitch`）[6][7][8]。以上架构设计在很大程度上参考了 "Wwise SoundCaster 独立客户端实现指南" 提出的分层和 MVVM 思路[1][2]。

---

[1][2][3][4][5][6][7][8][10][11][12][13][14] Wwise SoundCaster 独立客户端实现指南.pdf
file://file_0000000021f071f79a0afc9baf4626a7

[9] The MVVM Pattern | Avalonia Docs
https://docs.avaloniaui.net/docs/concepts/the-mvvm-pattern