

# 获取Polar数据流程1

目标：测试基本的技术流程。在仅有 Mac 与 iOS 模拟器（无真机、无 Polar 设备）的前提下，完成一条从 iOS 应用发出数据，经 UDP 到 Mac，再转为 LSL，被 LabRecorder 录制为 .xdf 的可用链路。为未来接入 Polar H10/Verity、皮电/呼吸/脑电等设备预留接口。

## 0. 架构概览（逻辑框架）

```
1  [iOS App] —UDP(局域网/本机)→ [Mac 代理脚本: UDP→LSL] —LSL→  
   [LabRecorder] → XDF 文件  
2  ↑ Swift/Polar SDK           ↑ Python + pylsl           ↑ pyxdf 验证
```

- 现阶段：iOS 模拟器发“心跳”假数据即可。蓝牙在模拟器不可用，后续真机接入 Polar 再做。
- 为什么要 UDP→LSL：UDP 通用、跨平台；LSL 负责与其他设备时间对齐与统一存储 (.xdf)。
- 扩展性：将来替换“心跳”为 Polar 数据（PPI/ECG），或其他传感器数据，沿用同一通道与录制流程。

## 1. 环境与工具（版本为本次实测）

- macOS 15.6（Apple Silicon, M 系列）
- Xcode 16.4（仅用于 iOS App 与模拟器）
- Python 3.11（系统/官方框架版均可）
- LabRecorder（macOS 构建），下载自 GitHub Releases（详见步骤）
- pylsl 与 pyxdf（Python 包，用于发布/读取 LSL）
- 可选：Homebrew（如用 brew 安装 liblsl）

## 2. 资源下载与安装

### 2.1 LabRecorder（macOS）

- 访问：<https://github.com/labstreaminglayer/App-LabRecorder/releases>
- Apple Silicon 选 LabRecorder-OSX\_arm64.tar.bz2（若最新发布没带 mac 资产，选上一版）
- 解压后得到 LabRecorder.app 与 LabRecorderCLI 等文件。
- 把 LabRecorder.app 拖到 /Applications 中，不要放在桌面，避免管理员权限问题。

首次启动常见问题与修复（已在本次实践验证）

某些包缺少苹果公证或带有隔离属性，直接双击会出现：

- “... is damaged and can't be opened”
- 或启动即崩溃（Code Signature Invalid）如下图



遇到的是Gatekeeper/签名问题，不是应用本身坏了。我们只做一件事：把 LabRecorder 解除隔离并进行临时本地签名，保证能在 macOS 15 上启动。

### 标准修复流程（在终端执行）：

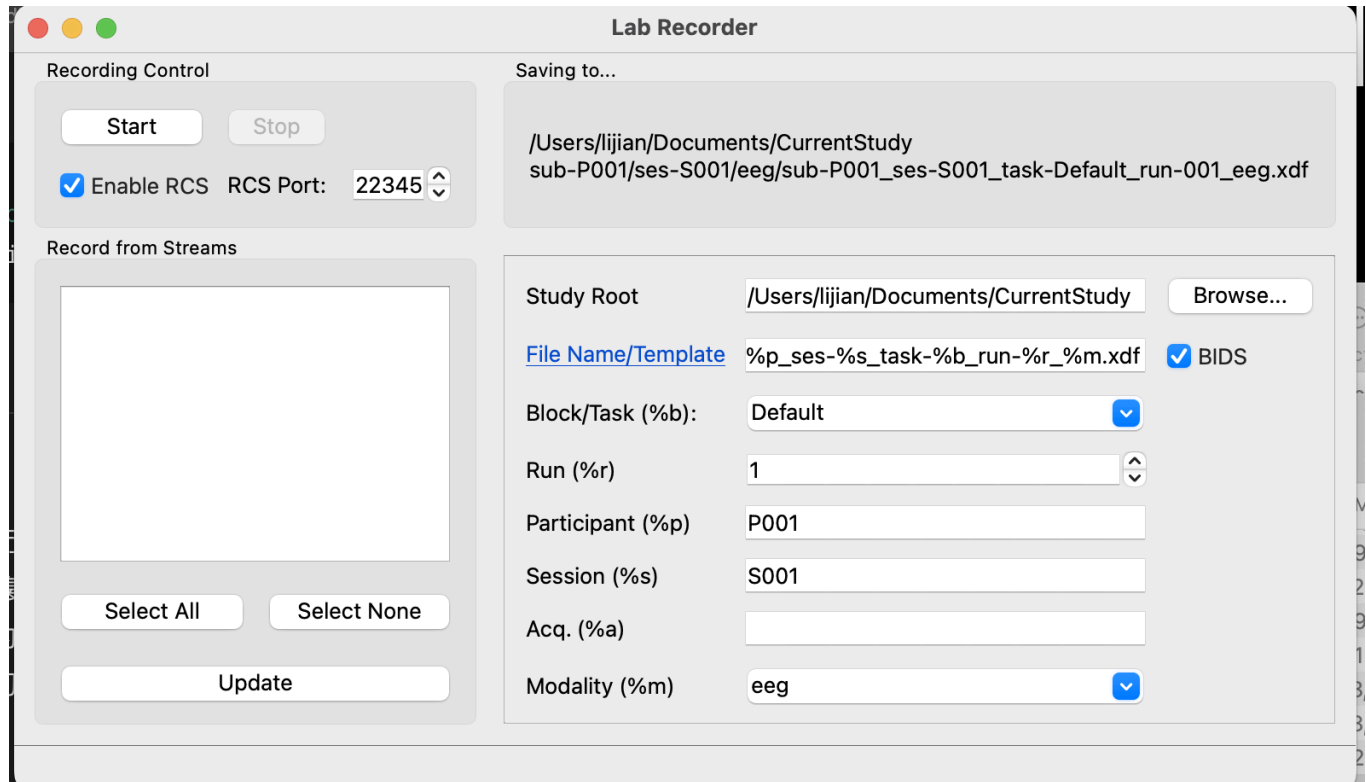
```
1  # 需要把 App 放到 /Applications,在终端中一条一条运行
2  # 1) 删除坏的符号链接，并用 Frameworks 里的真库建一个兼容名的链接
3  sudo rm -f
4  "/Applications/LabRecorder.app/Contents/MacOS/libls1.1.16.2.dylib"
5  sudo ln -s ../Frameworks/libls1.2.dylib
6  "/Applications/LabRecorder.app/Contents/MacOS/libls1.1.16.2.dylib"
7
8  # 可选：确认现在这个链接是有效的（应当指向 ../Frameworks/libls1.2.dylib）
9  ls -l "/Applications/LabRecorder.app/Contents/MacOS/libls1.1.16.2.dylib"
10
11 # 2) 清理隐藏的 AppleDouble 文件，避免 codesign 报 detritus not allowed
12 sudo find "/Applications/LabRecorder.app" -name '._*' -delete
13
14 # 3) 递归移除扩展属性（这次不要跟随符号链接：加 -h）
15 sudo xattr -h -r -c "/Applications/LabRecorder.app"
16
17 # 4) 深度本地重签（ad-hoc）
18 sudo codesign --force --deep -s - --timestamp=none
19 "/Applications/LabRecorder.app"
```

```

17
18 # 5) 验证
19 codesign -dv --verbose=4 "/Applications/LabRecorder.app" | head -n 8
20 spctl --assess --type execute --verbose "/Applications/LabRecorder.app" ||
    true
21

```

启动方式：首次在 Finder 中“右键 → 打开”，再点“打开”。看到主界面即成功。



### 3. Python 依赖安装

安装 pylsl (Python 绑定)

```

1 python3 -m pip install --upgrade pip
2 python3 -m pip install pylsl pyxdf`

```

若运行 pylsl 提示“LSL binary library file was not found”

方案一（推荐）：使用 LabRecorder 自带的 liblsl.2.dylib，将其路径写入环境变量。把下面两行添加到 ~/.zshrc（或 ~/.zprofile），然后 exec zsh 重载：

```

1 echo 'export PYLSL_LIB="/Applications/LabRecorder.app/Contents/Frameworks/liblsl.2.dylib"'
>> ~/.zshrc

```

```
2  echo 'export
    DYLD_LIBRARY_PATH="/Applications/LabRecorder.app/Contents/Frameworks:${DYLD
    >> ~/.zshrc
3  exec zsh`
```

方案二：Homebrew 安装系统级 libls

```
1  brew install labstreaminglayer/tap/ls1
2  echo 'export DYLD_LIBRARY_PATH="/opt/homebrew/lib:${DYLD_LIBRARY_PATH}"'
    >> ~/.zshrc
3  echo 'export PYLSL_LIB="/opt/homebrew/lib/libls1.2.dylib"' >> ~/.zshrc
4  exec zsh
```

方案三：写一个一次性的运行脚本（不改全局环境）。建 `run_markers.sh`：

```
1  #!/usr/bin/env bash
2  export PYLSL_LIB="/Applications/LabRecorder.app/Contents/Frameworks/libls1.
3  export
    DYLD_LIBRARY_PATH="/Applications/LabRecorder.app/Contents/Frameworks:${DYLD
4  python3 /Users/lijian/Desktop/GitHub/polar_bridge/send_markers.py
```

然后：

```
1  chmod +x run_markers.sh
2  ./run_markers.sh
```

## 4. Xcode 工程准备（iOS 模拟器端）

### 4.1 新建 App

- 打开 Xcode → Create New Project... → iOS → App
- Product Name: `PolarBridge`
- Team: 选择你的 Apple ID
- Organization Identifier: `com.yourname`（任意唯一字符串）
- Interface: SwiftUI, Language: Swift
- 取消勾选 “Include Tests”“Include Core Data”
- 选一个 iPhone 模拟器作为运行目标（因为目前无数据线）

→ 点 **Next**，把工程保存到你之前规划的目录，勾选 “Create Git repository”。

### 4.2 通过 SPM 添加 Polar BLE SDK（先只保证能编译）

- 菜单：File → Add Packages...
- 输入仓库地址： `https://github.com/polarofficial/polar-ble-sdk`
- 选择 `PolarBleSdk` 目标加入到 App target
- 在 `ContentView.swift` 顶部试加一行 `import PolarBleSdk` , Product → Build 出现 **Build Succeeded** 即可

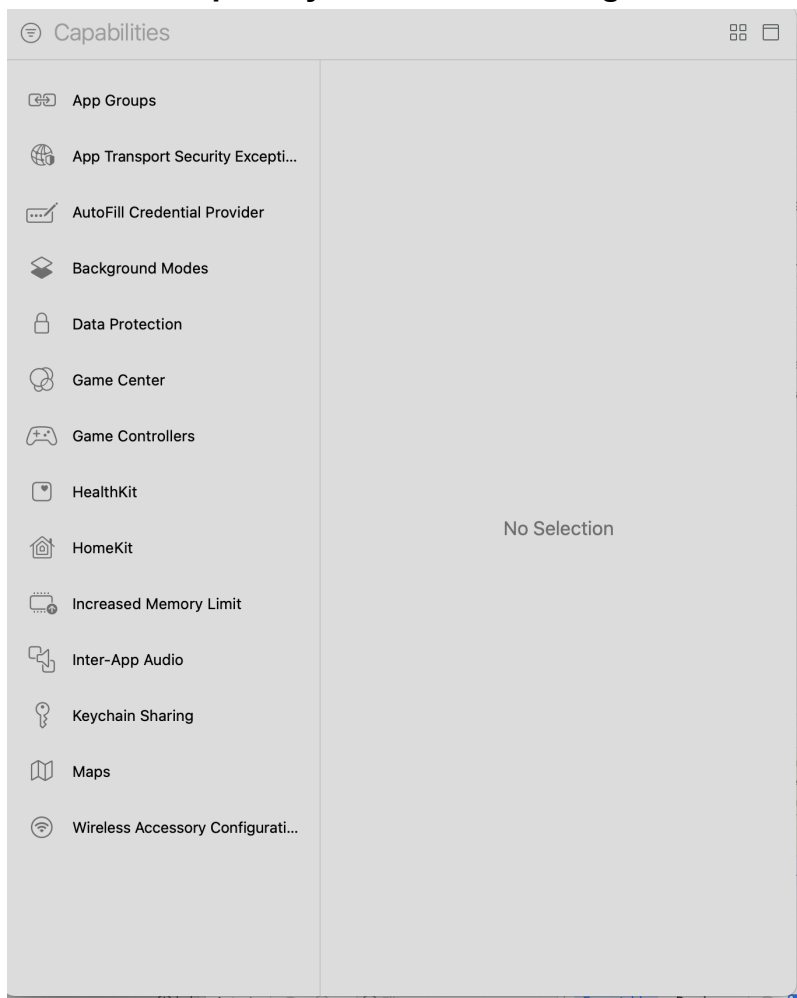
说明：模拟器下不会真正连蓝牙，先确保集成无误，后续真机再用。

## 4.3 配置后台与隐私权限

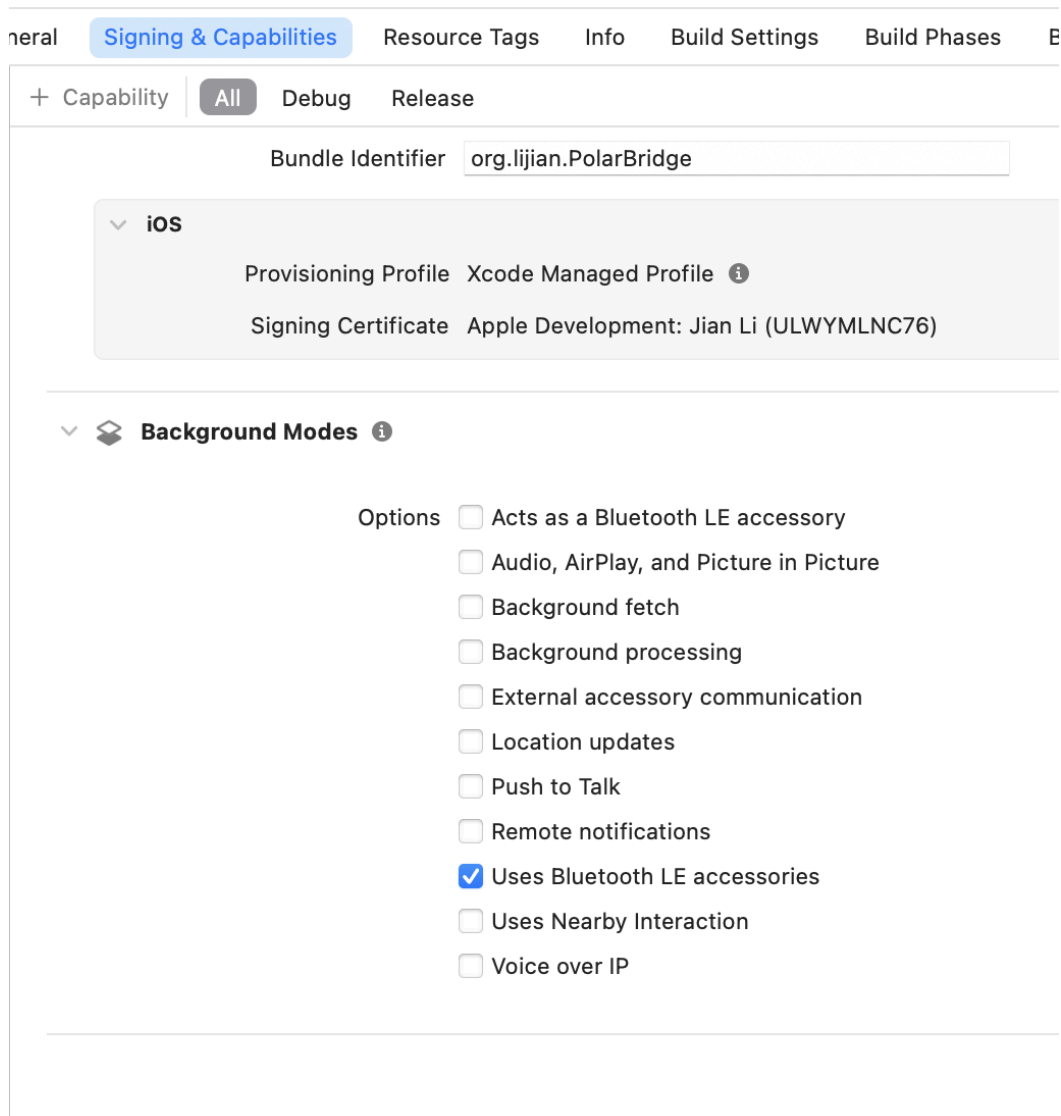
目的：向用户请求使用蓝牙采集与局域网发送数据。只改 Xcode 配置与 Info.plist。

### 启动蓝牙

1. 在左侧导航点选你的工程顶层（蓝色图标）→ Targets 里的 **PolarBridge**。
2. 打开页签 **Signing & Capabilities**。
3. 左上角点 **+ Capability**，搜索并添加 **Background Modes**。



4. 在出现的复选项里只勾选： - **Uses Bluetooth LE accessories**（这会在 Info 里加入 `UIBackgroundModes` 的 `bluetooth-central`）。



说明：只勾这一个就够了，其他如 Audio/Location 都不要动。

## 添加隐私与本地网络说明

1. 同一 Target 下切到 **Info** 页签（或直接打开项目里的 Info.plist）。
2. 在 **Custom iOS Target Properties** 列表点击 +，新增以下两项键值（Xcode 会帮你补齐人类可读的名字）：
  - **Privacy – Bluetooth Always Usage Description**  
Key: NSBluetoothAlwaysUsageDescription  
Value（建议文案，可直接粘贴中文）： 本应用需要使用蓝牙连接 Polar 等生理传感器，以进行实验数据采集（心率/PPG/ECG）。
  - **Privacy – Local Network Usage Description**  
Key: NSLocalNetworkUsageDescription  
Value： 本应用需要在局域网内把采集到的生理数据实时发送到你的 Mac 进行同步记录与分析。

说明：iOS 14+ 访问局域网会弹权限框，必须先写清楚用途，否则连接时会被系统拦截。“Custom iOS Target Properties” 是 Target 级别的键列表（写入到最终 Info.plist）。另一个“Information Property List” 视图只把部分键分组展示，所以你此刻只看到了 Required background modes。

构建已成功，说明两条隐私说明也会进入最终的 Info.plist。若想多一层确认：Product → Show Build Folder in Finder → Products/Debug-iphonesimulator/PolarBridge.app/Info.plist 打开即可看到合成后的完整文件。

（可忽略：不使用 Bonjour 就不用配置 NSBonjourServices。）

## 验证配置

- 菜单 **Product** → **Build** (⌘B)。
- 期待结果：**Build Succeeded**，且没有关于 Info.plist 缺少隐私说明或 Background Modes 的红色错误。

## 5. iOS 端最小 UDP 发送（模拟器可测）

目的：先验证“App 能把数据发到 Mac”，为以后对接 Polar 数据和 LSL 做准备。此步只做一件事：每秒从 App 向 Mac 发一条心跳，Mac 端用 Python 打印收到的内容。

### 5.1 发送器类 UdpSender.swift

在 Xcode 的 ios-app 工程里，新建一个 Swift 文件 UdpSender.swift，填入：

```
1  import Foundation
2  import Network
3
4  final class UdpSender {
5      private let connection: NWConnection
6      init(host: String, port: UInt16) {
7          let params = NWParameters.udp
8          self.connection = NWConnection(
9              host: NWEndpoint.Host(host),
10             port: NWEndpoint.Port(rawValue: port)!,
11             using: params
12         )
13         self.connection.start(queue: .global(qos: .utility))
14     }
15     func send(text: String) {
16         guard let data = text.data(using: .utf8) else { return }
17         connection.send(content: data, completion: .contentProcessed { _
18             in })
19     }
20 }
```

## 5.2 在 ContentView.swift 中定时发送心跳

打开 ContentView.swift，在 ContentView 里加入最小心跳（示例）：

```

1  import SwiftUI
2  import PolarBleSdk // 仅占位导入，证明包正常；本步不使用
3
4  struct ContentView: View {
5      @State private var sender = UdpSender(host: "127.0.0.1", port: 9001)
6      @State private var timer: Timer?
7
8      var body: some View {
9          VStack(spacing: 12) {
10             Image(systemName: "globe").font(.largeTitle)
11             Text("Hello, world!")
12             Text("UDP heartbeat → 127.0.0.1:9001")
13                 .font(.footnote).foregroundColor(.secondary)
14         }
15         .onAppear {
16             timer = Timer.scheduledTimer(withTimeInterval: 1, repeats:
17 true) { _ in
18                 let t = Date().timeIntervalSince1970
19                 let msg = #"{"type":"heartbeat","t":\#(t)}"#
20                 sender.send(text: msg)
21             }
22         }
23         .onDisappear { timer?.invalidate() }
24         .padding()
25     }
26 }


```

说明：在 **iOS 模拟器**里，127.0.0.1 指向你的 Mac 本机，所以可以直接发到 Python 接收端。将来在真机上，需要把 host 改成 **Mac 的局域网 IP**（例如 192.168.x.y）。



终端里运行的 `udp_recv.py` 每秒输出一行，能看到类似如下结果

```
8 while True:
9     data, addr = sock.recvfrom(65535)
10    try:
11        msg = data.decode("utf-8", errors="ignore")
12    except Exception:
13        msg = f"<{len(data)} bytes>"
14    print(f"{addr} {msg}")
15
```



```
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657803.364083}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657804.364054}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657805.364101}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657806.363387}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657807.3641338}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657808.3641062}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657809.363797}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657810.363202}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657811.364167}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657812.364173}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657813.364172}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657814.364193}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657815.364182}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657816.364221}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657817.364191}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657818.364217}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657819.364269}
('127.0.0.1', 64458) {"type":"heartbeat","t":1755657820.364255}
```

注意：xcode里显示以下提示（截图2）：Failed to send CA Event for app launch measurements for ca\_event\_type: 0 event\_name: com.apple.app\_launch\_measurement.FirstFramePresentationMetric Failed to send CA Event for app launch measurements for ca\_event\_type: 1 event\_name: com.apple.app\_launch\_measurement.ExtendedLaunchMetrics。不用担心，这两行“Failed to send CA Event ... LaunchMetrics”是模拟器里常见的诊断日志，不影响功能，也不需要处理。它来自系统的启动性能采样组件，在模拟器/沙盒里找不到对应文件时会提示。忽略即可。

## 6. Mac 端验证 UDP 通道（不依赖 LSL）

`udp_recv.py`（可用 VS Code/Terminal 运行）

```
1 import socket
```

```

2  PORT = 9001
3  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
4  sock.bind(("0.0.0.0", PORT))
5  print(f"listening on 0.0.0.0:{PORT}")
6  while True:
7      data, addr = sock.recvfrom(65535)
8      print(addr, data.decode("utf-8", errors="ignore"))
9

```

- 先运行 `udp_recv.py`
- 再运行 iOS 模拟器 App
- 期望每秒看到一条 `{"type":"heartbeat","t":...}`

## 7. LSL 最小验证 (Marker 流)

运行一个最小的 LSL Marker 发送脚本

### 7.1 发送 Marker

`send_markers.py`

```

1  from pylsl import StreamInfo, StreamOutlet
2  import time
3  info = StreamInfo(name="Markers", type="Markers", channel_count=1,
4                      nominal_srate=0, channel_format="string",
5                      source_id="marker_demo")
6  outlet = StreamOutlet(info)
7  print("sending markers every second...")
8  i = 0
9  while True:
10     tag = f"MARK_{i}"
11     outlet.push_sample([tag], time.time())
12     print("sent", tag)
13     i += 1
14     time.sleep(1.0)
15

```

### 7.2 在 LabRecorder 录制

- 打开 LabRecorder → 点击 **Update**，应该能看到一个名为 **Markers** 的流。
- 左侧看到 **Markers** 流 → 勾选 → 右上选择保存目录 → **Start**
- 数秒后 **Stop**，得到 `.xdf`

## 7.3 快速检查 XDF

最简单的方法是再写一个读取脚本

check\_xdf.py

```
1 import pyxdf, sys
2 p = sys.argv[1] if len(sys.argv)>1 else "record.xdf"
3 streams, header = pyxdf.load_xdf(p)
4 print([(s['info']['name'][0], len(s['time_stamps'])) for s in streams])
5
```

## 8. 建立 UDP→LSL 代理（把 iOS 心跳转成 LSL 流）

这一步把“UDP → LSL 代理”跑起来。这样 iOS App 发出的心跳，立刻就能进入 LabRecorder 的同一时间轴，为接 Polar 数据打通最后一段路。

udp\_to\_lsl.py

```
1 import socket, json
2 from pylsl import StreamInfo, StreamOutlet, local_clock
3
4 PORT = 9001
5 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6 sock.bind(("0.0.0.0", PORT))
7
8 info = StreamInfo(name="PB_UDP", type="json",
9                   channel_count=1, nominal_srate=0,
10                  channel_format="string", source_id="pb_udp_v1")
11 outlet = StreamOutlet(info)
12
13 print(f"listening UDP {PORT} -> LSL stream PB_UDP")
14 while True:
15     data, addr = sock.recvfrom(65535)
16     txt = data.decode("utf-8", errors="ignore")
17     # 可选：确保是 JSON 格式；不是也照样透传
18     try:
19         json.loads(txt)
20     except Exception:
21         pass
22     outlet.push_sample([txt], local_clock())
23     print("→", txt[:120])
24
```

## 运行顺序（本 MVP 的标准操作流程）

1. 打开终端 A：运行 `python3 udp_to_lsl.py`

注意：该py脚本路径根据实际路径，比如可能是 `python3 /Users/lijian/Desktop/GitHub/polar_bridge/udp_to_lsl.py` 如果运行出错，依赖找不到，参考[建立依赖方案一](#)的解决办法

2. 在 Xcode 里运行 iOS 模拟器 App，它每秒向 127.0.0.1:9001 发心跳

3. 打开 LabRecorder： **Update/Refresh** → 勾选 **PB\_UDP** → **Start** 录制

4. 数秒后 **Stop**，得到 `.xdf`

5. 用 `python3 check_xdf.py 路径/文件.xdf` 验证（应看到 `[('PB_UDP', N)]`）

## 9. 常见问题与解决

- **Xcode 报 “A build only device cannot be used...”**

说明当前目标是“仅构建设备”，选择一个模拟器或连真机。

- **Background Modes 里找不到 “Uses Bluetooth LE accessories”**

先把 **Background Modes** 能力添加到 Target，再在该卡片的复选框里勾选。或者在 Info.plist 手工加入

```
UIBackgroundModes = (bluetooth-central)。
```

- **Info 里只看到部分键值**

“Custom iOS Target Properties” 与 “Information Property List” 是两个视图，构建成功即表示键值已写入最终 Info.plist。

- **Python/pylsl 提示找不到 liblsl**

使用本文件第 3 节的环境变量方案，或用 Homebrew 安装 liblsl 并设置

```
DYLD_LIBRARY_PATH、PYLSL_LIB。
```

- **LabRecorder 无法打开/提示损坏/崩溃**

参照第 2.1 的“标准修复流程”（`find ... '.*' -delete`、`xattr -cr`、`codesign --deep -s -`、右键打开）。

如遇 `liblsl.1.16.2.dylib` 断链，按“特殊修复”修正为指向

```
Frameworks/liblsl.2.dylib。
```

- **LSL WARN “Could not bind multicast ...”**

常见的 IPv6 组播警告，可忽略，不影响功能。

- **录不到数据**

检查：防火墙是否允许 Python 与 LabRecorder 网络访问；LabRecorder 是否点击了 **Update/Refresh** 并勾选目标流；

UDP 端口是否一致（默认 9001）。

---

## 10. 后续接入 Polar 与多设备（展望）

- 真机 + Polar:
    - iOS 端通过 Polar BLE SDK 连接 H10/Verity，获取 PPI/ECG 后用与心跳相同的 UDP 管道发往 Mac。
    - 将 iOS 端报文升级为结构化 JSON，包含 `session_id/stream/packet_id/t_device/payload` 等字段，Mac 端 `udp_to_lsl` 可做丢包统计与统一时钟戳。
  - 多设备:
    - 保持“每种数据一个独立 LSL 流”的原则，并同时发一个 Marker 流（段落标记：`baseline/stim/intervention`）。
    - LabRecorder 勾选所有流一起录，确保 `.xdf` 时间轴一致。
  - 分析:
    - 后处理用 MATLAB、Python（neurokit2、mne 等），直接读取 `.xdf`，按 `stream name` 区分多源数据。
- 

## 附：本次用到的所有代码清单（含注释）

### A. `UdpSender.swift` (iOS)

```
1  import Foundation
2  import Network
3
4  /// 极简 UDP 发送器，使用苹果 Network.framework。
5  /// 初始化时指定 host/port；send(text:) 即可发。
6  final class UdpSender {
7      private let connection: NWConnection
8      init(host: String, port: UInt16) {
9          let params = NWParameters.udp
10         self.connection = NWConnection(
11             host: NWEndpoint.Host(host),
12             port: NWEndpoint.Port(rawValue: port)!,
13             using: params
14         )
15         self.connection.start(queue: .global(qos: .utility))
16     }
17     func send(text: String) {
18         guard let data = text.data(using: .utf8) else { return }
```

```

19         connection.send(content: data, completion: .contentProcessed { _
    in })
20     }
21 }
22

```

## B. ContentView.swift (iOS 心跳演示)

```

1  import SwiftUI
2  import PolarBleSdk // 先确保包可见；本示例未用
3
4  /// Demo: 每秒发送一条 JSON 心跳到 127.0.0.1:9001 (模拟器等于 Mac 本机)
5  struct ContentView: View {
6      @State private var sender = UdpSender(host: "127.0.0.1", port: 9001)
7      @State private var timer: Timer?
8
9      var body: some View {
10         VStack(spacing: 12) {
11             Image(systemName: "globe").font(.largeTitle)
12             Text("Hello, world!")
13             Text("UDP heartbeat → 127.0.0.1:9001")
14                 .font(.footnote).foregroundColor(.secondary)
15         }
16         .onAppear {
17             timer = Timer.scheduledTimer(withTimeInterval: 1, repeats:
true) { _ in
18                 let t = Date().timeIntervalSince1970
19                 let msg = #"{"type":"heartbeat","t":\#(t)}"#
20                 sender.send(text: msg)
21             }
22         }
23         .onDisappear { timer?.invalidate() }
24         .padding()
25     }
26 }
27

```

## C. udp\_recv.py (UDP 简单接收, 非 LSL)

```

1  import socket
2  PORT = 9001
3  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
4  sock.bind(("0.0.0.0", PORT))
5  print(f"listening on 0.0.0.0:{PORT}")

```

```

6  while True:
7      data, addr = sock.recvfrom(65535)
8      print(addr, data.decode("utf-8", errors="ignore"))
9

```

## D. send\_markers.py (pylsl Marker)

```

1  from pylsl import StreamInfo, StreamOutlet
2  import time
3
4  # 1 通道、0Hz 的字符串流, 用作事件/段落标记
5  info = StreamInfo(name="Markers", type="Markers", channel_count=1,
6                    nominal_srate=0, channel_format="string",
7                    source_id="marker_demo")
8  outlet = StreamOutlet(info)
9
10 print("sending markers every second...")
11 i = 0
12 while True:
13     tag = f"MARK_{i}"
14     outlet.push_sample([tag], time.time())
15     print("sent", tag)
16     i += 1
17     time.sleep(1.0)
18

```

## E. udp\_to\_lsl.py (UDP → LSL 桥)

```

1  import socket, json
2  from pylsl import StreamInfo, StreamOutlet, local_clock
3
4  PORT = 9001
5  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
6  sock.bind(("0.0.0.0", PORT))
7
8  # 输出一个 JSON 文本单通道流, 供 LabRecorder 录制
9  info = StreamInfo(name="PB_UDP", type="json",
10                   channel_count=1, nominal_srate=0,
11                   channel_format="string", source_id="pb_udp_v1")
12  outlet = StreamOutlet(info)
13
14  print(f"listening UDP {PORT} -> LSL stream PB_UDP")
15  while True:
16      data, addr = sock.recvfrom(65535)
17

```

```
17     txt = data.decode("utf-8", errors="ignore")
18     # 尽量保持原样；如果不是 JSON 也照样透传
19     try:
20         json.loads(txt)
21     except Exception:
22         pass
23     outlet.push_sample([txt], local_clock())
24     print("→", txt[:120])
```

## F. check\_xdf.py (快速检查 .xdf)

```
1  import pyxdf, sys
2  p = sys.argv[1] if len(sys.argv)>1 else "record.xdf"
3  streams, header = pyxdf.load_xdf(p)
4  print([(s['info']['name'][0], len(s['time_stamps'])) for s in streams])
```

---

## 结语

本 MVP 已经证明：在没有真机和生理设备的前提下，你可以稳定地获取一条可复用的“App→UDP→LSL→XDF”链路，并完成录制与验证。后续把 UDP 载荷替换为 Polar 数据即可纳入同一流程；再叠加皮电、呼吸、脑电等数据流，使用 LabRecorder 一次性录到同一时间轴。以上步骤、脚本与修复办法足以让没有经验的新手复刻出同样的结果。