

MPS_ReadBin.dll 函数接口说明文档

1. 概述

MPS_ReadBin.dll 是一个用于读取采集设备生成的二进制波形文件（.bin）的函数库。

项目	说明
DLL 文件名	MPS_ReadBin.dll
调用约定	__cdecl （C 调用约定，参数由调用方清理栈）

1.1 典型调用流程

Bin_Open()	→ 获得 BIN_HANDLE
↓	
Bin_ReadHeader()	→ 解析文件头，得到通道数/采样率/记录时长等信息
↓	
Bin_GetConfigString()	→ 导出可读配置文本用于日志/界面(非必要函数，可自由选择是否使用)
↓	
Bin_ReadWaveData()	→ 按范围循环读取波形数据
↓	
Bin_Close()	→ 关闭文件、释放BIN_HANDLE

2. 数据结构定义

2.1 BIN_HANDLE（不透明句柄）

<pre>typedef void* BIN_HANDLE;</pre>

- 由 Bin_Open 创建，使用完毕后必须传给 Bin_Close 释放。
- 句柄为 NULL 表示无效/已关闭。

2.2 Config（文件头配置结构体，跨语言二进制兼容）

字段	类型	字节数	含义
DataStartPos	ULONG	4	数据区相对偏移（内部数据类型，读取波形时会自动处理，其值可忽略）
ChannelCount	LONG	4	通道数
SampleRate	LONG	4	采样率
AcquisitionStartTime	INT64	8	采集开始时间（UTC 时间戳，秒）
SampleDataType	LONG	4	采样数据类型编码（内部数据类型，读取波形时会自动处理，其值可忽略）
ScaleFactor	double	8	物理量缩放系数（内部数据类型，读取波形时会自动处理，其值可忽略）
Offset	double	8	物理量偏置量（内部数据类型，读取波形时会自动处理，其值可忽略）
TotalSamples	ULONG	4	每通道总采样点数
DeviceModel[64]	char[64]	64	设备硬件型号字符串（ANSI/GBK 编码，以 \0 结尾）

字段	类型	字节数	含义
ChannelNames[64] [128]	char[64] [128]	8192	各通道名称二维数组，第 i 路名称位于 ChannelNames[i]

3. API 函数详解

3.1 Bin_Open — 打开文件

```
BIN_HANDLE Bin_Open(const char* folderPath, int fileIndex);
```

参数	方向	说明
folderPath	输入	.bin 文件的文件完整路径，或存放 .bin 文件的文件夹路径。传 NULL 或空字符串则使用默认路径：DLL所在目录\DATA
fileIndex	输入	文件序号，从 1 开始（非从0开始）。DLL 在目录下扫描 *.bin 并按文件名排序后取第 fileIndex 个

返回值

- 成功：返回非空 BIN_HANDLE 。
- 失败：返回 NULL （目录不存在、无 .bin 文件、序号越界、文件打开失败）。

3.2 Bin_ReadHeader — 读取并解析文件头

```
bool Bin_ReadHeader(BIN_HANDLE hBin, Config* outConfig);
```

参数	方向	说明
hBin	输入	Bin_Open 返回的句柄
outConfig	输出	接收解析后的 Config 结构体（跨语言时传 Config* ）

返回值：true 成功；false 句柄无效。

3.3 Bin_GetConfigString — 导出可读配置文本

```
void CDECL Bin_GetConfigString(  
    const Config* config,  
    int language,  
    char* outBuffer,  
    size_t bufferSize  
);
```

参数	方向	说明
config	输入	已由 Bin_ReadHeader 解析得到的配置结构体
language	输入	0 = 中文（LANG_ZH），1 = 英文（LANG_EN）
outBuffer	输出	跨语言版本的接收缓冲区
bufferSize	输入	outBuffer 容量（字节）。内容超长时按 _TRUNCATE 安全截断

返回值：版本 B 无返回；outBuffer 为 NULL 或 bufferSize 为 0 时直接返回。

文本内容：包含采集时间（UTC）、通道数、采样率、总点数、设备型号及逐通道名称，适合写入日志或显示在界面上。

3.4 Bin_ReadWaveData — 读取波形

```
bool CDECL Bin_ReadWaveData(
    BIN_HANDLE    hBin,
    __int64       startSampleIdx,
    __int64       samplesToRead,
    float*        outData,
    __int64*      outNextOffset
);
```

参数	方向	说明
hBin	输入	句柄
startSampleIdx	输入	起始采样点索引（从 0 开始）。小于 0 自动钳为 0
samplesToRead	输入	计划读取的采样点数
outData	输出	用户预分配的输出缓冲区，大小须 ≥ 实际读取点数 × ChannelCount × sizeof(float)
outNextOffset	输出	实际读到的下一个起始索引（= 本次起点 + 实际读取点数），可直接作为下次调用的 startSampleIdx

返回值： true 成功； false 句柄无效或文件定位/读取失败。

输出数据布局（交错格式）

```
[Ch0_Smp0, Ch1_Smp0, ..., ChN_Smp0,  Ch0_Smp1, Ch1_Smp1, ..., ChN_Smp1, ...]
索引计算公式: outData[ sampleIdx * ChannelCount + channelId ]
```

说明

- 若请求范围超出文件末尾，自动截断到文件末尾；若起点已越界，则不读取任何数据， outNextOffset 保持为入参起点。

3.5 Bin_Close — 关闭并释放

```
void CDECL Bin_Close(BIN_HANDLE hBin);
```

参数	方向	说明
hBin	输入	待关闭句柄；传 NULL 直接返回，安全

说明：关闭文件句柄并 delete 内部上下文对象。关闭后该句柄不可再用。

4. 跨语言调用注意事项

- 调用约定：所有导出函数均为 __cdecl 。C# 中需声明 CallingConvention = CallingConvention.Cdecl 。
- 引用参数映射：C++ 中的 Config& outConfig 、 __int64& outNextOffset 在 C 接口中即指针，跨语言时传地址/ ref 即可。
- 结构体布局： Config 为 1 字节定长数组结构体，C# 端需 [StructLayout(LayoutKind.Sequential, Pack=1)] 并用 [MarshalAs(UnmanagedType.ByValTStr, SizeConst=...)] 映射字符数组。
- 64 位兼容： __int64 / INT64 对应 C# 的 long 、LabVIEW 的 I64 、Python 的 int 。
- 字符串编码： DeviceModel 与 ChannelNames 为 ANSI（GBK）字节流，Python/LabVIEW 读取后需按 GBK 解码。
- 缓冲区所有权： outData / outDataBuffer / outBuffer 均由调用方分配与释放；DLL 不持有其指针。
- 线程安全：每个 BIN_HANDLE 对应一个打开的文件句柄，同一 handle 不要多线程并发读取；多文件并行处理请为每个文件单独 Bin_Open 。
- 句柄泄漏：任何打开成功的句柄都必须配对调用 Bin_Close ，否则句柄与堆内存泄漏。

5. 最小调用示例（C/C++）

```
#include "MPS_ReadBin.h"    // 自行声明或由DLL提供头文件

int main()
{
    BIN_HANDLE h = Bin_Open(NULL, 1);           // 打开 DATA 目录下第1个 .bin
    if (!h) return -1;

    Config cfg;
    Bin_ReadHeader(h, &cfg);

    // 打印配置（跨语言版）
    char info[2048] = {0};
    Bin_GetConfigString(&cfg, 0, info, sizeof(info));
    printf("%s", info);

    const long n = 1024;
    float* buf = new float[n * cfg.ChannelCount];
    long next = 0;

    while (next < cfg.TotalSamples)
    {
        Bin_ReadWaveData(h, next, n, buf, &next);
        // ... 处理 buf 中 [0 .. 实际读取点数*ChannelCount) 的交错数据 ...
    }

    delete[] buf;
    Bin_Close(h);
    return 0;
}
```

6. 跨语言绑定示例

6.1 C# P/Invoke 声明

适用 .NET Framework / .NET Core / .NET 5+。注意：DLL 为 VS2005 编译，若为 32 位则 C# 进程需勾选“**首选 32 位**”或平台目标 `x86`；若已编译为 64 位则选 `x64`。

```

using System;
using System.Runtime.InteropServices;
using System.Text;

public static class MpsBin
{
    private const string DLL = "MPS_ReadBin.dll";

    // ---- Config 结构体: 必须与 DLL 端 1 字节对齐 ----
    [StructLayout(LayoutKind.Sequential, Pack = 1)]
    public struct Config
    {
        public uint   DataStartPos;
        public int    ChannelCount;
        public int    SampleRate;
        public long   AcquisitionStartTime;
        public int    SampleDataType;
        public double ScaleFactor;
        public double Offset;
        public uint   TotalSamples;

        // 定长字符数组: 用 ByValTStr 映射 char[64] / char[128]
        [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 64)]
        public string DeviceModel;

        // 二维通道名数组 [64][128] 无法用 ByValTStr 直接映射,
        // 最稳妥的做法是按 64*128 字节的扁平字节数组接收, 再自行按 GBK 切分解码。
        [MarshalAs(UnmanagedType.ByValArray, SizeConst = 64 * 128)]
        public byte[] ChannelNames;

        /// <summary>从扁平字节数组中取出第 ch 个通道名 (GBK 解码) </summary>
        public string GetChannelName(int ch)
        {
            if (ChannelNames == null) return "";
            int off = ch * 128;
            int len = 0;
            while (len < 128 && ChannelNames[off + len] != 0) len++;
            return Encoding.GetEncoding("GBK").GetString(ChannelNames, off, len);
        }
    }

    // ---- 函数导入 ----
    // 句柄用 IntPtr 表示
    [DllImport(DLL, CallingConvention = CallingConvention.Cdecl)]
    public static extern IntPtr Bin_Open(string folderPath, int fileIndex);

    // Config& 用 ref Config; 注意 StructLayout.Pack=1
    [DllImport(DLL, CallingConvention = CallingConvention.Cdecl)]
    [return: MarshalAs(UnmanagedType.Bool)]
    public static extern bool Bin_ReadHeader(IntPtr hBin, ref Config outConfig);

    // char* + size_t: 缓冲区用 StringBuilder, size_t 用 UIntPtr 兼顾 32/64 位
    [DllImport(DLL, CallingConvention = CallingConvention.Cdecl)]
    public static extern void Bin_GetConfigString(
        ref Config config,
        int language,
        StringBuilder outBuffer,
        UIntPtr bufferSize);

    // float* 直接传 float[]; __int64& 用 ref long
    [DllImport(DLL, CallingConvention = CallingConvention.Cdecl)]
    [return: MarshalAs(UnmanagedType.Bool)]

```

```
public static extern bool Bin_ReadWaveData(
    IntPtr hBin,
    long startSampleIdx,
    long samplesToRead,
    [In, Out] float[] outData,
    ref long outNextOffset);

[DllImport(DLL, CallingConvention = CallingConvention.Cdecl)]
public static extern void Bin_Close(IntPtr hBin);
}
```

调用示例

```
IntPtr h = MpsBin.Bin_Open(null, 1);
if (h == IntPtr.Zero) { /* 打开失败 */ }

MpsBin.Config cfg = new MpsBin.Config();
MpsBin.Bin_ReadHeader(h, ref cfg);

// 配置文本
var sb = new StringBuilder(2048);
MpsBin.Bin_GetConfigString(ref cfg, 0, sb, (UIntPtr)sb.Capacity);
Console.WriteLine(sb.ToString());

// 循环读波形
const int chunk = 1024;
float[] buf = new float[chunk * cfg.ChannelCount];
long next = 0;
while (next < (long)cfg.TotalSamples)
{
    long outNext = 0;
    int readN = (int)Math.Min(chunk, (long)cfg.TotalSamples - next);
    MpsBin.Bin_ReadWaveData(h, next, readN, buf, ref outNext);
    for (int i = 0; i < readN * cfg.ChannelCount; i++) { /* 处理 buf[i] */ }
    next = outNext;
}
MpsBin.Bin_Close(h);
```

坑点提醒

- Bin_Open 传 null（不是 ""）才走默认 \DATA 目录。
- ChannelNames 用扁平 byte[8192] 接收后，必须按 GBK 编码解码，不能用 UTF8 / Default（中文 Windows 下 Default 恰为 GBK，但服务端/容器上不可靠）。
- size_t 在 C# 中不要声明为 int：若将来 DLL 重编译为 64 位，4 字节会与 DLL 的 8 字节错位；用 UIntPtr 一次到位。

6.2 LabVIEW 调用说明

LabVIEW 通过 调用库函数节点（Call Library Function Node, CLFN） 加载 DLL。核心难点是 Config 结构体与 C 的 char[] 布局对齐。

6.2.1 CLFN 通用配置

配置项	值
库路径	浏览选择 MPS_ReadBin.dll
调用约定（Calling Convention）	C（即 __cdecl，不要选 StdCall）
函数名（Function Name）	Bin_Open / Bin_ReadHeader / ... 逐个选择

6.2.2 各函数的参数映射

DLL 参数类型	LabVIEW 控件类型	说明
<code>BIN_HANDLE</code> （指针）	U32（32 位 DLL） 或 U64（64 位 DLL） 数值控件	LabVIEW 把指针当整数值持有即可，不做解引用
<code>const char* folderPath</code>	C String 句柄 （String 控件，参数勾选“String”，格式选 <code>C string</code> ）	传空字符串即走默认 <code>\DATA</code>
<code>int / LONG / ULONG</code>	I32 / U32	
<code>__int64</code>	I64	
<code>double</code>	DBLE	
<code>Config&</code>	簇（Cluster） ，见 8.2.3	
<code>float* outData</code>	数组（1D Array of SGL） ，预分配大小后作为"数组数据指针"传入	勾选“Array”→“Data Pointer”，并在数组尺寸输入端给出 <code>点数×通道数</code>
<code>__int64& outNextOffset</code>	I64 控件，参数勾选“ Pass by Value ”改为“ Pointer to Value ”（即 LabVIEW 里的"按指针传递"）	
<code>char* outBuffer + size_t</code>	C String 预分配缓冲区 + U32/U64 长度	

6.2.3 在 LabVIEW 中复刻 Config 簇

按以下顺序把控件拖入簇（顺序必须与 DLL 结构体字段完全一致，因为 LabVIEW 簇按控件顺序连续排布）：

- 1. U32 控件 —— `DataStartPos`
- 2. I32 控件 —— `ChannelCount`
- 3. I32 控件 —— `SampleRate`
- 4. I64 控件 —— `AcquisitionStartTime`
- 5. I32 控件 —— `SampleDataType`
- 6. DBL 控件 —— `ScaleFactor`
- 7. DBL 控件 —— `Offset`
- 8. U32 控件 —— `TotalSamples`
- 9. **DeviceModel**：拖入一个 **1D 字节数组（Array of U8）**，预先初始化长度为 64
- 10. **ChannelNames**：拖入一个 **1D 字节数组（Array of U8）**，预先初始化长度为 `64 × 128 = 8192`

为什么用 U8 字节数组而不是 LabVIEW 字符串？ LabVIEW 的字符串是"句柄+长度"结构，内存里第一个 4 字节是长度指针，与 C 的 `char[64]` 连续字节布局完全不同；直接传字符串会导致结构体整体错位。用预分配定长的 U8 数组才能保证与 `char[]` 一一对应。

簇配置完成后，在 CLFN 里把该参数选为“**Adapt to Type**”或“**Handles by Value**”，并确保字节对齐为 **1 字节（Pack=1）**——LabVIEW 簇默认按元素自然对齐，需要在簇属性里关闭**对齐填充**（或在 CLFN 的参数类型里手工指定结构体成员的紧凑排布；若版本不支持，最稳妥是把簇所有元素都用 U8 字节流自己拼，避免依赖 LabVIEW 对齐规则）。

6.2.4 读取通道名

`Bin_ReadHeader` 返回簇后，把 `ChannelNames` 的 8192 字节 U8 数组按 `ch*128` 偏移切片，用“**字节数组至字符串转换**”（String → Byte Array 反向）节点，编码选 **GBK / ANSI**，并按 `\0` 截断，即得到第 `ch` 路通道名。

6.3 Python ctypes 绑定

适用 Python 3.6+，32 位 Python 加载 32 位 DLL，64 位 Python 加载 64 位 DLL（位数必须一致）。

```

# -*- coding: utf-8 -*-
from ctypes import (
    cdll, c_uint32, c_int32, c_int64, c_double, c_float, c_char,
    c_void_p, c_bool, c_size_t, byref, POINTER, Structure,
)

# ---- 1. 定义 Config 结构体 (_pack_ = 1 对应 #pragma pack(1)) ----
MAX_DEVICE_MODEL_LEN = 64
MAX_CHANNEL_COUNT     = 64
MAX_CHANNEL_NAME_LEN  = 128

class Config(Structure):
    _pack_ = 1
    _fields_ = [
        ("DataStartPos",      c_uint32),
        ("ChannelCount",      c_int32),
        ("SampleRate",        c_int32),
        ("AcquisitionStartTime", c_int64),
        ("SampleDataType",     c_int32),
        ("ScaleFactor",        c_double),
        ("Offset",             c_double),
        ("TotalSamples",       c_uint32),
        # char[64] 与 char[64][128] 直接用定长字符数组
        ("DeviceModel",        c_char * MAX_DEVICE_MODEL_LEN),
        ("ChannelNames",       c_char * (MAX_CHANNEL_COUNT * MAX_CHANNEL_NAME_LEN)),
    ]

# ---- 2. 加载 DLL ----
dll = cdll.LoadLibrary(r"C:\path\to\MPS_ReadBin.dll")

# ---- 3. 显式声明函数签名（强烈建议，否则 ctypes 默认全当 int 处理） ----
# BIN_HANDLE = void*
dll.Bin_Open.restype = c_void_p
dll.Bin_Open.argtypes = [c_char_p, c_int32]          # folderPath, fileIndex

dll.Bin_ReadHeader.restype = c_bool
dll.Bin_ReadHeader.argtypes = [c_void_p, POINTER(Config)]

dll.Bin_GetConfigString.restype = None
dll.Bin_GetConfigString.argtypes = [
    POINTER(Config), c_int32, c_char_p, c_size_t
]

dll.Bin_ReadWaveData.restype = c_bool
dll.Bin_ReadWaveData.argtypes = [
    c_void_p,                # hBin
    c_int64,                 # startSampleIdx
    c_int64,                 # samplesToRead
    POINTER(c_float),        # outData (float*)
    POINTER(c_int64),        # outNextOffset
]

dll.Bin_Close.restype = None
dll.Bin_Close.argtypes = [c_void_p]

# ---- 4. 调用示例 ----
if __name__ == "__main__":
    # folderPath 传 None 走默认 \DATA
    h = dll.Bin_Open(None, 1)
    if not h:
        raise RuntimeError("Bin_Open 失败: 目录无 .bin 或文件被占用")

```

```
cfg = Config()
dll.Bin_ReadHeader(h, byref(cfg))

print("通道数 =", cfg.ChannelCount, " 采样率 =", cfg.SampleRate,
      " 总点数 =", cfg.TotalSamples)
print("设备型号 =", cfg.DeviceModel.decode("gbk", errors="ignore"))

# 解码通道名（按 128 字节切块，GBK）
ch_bytes = bytes(cfg.ChannelNames)
for i in range(cfg.ChannelCount):
    raw = ch_bytes[i*128:(i+1)*128]
    name = raw.split(b"\x00", 1)[0].decode("gbk", errors="ignore")
    print(f" 通道{i+1} = {name}")

# 读取波形（float 物理值）
import ctypes
chunk = 1024
nch = cfg.ChannelCount
buf = (ctypes.c_float * (chunk * nch))()
nxt = c_int64(0)

while nxt.value < cfg.TotalSamples:
    ok = dll.Bin_ReadWaveData(h, nxt, chunk, buf, byref(nxt))
    if not ok:
        break
    read_n = nxt.value # 实际推进后的起点 = 已读到的点数
    # buf[0 .. read_n*nch] 为交错 float 数据
    # 例如取第 0 通道前 10 点: [buf[i*nch+0] for i in range(10)]

# 配置文本
info = ctypes.create_string_buffer(2048)
dll.Bin_GetConfigString(byref(cfg), 0, info, len(info))
print(info.value.decode("gbk", errors="ignore"))

dll.Bin_Close(h)
```

- Python 端坑点**
- `folderPath` 传 `None` 等价于 C 的 `NULL`，走默认目录；传字符串必须是 **bytes**（`r"C:\data".encode()`）或用 `c_char_p`。
 - 结构体里的 `c_char * N` 读出来是 `bytes`，要 `.decode("gbk")`。
 - `ChannelNames` 按扁平 `c_char*8192` 接收后，按 `i*128` 切片。
 - 输出数组用 `(c_float * (n*ch))()` 预分配，再传给函数；不要直接传 `Python List`，`ctypes` 不会自动 `marshal` 成连续 `float` 数组。
 - 不写 `argtypes` 时 `ctypes` 会把所有参数当 `C int`，64 位指针会被截断成 32 位导致崩溃——**务必显式声明签名**。

7. 函数一览速查表

函数	作用	关键参数	返回
<code>Bin_Open</code>	打开目录下第 N 个 .bin	<code>folderPath</code> , <code>fileIndex</code>	句柄 / <code>NULL</code>
<code>Bin_ReadHeader</code>	解析文件头	<code>hBin</code> , <code>outConfig</code>	<code>bool</code>
<code>Bin_GetConfigString</code>	配置格式化文本	<code>config</code> , <code>lang</code> , <code>buf</code> , <code>size</code>	<code>void</code>
<code>Bin_ReadWaveData</code>	读取 float 物理值	<code>hBin</code> , <code>start</code> , <code>count</code> , <code>outData*</code> , <code>outNext*</code>	<code>bool</code>

函数	作用	关键参数	返回
<code>Bin_Close</code>	关闭并释放	hBin	void