

《数据接口 LANDCex.dll 使用说明》

武汉市蓝电电子股份有限公司

<http://www.whland.com>

目 录

目 录	1
第一章 文件组成说明	2
第二章 测试数据的逻辑结构	3
2.1 循环子表 (Cycle Table)	3
2.2 工步子表 (Step Table)	3
2.3 记录子表 (Rec Table)	3
第三章 库文件 LANDCex.dll 输出函数	5
3.1 版本检查	5
3.2 特殊函数	5
3.3 读取测试概要信息	6
3.4 读取详细数据	7
3.5 读取通道信息	11
第四章 典型的调用方法	12

《数据接口 LANDCex.dll 使用说明》

第一章 文件组成说明

文件名（或文件夹）	是否必须	说明
LANDCex.dll	必须	核心 DLL 库文件
LANHE.sys	必须	设备数据库文件（测试数据必须来自合法设备）
CexConst.h	VC 必须	常量定义
LANDCexDll.h	非必须，VC 推荐	动态（显式）加载 DLL 库文件的头文件
LANDCexDll.cpp	非必须，VC 推荐	动态（显式）加载 DLL 库文件的源代码
Column-ID & Col-Id-Str.pdf	重要文档	Column-ID & Col-Id-Str 的定义及说明
\Sample1(VS2015)	一个 VC 示例	1) 使用时必须将测试数据“SAMSUNG-D828.cex” 拷贝至 C:\ 2) VS 2015 编译并测试通过
\Sample2(ExcelVBA)	Interface(V30002) for Excel-VBA.txt 重要文档	Excel -VBA 需要的接口函数声明
	几个 Excel-VBA 示例	Excel 2007 / Excel 2010 测试通过

第二章 测试数据的逻辑结构

测试数据在逻辑上可以理解为由“循环子表”、“工步子表”和“记录子表”三张数据表相互交错而行成的一个复合数据表。了解这种数据的逻辑结构，对于准确理解和使用输出函数非常重要！

在没有歧义的场所，“循环子表”、“工步子表”和“记录子表”有时也会简称为“循环表”、“工步表”和“记录表”。

本文档选用的英文用词：

“Cycle”——“循环”，“Step”——“工步”，“Rec (Recorder)”——“记录”。

以下是各数据子表的截图（请注意，表头是自动变化的）：

2.1 循环子表(Cycle Table)

循环序号	充电容量/Ah	放电容量/Ah	效率/%	放电中压/V	恒流充入比...	平台/%	平台时间	循环保...	放电DCIR/mOhm
[+]	001	0.000	14.154	0.00	3.5122	0.00	14.27	00:04:50.81	0.00
[+]	002	26.868	26.853	99.94	3.6147	95.43	53.59	00:34:32.05	189.72
[+]	003	26.877	26.865	99.95	3.6142	95.49	53.27	00:34:20.47	100.04
[+]	004	26.878	26.868	99.96	3.6136	95.47	53.25	00:34:20.14	100.01
[+]	005	26.884	26.871	99.95	3.6136	95.44	53.24	00:34:20.19	100.01

2.2 工步子表(Step Table)

[提示] 如果忽略那些变灰的数据，其实就是一张简单的数据表

模式序号	工步模式	工步时间	容量/Ah	比容量/mAh/g	能量/Wh	电容/F	比能量/Wh/kg	中值电压/V
[-]	001	0.000	14.154	0.00	3.5122	0.00	14.27	00:04:50.81
[+]	0001 静置	00:10:00.03	0.000	0.0	0.00	0.0	0.0	3.6809
[+]	0002 恒流放电	00:33:58.04	14.154	0.0	49.07	0.0	0.0	3.5122
[+]	0003 静置	00:10:00.04	0.000	0.0	0.00	0.0	0.0	2.9948
[-]	002	26.868	26.853	99.94	3.6147	95.43	53.59	00:34:32.05
[+]	0004 恒流充电	01:01:32.45	25.641	0.0	96.08	0.0	0.0	3.7180
[+]	0005 恒压充电	00:11:48.99	1.228	0.0	5.09	0.0	0.0	4.1490
[+]	0006 静置	00:10:00.06	0.000	0.0	0.00	0.0	0.0	4.1437
[+]	0007 恒流放电	01:04:26.72	26.853	0.0	97.70	0.0	0.0	3.6147
[+]	0008 静置	00:10:00.06	0.000	0.0	0.00	0.0	0.0	2.9880
[-]	003	26.877	26.865	99.95	3.6142	95.49	53.27	00:34:20.47
[+]	0009 恒流充电	01:01:35.65	25.664	0.0	96.14	0.0	0.0	3.7175
[+]	0010 恒压充电	00:11:43.44	1.213	0.0	5.03	0.0	0.0	4.1490
[+]	0011 静置	00:10:00.01	0.000	0.0	0.00	0.0	0.0	4.1437
[+]	0012 恒流放电	01:04:28.36	26.865	0.0	97.72	0.0	0.0	3.6142
[+]	0013 静置	00:10:00.06	0.000	0.0	0.00	0.0	0.0	2.9849

2.3 记录子表(Rec Table)

[提示] 如果忽略那些变灰的数据，其实也是一张简单的数据表

记录序号	测试时间	步骤时间	电流/A	容量/Ah	电压/V	能量/Wh	系统时间
[-]	001	0.000	14.154	0.00	3.5122	0.00	14.27
[+]	0001 静置	00:10:00.03	0.000	0.0	0.00	0.0	0.0
[-]	0002 恒流放电	00:33:58.04	14.154	0.0	49.07	0.0	0.0
	23	00:10:00.03	00:00:00.00	-25.002	0.000	3.6559	0.00
	24	00:10:30.09	00:00:30.06	-25.002	0.209	3.6368	0.76
	25	00:11:00.15	00:01:00.12	-25.000	0.418	3.6297	1.52
	26	00:11:30.18	00:01:30.15	-25.000	0.627	3.6248	2.28
	27	00:12:00.21	00:02:00.18	-25.002	0.835	3.6207	3.03
	28	00:12:30.27	00:02:30.25	-25.000	1.043	3.6167	3.79
	29	00:13:00.29	00:03:00.26	-25.002	1.252	3.6130	4.54

综上，可以看出三个数据子表之间的逻辑关系。即：通常情况下，一个“循环”包含多个“工步”，而一个“工步”又包含多个“记录”。

而且，我们也很容易看出：每个数据都可以用 3 个参数来唯一定位，即：①数据位于哪个表（子表）？②在该表的哪一列？③在该表的哪一行？

但是，为了简化编程接口，我们将全部的 3 个子表中的所有数据“列”统一进行编号，分别用唯一的数字化的 `columnID` 来标识它们。这样，定位任何一个数据只需要 2 个参数，即：

数据所在的“列” ID: `columnID`

- 1) 各列的 ID 定义参见头文件 `CexConst.h`。

并且，为了程序的易读性，`columnID` 也分别配置了唯一的字符串化的“列” Id 供选用，这里称为 `ColIdStr`。

例如：记录子表的电压，`columnID=0x4008`，`ColIdStr="Rec.Voltage"`；

再如：工步子表的容量，`columnID=0x2004`，`ColIdStr="Step.Capacity"`。

- 2) 数据所在的“行”序号: `nRow`

请注意，各子表中的“行”分别在各子表内独立编号，并且，0 是起始的第一行。

第三章 库文件 LANDCex.dll 输出函数

【注】参数用词说明：

- a) “Cycle”--“循环”，“Step”--“工步”，“Rec(Recorder)”--“记录”；
- b) “Mode”--“工作模式”，例如静置、恒流充电、恒流放电、恒压放电等；
- c) 所有的序号都是从 0 开始计数，例如 nRow, nCycle, nStep 以及函数返回的序号值等；
- d) 所有的输出函数，均可用于 VC 和 VB，也能用于其它的绝大部分编程语言。

以下为输出函数的原型（Function Prototype）及说明：

3.1 版本检查

1. DLL 接口版本检查
<div>1) bool __stdcall CheckDllVersion (DWORD dwVerRequested)</div> <div>【描述】验证 DLL 的接口版本的兼容性。强烈建议首先验证 DLL 的接口版本！</div> <div>【参数】dwVerRequested: 指定程序需要的 DLL 接口版本。可用的版本号参见头文件 CexConst.h。目前最新版本为 LANDCEX_DLL_VER_0_3_1_3，即 0x00030102（VB 中为 &H30103）。</div> <div>【返回值】false，不兼容；true，兼容。</div>
<div>2) DWORD __stdcall GetDllVersion (void)</div> <div>【描述】获取 DLL 的接口版本号。推荐使用 CheckDllVersion()而不是直接使用 GetDllVersion()，更容易保持用户端二次开发代码的兼容性。</div>
<div>3) DWORD __stdcall GetDllProperty (void)</div> <div>【描述】该函数为升级预留，暂时可以忽略它。</div>

3.2 特殊函数

1. 日期数据格式
<div>1) bool __stdcall EnableUSDateFormat (bool bEnable=true)</div> <div>【描述】将输出的日期数据设置为（或取消）美式习惯。默认日期格式为“年/月/日”顺序，而美式习惯为“MM/DD/YYYY”顺序。</div> <div>【参数】bEnable: true 表示启用，false 表示取消。</div> <div>【返回值】返回本函数调用前的状态，即原来是否启用。</div>

2. 获取工作模式的名称
<div>1) bool __stdcall GetDescriptionOfMode (BYTE cMode, VARIANT& vDesc)</div> <div>【描述】通过工作模式 ID 获取其名称。</div> <div>【参数】cMode: 工作模式 ID。其常用的取值参见头文件 CexConst.h；</div> <div> vDesc: 获取的工作模式名称，标记为 VT_BSTR 类型的字符串。</div> <div>【返回值】false，失败；true，成功。</div>

3.3 读取测试概要信息

1. 调入和释放文件的概要信息（对象句柄）	
1) <code>HANDLE __stdcall LoadBriefInfo (const char* pszPath)</code>	<p>【描述】将测试数据文件的概要信息调入内存。只有成功调入后，才能访问其概要信息的具体内容。概要信息访问完毕，必须调用 <code>FreeBriefInfo</code> 释放内存。</p> <p>【参数】<code>pszPathName</code>: 指定数据文件，可以包含路径，其扩展名通常为“<code>cex</code>”。</p> <p>【返回值】非 0 成功；0 失败（数据文件未找到，或文件格式、版本不对，或者数据来自非法设备）。</p> <p>【提示】调入文件时，接口函数将首先检查该测试数据是否来自合法设备，所以您需要同时将设备数据库文件 <code>LANHE.sys</code> 与 <code>LANDCex.dll</code> 放在一个目录下。</p>
2) <code>void __stdcall FreeBriefInfo (HANDLE hBriefInfo)</code>	<p>【描述】释放由 <code>LoadBriefInfo</code> 调入内存中的概要信息。</p>
2. 获取概要信息内容	
1) <code>bool __stdcall GetChannel (HANDLE hBriefInfoOrDataObj, BYTE& cBoxNo, BYTE &cChI, DWORD& dwDownId, WORD& wDownVer)</code>	<p>【描述】获取概要信息中的测试通道信息。</p> <p>【参数】<code>hBriefInfoOrDataObj</code>: 可以由 <code>LoadBriefInfo</code> 调入的仅拥有概要信息的对象，也可以是由 <code>LoadData</code> 调入的拥有详细数据的对象；</p> <p><code>cBoxNo</code>: 获取的设备的箱号；</p> <p><code>cChI</code>: 获取的通道编号；</p> <p><code>dwDownId</code>: 获取的下位机 ID；</p> <p><code>wDownVer</code>: 获取的下位机版本号。</p> <p>【返回值】<code>false</code>，失败；<code>true</code>，成功。</p>
2) <code>bool __stdcall GetBattNo (HANDLE hBriefInfoOrDataObj, VARIANT& vBattNo)</code>	<p>【描述】获取电池编号（字符串）。</p> <p>【参数】<code>hBriefInfoOrDataObj</code>: 由 <code>LoadBriefInfo</code> 调入的概要信息对象，或由 <code>LoadData</code> 调入的详细数据对象；</p> <p><code>vBattNo</code>: 获取的电池编号，标记为 <code>VT_BSTR</code> 类型的字符串。</p> <p>【返回值】<code>false</code>，失败；<code>true</code>，成功。</p>
3) <code>__time32_t __stdcall GetStartTime (HANDLE hBriefInfoOrDataObj)</code>	<p>【描述】获取测试开始时间。</p> <p>【参数】<code>hBriefInfoOrDataObj</code>: 由 <code>LoadBriefInfo</code> 调入的概要信息对象，或由 <code>LoadData</code> 调入的详细数据对象。</p> <p>【返回值】返回值实质上为 <code>long</code> 型：表示自 1970/01/01 00:00:00 开始计时，已经过去的秒数。</p>
4) <code>__time32_t __stdcall GetEndTime (HANDLE hBriefInfoOrDataObj)</code>	<p>【描述】获取测试结束时间。</p> <p>【参数】<code>hBriefInfoOrDataObj</code>: 由 <code>LoadBriefInfo</code> 调入的概要信息对象，或由 <code>LoadData</code> 调入的详细数据对象。</p> <p>【返回值】与 <code>GetStartTime</code> 类似。</p>
5) <code>bool __stdcall IsTimeHighResolution (HANDLE hBriefInfoOrDataObj)</code>	<p>【描述】判断是否为高精度时间。</p> <p>【参数】<code>hBriefInfoOrDataObj</code>: 由 <code>LoadBriefInfo</code> 调入的概要信息对象，或由 <code>LoadData</code> 调入的详细数据对象；</p> <p>【返回值】<code>false</code>，表示不是高精度时间；<code>true</code>，表示是高精度时间。</p>

6) bool __stdcall GetFormationName (HANDLE hBriefInfoOrDataObj, VARIANT& vName)

【描述】获取化成名（字符串）。

【参数】hBriefInfoOrDataObj: 由 LoadBriefInfo 调入的概要信息对象，或由 LoadData 调入的详细数据对象；

vName: 获取的化成名，标记为 VT_BSTR 类型的字符串。

【返回值】false，失败；true，成功。

3.4 读取详细数据

1. 调入和释放数据对象（句柄）

1) HANDLE __stdcall LoadData (const char* pszPath, int nHopeUnitScheme=USCH_BaseOn_Auto)

【描述】将测试数据文件的详细数据调入内存。只有成功调入后，才能访问其概要信息和详细数据。数据访问完毕，必须调用 FreeData 释放内存。

【参数】pszPath: 指定数据文件(.cex)，可以包含路径；

nHopeUnitScheme: 指定单位方案，其取值参见头文件 CexConst.h。

【返回值】非 0 成功；0 失败（数据文件未找到，或文件格式、版本不对，或者数据来自非法设备）。

【提示】调入文件时，接口函数将首先检查该测试数据是否来自合法设备，所以您需要同时将设备数据库文件 LANHE.sys 与 LANDCex.dll 放在一个目录下

2) void __stdcall FreeData (HANDLE hDataObj)

【描述】释放由 LoadData 调入内存中的详细数据。

2. 数据访问(依据数字化的列 ID 和字符串化的列 ID)

1) int __stdcall GetRows (HANDLE hDataObj, UINT columnID)

【描述】获取指定数据列的行数，即数据的个数。

【参数】hDataObj: 由 LoadData 调入的详细数据对象；

columnID: 指定数据的列 ID，其取值参见头文件 CexConst.h。

【返回值】>0 成功；<=0 失败（入口参数无效）。

2) int __stdcall GetRows2 (HANDLE hDataObj, const char* pszColIdStr)

【描述】与上文的 GetRows(...) 完全对应。

3) bool __stdcall GetDataEx (HANDLE hDataObj, UINT columnID, int nRow, VARIANT& vData)

【描述】获取指定位置的数据。

【参数】hDataObj: 由 LoadData 调入的详细数据对象；

columnID: 指定数据的列 ID，其取值参见头文件 CexConst.h；

nRow: 指定数据的行序号；

vData: 返回指定位置的数据。

【返回值】false，失败；true，成功。

【提示】当返回成功时，返回值名义上为 VARIANT 类型，它通常可以精确地转换为一个明确的数据类型。其具体情况，视传入的参数 columnID 而定。

绝大多数的情况下，可以精确转换为一个 float 类型（即 VT_R4 型）。以下是特殊情况：

Column-ID	Col-Id-Str	精确的返回值类型	示例或补充说明
0x1000	"Cycle.Index"	long (即 VT_I4 型)	long nIndex = (long)(_variant_t)GetData(...);
0x2001	"Step.Index"		
0x4001	"Rec.Index"		
0x400f	"Rec.SysTimeLong"		

0x2002	"Step.Mode"	BYTE (即 VT_UI1 型)	BYTE cMode = (BYTE)(_variant_t) GetData(...); 以下是几个返回值的常数定义（参见头文件 CexConst.h）： #define PM_CONST_CURRENT_DISCHARGE 0x02 #define PM_CONST_CURRENT_CHARGE 0x03 另：可继续通过 GetDescriptionOfMode 函数获取文本名称。
0x400b	"Rec.SysTime"	BSTR (即 VT_BSTR 型)	CString strSysTime = (LPCSTR)(_variant_t) GetData(...);

4) bool __stdcall GetDataEx2 (HANDLE hDataObj, const char* pszColIdStr, int nRow, VARIANT& vData)

【描述】与上文的 GetDataEx(...) 完全对应，仅仅是为了程序的易读。

3. 数据访问（返回指定类型的数据）
<div>1) float __stdcall GetDataAsFloat(HANDLE hDataObj, UINT columnID, int nRow)</div> <div>【描述】该函数是上文 GetDataEx(...) 的特殊情况，只能用于访问 float 类型（即 VT_R4 型）的数据，如电压、电流等等。</div> <div>【返回值】返回 float 型的数据。</div>
<div>2) BYTE __stdcall GetDataAsByte(HANDLE hDataObj, UINT columnID, int nRow)</div> <div>【描述】该函数是上文 GetDataEx(...) 的特殊应用，只能用于访问 BYTE 型（即 VT_UI1 型）的数据，如工步模式。</div> <div>【返回值】返回 BYTE 型的数据。</div>

4. 数据访问（仅仅用于兼容旧版）
<div>1) VARIANT __stdcall GetData (HANDLE hDataObj, UINT columnID, int nRow)</div> <div>【描述】与上文的 GetDataEx (...) 功能完全等效。</div>
<div>2) VARIANT __stdcall GetData2 (HANDLE hDataObj, const char* pszColIdStr, int nRow)</div> <div>【描述】与上文的 GetDataEx2(...) 功能完全等效。</div>

5. 数字化的列 ID 与 字符串化的列 Id 相互转换
<div>1) UINT __stdcall Str_to_ColumnID (const char* pszColIdStr)</div> <div>【描述】从字符串化的列 Id 获取其对应的数字化的列 ID。</div> <div>【返回值】返回 0xffffffff，失败；反之成功。</div>
<div>2) bool __stdcall ColumnID_to_Str (UINT columnID, VARIANT& vColIdStr)</div> <div>【描述】从数字化的列 ID 获取其对应的字符串化的列 Id。</div> <div>【返回值】false，失败；true，成功。</div>

6. 获取数据列的文字描述
<div>1) bool __stdcall GetDescriptionOfColumn (UINT columnID, VARIANT& vDesc)</div> <div>【描述】获取“列”ID 的文字描述。</div> <div>【参数】columnID: 指定数据的列 ID，其取值参见头文件 CexConst.h; vDesc: 返回指定数据列 ID 对应的文字描述，标记为 VT_BSTR 类型的字符串。</div>

【返回值】 false，失败；true，成功。

7. 获取数据列的物理单位名称

1) bool __stdcall GetUnitNameOfColumn (HANDLE hDataObj, UINT columnID, VARIANT& vName)

【描述】 获取当前数据文件指定子表的物理单位名称。

【参数】 hDataObj: 由 LoadData 调入的详细数据；
columnID: 指定数据的列 ID，其取值参见头文件 CexConst.h；
vName: 返回当前数据文件指定子表的物理单位名称。

【返回值】 false，失败；true，成功。

8. 通过记录序号获取循环序号和工步序号

1) int __stdcall GetCycleFromRec (HANDLE hDataObj, int nRec, int* pnStep=nullptr)

【描述】 获取当前记录的循环序号和工步序号。

【参数】 hDataObj: 由 LoadData 调入的详细数据；
nRec: 指定记录序号；
pnStep: 指定工步序号的返回地址。

【返回值】 >=0 成功；<0 失败（入口参数无效）。

9. “循环”序号与“记录”序号之间的逻辑关系

“循环”序号与“记录”序号之间的逻辑关系，可以先通过“循环”序=>“工步”序号，再通过“工步”序号=>“记录”序号辗转得到（结果一定是相同的）。但如果您不关心“工步”数据，辗转的方法就显得有点繁琐。这几个函数提供了“循环”序号=>“记录”序号直达的访问方式。

1) int __stdcall GetFirstRecOfCycle (HANDLE hDataObj, int nCycle)

【描述】 获取循环第一条记录。

【参数】 hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】 >=0 成功；<0 失败（入口参数无效）。

2) int __stdcall GetLastRecOfCycle (HANDLE hDataObj, int nCycle)

【描述】 获取循环最后一条记录。

【参数】 hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】 >=0 成功；<0 失败（入口参数无效）。

如果您还需要将充电和放电数据分开处理，可以用以下几个函数：

3) int __stdcall GetFirstChargeRecOfCycle (HANDLE hDataObj, int nCycle)

【描述】 获取循环第一条充电记录。

【参数】 hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】 >=0 成功；<0 失败（入口参数无效，或该循环没有充电数据）。

4) int __stdcall GetLastChargeRecOfCycle (HANDLE hDataObj, int nCycle)

【描述】 获取循环最后一条充电记录。

【参数】 hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】>=0 成功；<0 失败（入口参数无效，或该循环没有充电数据）。

5) `int __stdcall GetFirstDischRecOfCycle (HANDLE hDataObj, int nCycle)`

【描述】获取循环第一条放电记录。

【参数】hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】>=0 成功；<0 失败（入口参数无效，或该循环没有放电数据）。

6) `int __stdcall GetLastDischRecOfCycle (HANDLE hDataObj, int nCycle)`

【描述】获取循环最后一条放电记录。

【参数】hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】>=0 成功；<0 失败（入口参数无效，或该循环没有放电数据）。

10. “工步”序号与“记录”序号之间的逻辑关系

1) `int __stdcall GetFirstRecOfStep (HANDLE hDataObj, int nStep)`

【描述】获取工步第一条记录。

【参数】hDataObj: 由 LoadData 调入的详细数据；
nStep: 指定工步序号。

【返回值】>=0 成功；<0 失败（入口参数无效，或该工步没有记录数据）。

2) `int __stdcall GetLastRecOfStep (HANDLE hDataObj, int nStep)`

【描述】获取工步最后一条记录。

【参数】hDataObj: 由 LoadData 调入的详细数据；
nStep: 指定工步序号。

【返回值】>=0 成功；<0 失败（入口参数无效，或该工步没有记录数据）。

11. “循环”序号与“工步”序号之间的逻辑关系

1) `int __stdcall GetFirstStepOfCycle (HANDLE hDataObj, int nCycle)`

【描述】获取循环第一个测试工步。

【参数】hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】>=0 成功；<0 失败（入口参数无效）。

2) `int __stdcall GetLastStepOfCycle (HANDLE hDataObj, int nCycle)`

【描述】获取循环最后一个测试工步。

【参数】hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】>=0 成功；<0 失败（入口参数无效）。

如果您需要将充电和放电数据分开处理，可以用以下几个函数：

3) `int __stdcall GetFirstChargeStepOfCycle (HANDLE hDataObj, int nCycle)`

【描述】获取循环第一个充电测试工步。

【参数】hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】>=0 成功；<0 失败（入口参数无效，或该循环没有充电数据）。

4) `int __stdcall GetLastChargeStepOfCycle (HANDLE hDataObj, int nCycle)`

【描述】获取循环最后一个充电测试工步。

【参数】hDataObj: 由 LoadData 调入的详细数据；
nCycle: 指定循环序号。

【返回值】>=0 成功；<0 失败（入口参数无效，或该循环没有充电数据）。

5) `int __stdcall GetFirstDischStepOfCycle (HANDLE hDataObj, int nCycle)`

【描述】获取循环第一个放电测试工步。

【参数】hDataObj: 由 LoadData 调入的详细数据;
nCycle: 指定循环序号。

【返回值】>=0 成功; <0 失败 (入口参数无效, 或该循环没有放电数据)。

6) `int __stdcall GetLastDischStepOfCycle (HANDLE hDataObj, int nCycle)`

【描述】获取循环最后一个放电测试工步。

【参数】hDataObj: 由 LoadData 调入的详细数据;
nCycle: 指定循环序号。

【返回值】>=0 成功; <0 失败 (入口参数无效, 或该循环没有放电数据)。

12. 获取测试工步流程信息

1) `int __stdcall GetNumOfProcedure (HANDLE hDataObj)`

【描述】获取测试工步流程的个数 (一次 “工步参数重置” 操作, 会增加一个工步流程)。

【参数】hDataObj: 由 LoadData 调入的详细数据。

【返回值】>0 成功, 即返回个数; <=0 失败 (其中<0 表示入口参数无效)。

2) `bool __stdcall GetProcedureName (HANDLE hDataObj, VARIANT& vName, int nIndex=-1, long*pnHappenTime=nullptr)`

【描述】获取工步流程名 (字符串) 及启动时间。

【参数】hDataObj: 由 LoadData 调入的详细数据;

VName: 获取的工步流程名;

nIndex: 指定工步流程的序号;

pnHappenTime: 获取的工步流程对应的启动时间 (可选; 格式与 GetStartTime 相同)。

【返回值】false, 失败; true, 成功。

3.5 读取本地电脑当前连接的通道信息

1. 获取通道数据的路径

1) `bool __stdcall GetChlDataFullPath (BYTE cBoxNo, BYTE cChl, VARIANT& vPath)`

【描述】通过箱号和通道号得到通道数据的全路径。

【参数】cBoxNo: 指定箱号;

cChl: 指定通道号;

vPath: 获取指定箱号和通道号的通道数据的全路径。

【返回值】false, 失败; true, 成功。

2. 调入和释放通道快照

1) `HANDLE __stdcall LoadChlSnapshot (UINT* pnChlCount, int nDirection=CHLSS_BaseOnData)`

【描述】调入通道快照信息。通道快照信息就是通道数据在某一时刻的状况, 它是指向保存在存储设备中的通道数据的引用标记或指针。只有成功调入, 且未释放 (FreeChlSnapshot), 才能访问通道快照信息。通道快照信息访问完毕, 必须调用 FreeChlSnapshot 释放内存。

【参数】pnChlCount: 返回通道总数;

nDirection: 指定通道快照方式, 其取值参见头文件 CexConst.h。

【返回值】返回非 0, 成功; false, 失败。

2) void __stdcall FreeChlSnapshot (HANDLE hChlSnapshot)

【描述】释放由 LoadChlSnapshot 调入内存中的通道快照信息。

【参数】hChlSnapshot: 通道快照信息对象。

3. 从通道快照中获取通道信息

1) bool __stdcall ChannelFromChlSnapshot (HANDLE hChlSnapshot, int nIndex, BYTE& cBoxNo, BYTE &cChl)

【描述】从通道快照中获取箱号和通道号。

【参数】hChlSnapshot: 由 LoadChlSnapshot 调入的通道快照信息的对象；

nIndex: 指定通道索引；

cBoxNo: 返回的箱号；

cChl: 返回通道号。

【返回值】true, 成功; false, 失败。

第四章 典型的调用方法

(一) 读取数据

1. 调用 CheckDllVersion() 检查 DLL 的版本……
2. 调用 LoadData(), 调入数据……
3. 访问数据, 例如 GetRows()/GetRows2()、GetDataEx()/GetDataAsFloat()、GetStartRecFromCycle()、GetEndRecFromCycle()、……
4. 调用 FreeData() 释放内存中的数据。

(二) 枚举通道, 并读取数据

1. 调用 CheckDllVersion() 检查 DLL 的版本……
2. 调用 LoadChlSnapshot() 获取通道快照……
3. 循环开始: 枚举通道……
4. 调用 ChannelFromChlSnapshot() 获取箱号和通道……
5. 调用 GetChlDataFullPath() 获取该通道对应的数据文件的全路径名……
6. 如上节: 调用 LoadData()/…/FreeData() 读取数据……
7. 循环结束 (即完成枚举)
8. 调用 FreeChlSnapshot() 释放快照内存。

<完>