声明：

LaserLib 应用程序 API 帮助手册

Version 2.1.0.11

2021.12.10

目 录

A、 LaserLib API 简介 ---------- P6
B、 LaserLib API 调用顺序 ---------- P7
C、 LaserLib API 接口说明
1、库的基本信息
1.1 读取库版本号 GetAPILibVersion ---------- P8
1.2 读取 API 的编译信息 GetAPILibCompileInfo ----------P8
1.3 获取本库使用语言 GetLanguge ---------- P8
1.4 设置本库使用语言 SetLanguge ---------- P8
2、库的安装及卸载
2.1 安装 LaserLib 库,传递调用本库的宿主进程句柄 InitLib ---------- P9
2.2 卸载 LaserLib 库 UnInitLib ---------- P9
3、系统回调
3.1 系统进度条事件回调 ProgressCallBack ---------- P10
3.2 系统消息事件回调 SysMessageCallBack ---------- P10
4、通讯端口初始化及卸载
4.1 加载通讯端口列表 GetComPortList ----------P11
4.2 初始化并尝试打开串口 InitComPort ---------- P11
4.3 卸载并尝试关闭串口 UnInitComPort ---------- P11
5、系统全局信息
5.1 显示关于信息窗口 ShowAboutWindow ---------- P12
5.2 关闭已经打开的关于信息窗口 CloseAboutWindow ---------- P12
5.3 在关于窗口中显示加载信息 ShowLoaddingInfo ---------- P12
5.4 返回 Json 格式的关于信息 GetLaserLibInfo ---------- P12
5.5 设置厂商名称 SetFactoryType---------- P12
5.6 设置数据传输超时时间 SetTransTimeOutInterval ---------- P12
5.7 开启或关闭详细日志输出 OpenDetailedLog---------- P12
5.8 是否启用调试日志输出 DebugLogger---------- P13
6、寄存器操作
6.1 保存系统参数到寄存器 WriteSysParamToCard ---------- P14
6.2 从寄存器中读取系统参数 ReadSysParamFromCard ---------- P14
6.3 保存系统参数到服务器 SaveMainBoardParamsToServer ---------- P14
6.4 从服务器中读取系统参数 ReadMainBoardParamsFromServer ---------- P14
6.5 保存用户参数到寄存器 WriteUserParamToCard---------- P14
6.6 从寄存器中读取用户参数 ReadUserParamFromCard---------- P15
6.7 保存上位机参数到寄存器 WriteComputerParamToCard---------- P15
6.8 从寄存器中读取上位机参数 ReadComputerParamFromCard---------- P15

6.9 修改厂家密码（系统参数密码）ChangeFactoryPassword---------- P15
6.10 验证厂家密码（系统参数密码）CheckFactoryPassword---------- P16

7、硬件信息
7.1 读取本机的外网 IP 地址 GetClientAddr ---------- P17
7.2 读取电脑硬件序列号 GetDeviceID ---------- P17
7.3 读取加密锁信息 GetHardwareKeyInfo ---------- P17
7.4 读取加密锁序列号 GetHardwareKeyID ---------- P17
7.5 读取板卡序列号 GetMainCardID ---------- P17
7.6 读取板卡注册状态 GetMainCardRegState---------- P17
7.7 使用注册码注册板卡 RegisterMainCard---------- P17
7.8 申请临时许可证 CreateLicenseFile---------- P18
7.9 查询当前联机的板卡及绑定机器更多详细数据 GetMainCardInfo ---------- P18
7.10 读取板卡固件版本号 GetMainHardVersion ---------- P18
7.11 获取硬件身份识别码（SN） GetHardwareIdentID ---------- P18

8、板卡激活及邮箱注册
8.1 给指定邮箱发送激活认证码(邮箱认证) SendAuthenticationEmail---------- P20
8.2 激活板卡及机器 ActivationMainCard---------- P20
8.3 重新激活板卡及机器 ActivationMainCardEx ---------- P20

9、在线升级
9.1 查询新版本 CheckVersionUpdate ---------- P22
9.2 更新版本 StartVersionUpdate ---------- P23
9.3 取消在线更新 AbortVersionUpdate ---------- P23
9.4 弹出软件升级窗口 StartSoftUpdateWizard ---------- P23
9.5 弹出固件升级窗口 StartFirmwareUpdateWizard ---------- P23

10、设备基础控制
10.1 回机械原点 LPenMoveToOriginalPoint ---------- P24
10.2 Z 轴回机械原点 LPenMoveToOriginalPointZ---------- P24
10.3 快速移动到指定坐标点 LPenQuickMoveTo ---------- P24
10.4 开始加工 StartMachining ---------- P24
10.5 暂停继续加工 PauseContinueMachining ---------- P24
10.6 停止加工 StopMachining ---------- P24
10.7 加载卸载电机 ControlMotor ---------- P25
10.8 激光点射测试 TestLaserLight ---------- P25
10.9 载入断点续传数据 LoadBreakPiontData---------- P25
10.10 启动超时检测(按键按下后) CheckMoveLaserMotors ---------- P25
10.11 开始点动、连动模式移动激光头(按键弹起后) StartMoveLaserMotors ---------- P25
10.12 载入加工数据 JSON 文件 LoadDataFromFile ---------- P26
10.13 载入加工数据 JSON 流 ImportData ---------- P26
10.14 导出加工数据 JSON 文件 ExportData ---------- P26
10.15 获取设备当前运行信息 GetDeviceWorkState ---------- P26

11、控制 CorelDRAW
11.1 导出到文件 ExportsFileFromCDR ---------- P27
11.2 获取正在运行的 CorelDRW.exe 的主窗口句柄 GetCDRMainFormHWND ---------- P27

12、其它 API
12.1 毫秒格式化为字符串 API_FormatMillisecondString ---------- P29
12.2 获取文件版本号 API_GetFileVersionStr---------- P29
12.3 设置程序自动运行 API_SetAutoStart---------- P29
12.4 检测某个端口是否被占用 API_PortTCPIsOpen---------- P29
12.5 获取 CPU 版本信息 API_GetProcessorVersion----------P29
12.6 获取物理内存大小 API_GetPhysicsMemoryTotalSize----------P29
12.7 获取进程内存使用量 API_GetMemoryUsed---------- P29
12.8 获取进程线程数量 API_GetThreadCount---------- P30
12.9 检测文件是否是 64 位的 API_IsCompiledAsX64---------- P30
12.10 格式化字节数 API_FormatByteNumber---------- P30
12.11 获取字符串 MD5 的值 API_GetStringMD5----------P30
12.12 十进制转换为二进制 API_IntToBin ---------- P30
12.13 二进制转换为十进制 API_ BintoInt ---------- P30
12.14 字符串转换为十六进制 API_StringToHex---------- P31
12.15 十六进制转换为字符串 API_HexToString---------- P31
12.16 十六进制字符串转换为二进制字符串 API_HexToBinString---------- P31
12.17 把二进制串转换成十六进制串 API_BinStringToHex---------- P31
12.18 生成随机字符串（包含数字、字母）API_GenerateRandomString---------- P31
12.19 获取设备名称 API_GetDeviceModel---------- P31
12.20 获取设备唯一 ID API_GetDeviceId---------- P32
12.21 获取操作系统语言信息 API_GetWindowsLanguage ---------- P32
12.22 获取文件大小 API_FileSizeByName---------- ---------- P32
12.23 获取文件的时间属性 API_GetFileTimeByName------ ---------- P32

13、经典排序算法
13.1 冒泡排序 API_BubbleSort---------- P33
13.2 摇动排序 API_ShakerSort---------- P33
13.3 梳子排序 API_CombSort---------- P33
13.4 选择排序 API_SelectionSort---------- P33
13.5 标准插入排序 API_InsertionSortStd---------- P33
13.6 优化的插入排序 API_InsertionSort---------- P33
13.7 希尔排序 API_ShellSort---------- P33
13.8 标准归并排序 API_MergeSortStd---------- P33
13.9 优化的归并排序 API_MergeSort---------- P33
13.10 标准快速排序 API_QuickSortStd---------- P33
13.11 无递归的快速排序 API_QuickSortNoRecurse---------- P33
13.12 随机的快速排序 API_QuickSortRandom---------- P34
13.13 中间值的快速排序 API_QuickSortMedian---------- P34
13.14 优化的插入快速排序 API_QuickSort---------- P34

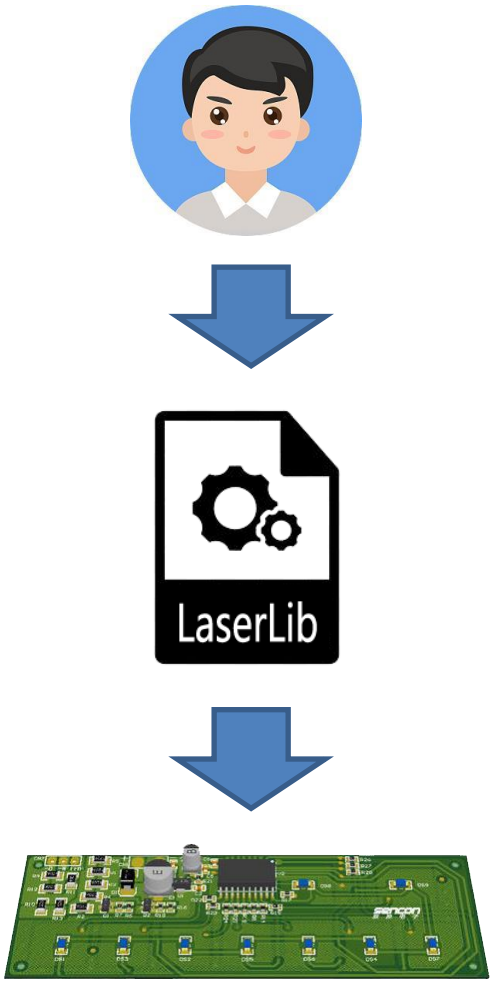
13.15 堆排序 API_HeapSort---------- P34
附件 1 ---------- P35
附件 2 ---------- P39
附件 3 ---------- P41
附件 4 ---------- P43
附件 5 ---------- P44

LaserLib API 简介

LaserLib API 库是一个中间件，是介于用户程序和硬件之间联系沟通的一座桥梁，库内封装大量底层处理过程和业务逻辑，基于本库接口开发应用程序的开发人员无需关心处理繁琐的底层即可快速开发出高效稳定的终端用户程序。

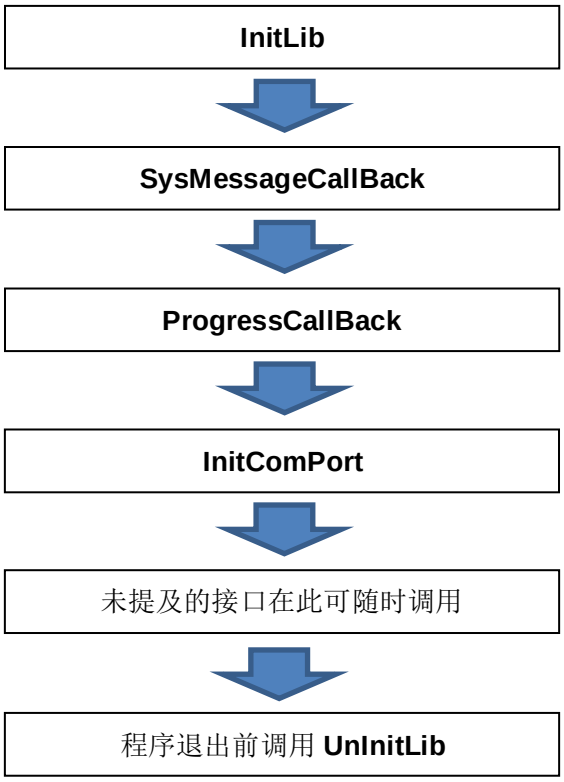
DLL是大多数编程序语言所支持的一种开发方式，DLL包含所需要的所有函数用户只需在程序中进行声明并调用即可，在运行时必须保证该DLL文件在系统目录下或与EXE文件在同一目录下。

通过阅读本手册可以快速掌握LaserLib API 库中接口函数的功能和用法。



LaserLib API 调用顺序

本库中 API 接口调用是有顺序的，开发人员在软件中使用时应遵循如下规则：



LaserLib API

下面列出了 LaserLib.Dll（以下简称本库）支持的应用程序 API 及详细说明。

1、 库的基本信息

1.1 读取库版本号

接口：**GetAPILibVersion**
原型：function GetAPILibVersion: PChar; stdcall;
参数：无
返回：Pchar
备注：无

1.2 读取 API 的编译信息

接口：**GetAPILibCompileInfo**
原型：function GetAPILibCompileInfo: PChar; stdcall;//获取 API 的编译信息
参数：无
返回：字符串形式的编译信息
备注：

1.3 获取本库使用语言

接口：**GetLanguge**
原型：function GetLanguge: Integer; stdcall; //返回本库使用语言 0 英文 1 中文
参数：无
返回：0 英文 1 中文(默认中文)

1.4 设置本库使用语言

接口：**SetLanguge**
原型：function SetLanguge(const Languge: Byte = 0): Integer; stdcall; //设置本库使用语言 0 英文 1 中文
参数：Languge 取值范围 0 英文 1 中文
返回：0 英文 1 中文
备注：设置使用语言直接影响到库返回的文字信息，如软件或固件返回的更新日志内容
建议上位机 EXE 在检测新版本（9.1）前根据当前 EXE 的语言版本作同步设置。

2、 库的安装及卸载

2.1 安装 LaserLib 库，传递调用本库的宿主进程句柄

接口：**InitLib**
原型：procedure InitLib(const Hwnd: THandle); stdcall;
参数：Hwnd 宿主进程主窗口句柄，int 型
返回：无
备注：宿主进程初始化时调用

2.2 卸载 LaserLib 库

接口：**UnInitLib**
原型：procedure UnInitLib; stdcall;
参数：无
返回：无
备注：宿主进程退出时调用

3、 系统回调

3.1 系统进度条事件回调

接口：**ProgressCallBack**
原型：procedure ProgressCallBack(CB: TProgressCallBack); cdecl;
参数：CB 回调接口，TProgressCallBack 型
返回：无
备注：宿主进程初始化时调用

TprogressCallBack 类原型：
TProgressCallBack = procedure(const Postion: Integer; const TotalCount: Integer) of object; cdecl;

Pascal 代码示例，其它语言类同：
在应用程序中创建自定义过程，参数同 TprogressCallBack：

```
Procedure MyProgressCallBack (const Postion: Integer; const TotalCount: Integer); cdecl;
Begin
   ProgressBar1.Position := Postion;//实时进度值

   ProgressBar1.Max := TotalCount;//进度总量

End;
```

宿主进程初始化时调用：

```
ProgressCallBack(MyProgressCallBack);
```

3.2 系统消息事件回调

接口：**SysMessageCallBack**
原型：procedure SysMessageCallBack (CB: TSysMessageCallBack); cdecl;
参数：CB 回调接口，TSysMessageCallBack 型
返回：无
备注：宿主进程初始化时调用

TSysMessageCallBack 类声明：

TSysMessageCallBack = procedure(const SysMsgIndex: Integer; const SysMsgCode: Integer; const SysEventData: PChar) of object; cdecl;

参数：
SysMsgIndex 消息索引号，int 型，库调试时使用
SysMsgCode 消息编码，int 型，是库在执行业务逻辑时反馈给宿主程序的工作状态标识，具体定义见附件一：
SysEventData 系统返回信息，以字符串形式返回，在不同情况下会有不同格式，具体标准会在相关接口函数中给出说明。

4、 通讯端口初始化及卸载

4.1 加载通讯端口列表

接口：**GetComPortList**
原型：function GetComPortList: PChar; stdcall;
参数：无
返回：失败返回空字符；成功返回以;分隔的列表项目
备注：若本机连接两台机器，则获取的列表返回为 JAT2020 LASER COM (COM3); JAT2020 LASER COM (COM12)
开发人员需要自行解析并提取出串口编号(在本例中为 3 和 12)供 InitComPort (4.2)作为参数使用。

4.2 初始化并尝试打开串口

接口：**InitComPort**
原型：function InitComPort(const cComPort: Integer): Integer; stdcall;
参数：cComPort 系统串口索引，int 型，**1 表示 com1，2 表示 com2 .....类推**
返回：Integer 成功返回 0，失败返回-1
备注： 1、此处返回 0 并不表示串口被正确打开，串口状态以回调函数中为准；
2、当板卡与加密锁绑定信息不一致时，应该重新绑定，不然无法正常使用(临时许可证授权除外)；
3、若板卡与加密锁绑定信息不一致，会在通讯建立连接（联机）之后收到回调 CardDongleBindinvalid 消息，开发人员应该在此消息过程中弹窗询问用户“加密锁和板卡绑定信息不匹配，需要重新绑定吗？”，之后根据用户选择是或否，进行下一步重新绑定或取消操作；
4、选择重新绑定时，应该调用 MainCardBindDongle (5.27)，所有参数可以为空字符；
5、绑定结果会在回调消息中返回；
6、一只加密锁最多重复绑定 3 次，超限后无法再次绑定。<**2~6 板卡与加密锁绑定功能暂停使用**>

4.3 卸载并尝试关闭串口

接口：**UnInitComPort**
原型：function UnInitComPort: Integer; stdcall;
参数：无
返回：Integer 成功返回 0，失败返回-1
备注：此处返回 0 并不表示串口被正确关闭，串口状态以回调函数中为准

5、 系统全局信息

5.1 显示关于信息窗口

接口：**ShowAboutWindow**
原型：function ShowAboutWindow(const Interval: Integer = 0; const Modal: Boolean = True): Integer; stdcall;
参数：Interval 自动关闭窗口时间，单位秒，0 表示不自动关闭；Modal 为 TRUE，显示为模态窗口。
返回：关于信息窗口句柄
备注：默认不自动关闭、模态窗口

5.2 关闭已经打开的关于信息窗口

接口：**CloseAboutWindow**
原型：procedure CloseAboutWindow; stdcall;//关闭关于信息窗口
参数：无
返回：无

5.3 在关于窗口中显示加载信息

接口：**ShowLoaddingInfo**
原型：procedure ShowLoaddingInfo(const Info: PChar); stdcall;
参数：Info 初始化进度时需要显示的字符
备注：可多次调用

5.4 返回 Json 格式的关于信息

接口：**GetLaserLibInfo**
原型：function GetLaserLibInfo: PChar; stdcall;//
参数：无
返回：返回内容为 JSON 格式字符串，内容详见**附件四**
备注：自由设计关于信息窗口时，可调用本接口解析并显示返回的文字内容。

5.5 设置厂商名称

接口：**SetFactoryType**
原型：procedure SetFactoryType (const TypeStr: PChar); stdcall;
参数：TypeStr 上位机 EXE 标识英文名称，用于定位升级数据包 URL 地址。
备注：在初始化或每次**查询是否有新版本（9.1）**前调用 **SetFactoryType** 设置名称

5.6 设置数据传输超时时间

接口：**SetTransTimeOutInterval**
原型：procedure SetTransTimeOutInterval(const cInterval: Integer = 20); stdcall;
参数：cInterval 超时时间长度，单位秒，int 型，20 表示 20 秒超时
返回：无
备注：设定即时生效，如不调用则系统默认 20 秒超时检测

5.7 开启或关闭详细日志输出

接口：**OpenDetailedLog** 非调试勿启用此功能
原型：procedure OpenDetailedLog(const IsOut: Boolean = False); stdcall;
参数：IsOut 输出开关 True 为开启，False 为关闭
备注：1、默认关闭详细日志，可节省资源提高效率；
2、开启详细日志输出后仅对本次启动有效，下次启动默认关闭；
3、开关详细日志会直接影响到 **5.6 DebugLogger** 输出的内容。

5.8 是否启用调试日志输出(到文件)

接口：**DebugLogger** 非调试勿启用此功能

原型：function DebugLogger(const OutEnable: Boolean): Boolean; stdcall;

参数：OutEnable:True 启用 False 禁用 默认禁用

备注：返回值同 OutEnable；启用后日志文件路径为<. \Log\Debug\ DebugLog.log >；

6、 寄存器操作

6.1 保存系统参数到寄存器

接口：**WriteSysParamToCard**

原型：procedure WriteSysParamToCard(//
const cAdd: PChar; //地址 0-255 可同时指定多个地址，地址功能定义参考**附表**
const cData: PChar//数据 0-65535 可同时指定多个对应值
); stdcall;

参数：见上返回：-1 表示保存失败

备注：各地址号(cAdd)以,分隔开，如：13,14,15,16,17...
数据(cData)同样以,分隔开 示例同上
数据和地址号的顺序要严格保持一致，避免数据错乱

6.2 从寄存器中读取系统参数

接口：**ReadSysParamFromCard**

原型：procedure ReadSysParamFromCard (//
const cAdd: Pchar //地址 0-255 可同时指定多个地址，地址功能定义参考**附表**
); stdcall;

参数：cAdd 如取值 '$FFFF'，表示读取全部地址

返回：-1 表示保存失败，读取成功后在 SysEventCallBack 回调中返回数据，
格式为：**地址 1,数据 1;地址 2,数据 2;...地址 n,数据 n;**

备注：1、各地址号(cAdd)以,分隔开，如：13,14,15,16,17...
2、系统寄存器的地址和内容见附件二。

6.3 保存系统参数到服务器

接口：**SaveMainBoardParamsToServer**

原型：procedure SaveMainBoardParamsToServer; stdcall;

参数：无

返回：无

备注：保存到云端的参数必须是当前板卡已写入的信息

6.4 从服务器读取系统参数

接口：**ReadMainBoardParamsFromServer**

原型：procedure ReadMainBoardParamsFromServer; stdcall;

参数：无

返回：在系统回调 ReadParamsFromServerOK 返回读取的参数信息,类似 ReadSysParamFromCardOK

备注：无

6.5 保存用户参数到寄存器

接口：**WriteUserParamToCard**

原型：procedure WriteUserParamToCard (//
const cAdd: PChar; //地址 0-255 可同时指定多个地址，地址功能定义参考**附表**
const cData: PChar//数据 0-65535 可同时指定多个对应值
); stdcall;

参数：见上返回：-1 表示保存失败

备注：各地址号(cAdd)以,分隔开，如：13,14,15,16,17...
数据(cData)同样以,分隔开 示例同上

数据和地址号的顺序要严格保持一致，避免数据错乱

6.6 从寄存器中读取用户参数

接口：**ReadUserParamFromCard**

原型：procedure ReadUserParamFromCard (//
const cAdd: Pchar //地址 0-255 可同时指定多个地址，地址功能定义参考**附表**
); stdcall;

参数：cAdd 如取值 '$FFFF'，表示读取全部地址

返回：-1 表示保存失败，读取成功后在 SysEventCallBack 回调中返回数据，
格式为：**地址 1,数据 1;地址 2,数据 2;...地址 n,数据 n;**

备注：

1、各地址号(cAdd)以,分隔开，如：13,14,15,16,17...

2、用户寄存器的地址和内容见**附件三。**

6.7 保存上位机参数到寄存器

接口：**WriteComputerParamToCard**

原型：procedure WriteUserParamToCard (//
const cAdd: PChar; //地址 0-255 可同时指定多个地址，地址功能定义参考**附表**
const cData: PChar//数据 0-65535 可同时指定多个对应值
); stdcall;

参数：见上返回：-1 表示保存失败

备注：各地址号(cAdd)以,分隔开，如：13,14,15,16,17...
数据(cData)同样以,分隔开 示例同上
数据和地址号的顺序要严格保持一致，避免数据错乱

6.8 从寄存器中读取上位机参数

接口：**ReadComputerParamFromCard**

原型：procedure ReadComputerParamFromCard (//
const cAdd: Pchar //地址 0-255 可同时指定多个地址，地址功能定义参考**附表**
); stdcall;

参数：cAdd 如取值 '$FFFF'，表示读取全部地址

返回：-1 表示保存失败，读取成功后在 SysEventCallBack 回调中返回数据，
格式为：**地址 1,数据 1;地址 2,数据 2;...地址 n,数据 n;**

备注：

1、 地址号(cAdd)以,分隔开，如：13,14,15,16,17...

2、 返回字符串格式：地址与数据以半角,分隔为一组，组之间以半角; 分隔

6.9 修改厂家密码（系统参数密码）

接口：**ChangeFactoryPassword**

原型：function ChangeFactoryPassword(
const OldPassword: PChar; //旧密码
const NewPassword: PChar//新密码
): Boolean;; stdcall;

返回：调用成功返回 True，失败返回 False

备注：本操作验证密码是否修改成功在回调消息中返回：

ChangeFactoryPasswordOK、FactoryPasswordValid、ChangeFactoryPasswordError、FactoryPasswordInValid

6.10 验证厂家密码（系统参数密码）

接口：**CheckFactoryPassword**

原型：function CheckFactoryPassword (
const Password: PChar; //待验证的密码
var InputErr: Integer//密码输错次数
): Boolean;; stdcall;

返回：调用成功返回 True，失败返回 False

备注：InputErr 为返回的错误密码次数，连续 3 次输错，将不再验证密码（再次输入的密码，无论正确与否一律返回 False，除非重启上位机软件才可再次验证）

7、 硬件信息

7.1 获取本机的外网 IP 地址

接口：**GetClientAddr**
原型：function GetClientAddr(const ReLoad: Boolean): PChar; stdcall;
参数：ReLoad = True,表示强制重新获取，此时函数返回值为空，需要从回调中返回（异步）；
ReLoad = False,表示直接使用,此时函数返回值为上一次获取到的信息（同步，回调中也有返回）
备注：无

7.2 获取电脑硬件序列号

接口：**GetDeviceID**
原型：function GetDeviceID(const ReLoad: Boolean): PChar; stdcall;
参数：ReLoad = True,表示强制重新获取，此时函数返回值为空，需要从回调中返回（异步）；
ReLoad = False,表示直接使用,此时函数返回值为上一次获取到的信息（同步，回调中也有返回）
备注：无

7.3 读取加密锁信息

接口：**GetHardwareKeyInfo**
原型：function GetHardwareKeyInfo: PChar; stdcall;
参数：无
返回：返回字符串以;间隔，顺序为：
经销商;厂址;区域代码;售后电话;QQ 号码;微信号码;电子邮箱;网址;ApplicationID

7.4 读取 USB 加密锁 ID

接口：**GetHardwareKeyID**
原型：function GetHardwareKeyID: PChar; stdcall;
参数：
返回：-1 表示调用失败，>-1 表示调用成功
备注：

7.5 读取板卡序列号

接口：**GetMainCardID**
原型：function GetMainCardID: PChar; stdcall;
参数：
返回：直接返回板卡 ID aaaa 和注册 ID bbbb，格式为：aaaa;bbbb
备注：aaaa 表示的 ID 用于服务端入库，bbbb 表示的注册 ID 用于注册板卡

7.6 读取板卡注册状态

接口：**GetMainCardRegState**
原型：procedure GetMainCardRegState; stdcall;
参数：
返回：回调中返回 66 注册成功 或 67 注册失败
备注：

7.7 使用注册码注册板卡

接口：RegisterMainCard
原型：function RegisterMainCard(const RegCode: PChar): PChar; stdcall;
参数：RegCode 板卡的注册码
备注：1、若注册码成功写入板卡会返回 RegCode，写入失败返回空字符；
2、仅当板卡未注册时需要调用此函数进行注册(通常情况为板卡在出厂前已提前注册好)；

3、注册成功的板卡再次调用此函数会在回调中返回 MainCardRegisterOK 消息；
4、注册后板卡重新上电后会在回调中显示注册状态；
5、板卡注册和激活的概念用途不同：
板卡注册由板卡厂家在板卡出厂前完成；
激活由终端用户完成，且必须在首次使用前在线完成激活操作，用于记录板卡的启用日期等信息；
临时激活为雕刻机生产厂家在板卡未激活的情况下调试机器使用，必须使用专用管理锁在线**申请临时许可证(7.8)**。

7.8 申请临时许可证

接口：**CreateLicenseFile**
原型：function CreateLicenseFile(const LicFlag: PChar): Boolean; stdcall;
参数：LicFlag = {FFFD38EB-DC3A-45A8-A06D-B10671CF18B3}
返回：此函数会自动完成在线申请许可，本地许可证文件创建成功后会返回 True
备注：1、使用临时许可证，可在板卡未激活前进入调试状态，供机器生产厂家测试整机使用；
2、临时许可证仅可在板卡未激活前申请;
3、许可证绑定电脑，超级管理员加密锁才可申请成功。

7.9 查询当前联机的板卡及绑定机器更多详细数据

接口：**GetMainCardInfo**
原型：function GetMainCardInfo: PChar; stdcall;
参数：无
备注：激活信息返回字符串以;间隔，请自行拆解内容。内容顺序为：

0:板卡 ID;1:板卡注册日期;2:板卡激活日期;

3:激活绑定的邮箱;

4:板卡绑定的加密锁 ID;5:锁注册日期;6:锁激活日期;7:锁绑定次数;

8:当前加密锁 ID;9:锁注册日期;10:锁激活日期;11:锁绑定次数; 8~11 表示绑定锁 ID 与当前锁 ID 不一致才显示（暂未启用绑定功能）

12:机器注册日期;13:机器激活日期;14:机器维护次数（更换灯管清零次数）（暂未启用）

返回字串示例：

5711660732374742O5D9FF318F22AC7A;2021-01-19 18:19:56;2021-04-19 09:37:02;xxxxx@qq.com;1242510512650915;2021-03-30 22:51:21;2021-04-22 12:29:55;2;1242510512650915;2021-03-30 22:51:21;2021-04-22 12:29:55;2;;;0;;;;;

7.10 读取固件版本号

接口：**GetMainHardVersion**
原型：function GetMainHardVersion: PChar; stdcall;
返回：版本号格式 yymmddhh

7.11 获取硬件身份识别码（SN）

接口：**GetHardwareIdentID**
原型：function GetHardwareIdentID: PChar; stdcall;
返回：为 32 个半角字符长度的字串，该识别码在激活后生成，为关联查询客户硬件信息使用。

7.12 激光管使用时间清零 （暂未启用）

接口：**LaserTubeZeroClearing**
原型：procedure LaserTubeZeroClearing(const PassWord: PChar); stdcall;
参数：PassWord 清零密码，清零功能需要配合管理员专用加密锁才可执行成功
返回：无

7.13 板卡绑定加密锁 （暂未启用）

接口：**MainCardBindDongle**
原型：function MainCardBindDongle(const Bind: PChar): Integer; stdcall;
参数：Bind = {EF45438C-F8A7-4896-BACF-74C4B493881E}
返回：此函数会自动完成在线绑定加密锁 执行成功返回 0，失败返回-1（返回值不代表绑定是否成功）
备注：详情参考 4.2 初始化并尝试打开串口 ；

8、板卡激活及邮箱注册

激活板卡步骤：

1、 给指定邮箱发送邮件获取认证码 **SendAuthenticationEmail**；

2、 把邮件中收到的认证码和姓名、邮址一起作为参数传递给 **ActivationMainCard**。

8.1 给指定邮箱发送激活认证码(邮箱认证)

接口：**SendAuthenticationEmail**

原型：function SendAuthenticationEmail(//

```
const GUID: PChar; //'{464D4EDA-2645-4427-AA92-80D53231089F}'
const Email: PChar; //邮箱地址
const Languge: Integer = -1////邮件语言 0 英文 1 中文 2 其它 -1 根据 OS 自动选择语言
): Integer; stdcall;
```

参数：GUID 为固定值；Email 填写邮件地址；Languge 默认根据 OS 自动选择邮件所用语言

返回：Integer，0 邮件发送失败，1 邮件发送成功

备注：

8.2 激活板卡及机器

接口：**ActivationMainCard**

原型：function ActivationMainCard(//激活板卡及机器

```
const vEmail: PChar; //邮箱，必填
const vActiveCode: PChar; //认证码(从指定邮箱接收到的邮件中获取)，验证是邮箱否合法，必填
const vUserName: PChar; //姓名，必填
const vTelphone: PChar; //电话
const vAddress: PChar; //地址
const vQQ: PChar; //QQ
const vWX: PChar; //WX
const Country: PChar; //国家
const Distributor: PChar; //经销商
const Trademark: PChar; //品牌
const Model: PChar; //型号
const MachineID: PChar//机器 ID
): PChar; stdcall;
```

参数：见上

返回：

1、已激活或激活成功返回板卡的注册日期和激活日期字符串，以;间隔。未激活成功返回空；

2、本接口仅适用于板卡的初次激活。

8.3 重新激活板卡及机器

接口：**ActivationMainCardEx**

原型：function ActivationMainCardEx (const Flag: PChar): Integer; stdcall;

参数：Flag = '{3567D29E-394B-4814-80C4-510331CD39CD}'

返回：0 激活成功 1 未激活过，需要进入邮箱认证激活流程（8.2 **ActivationMainCard**）

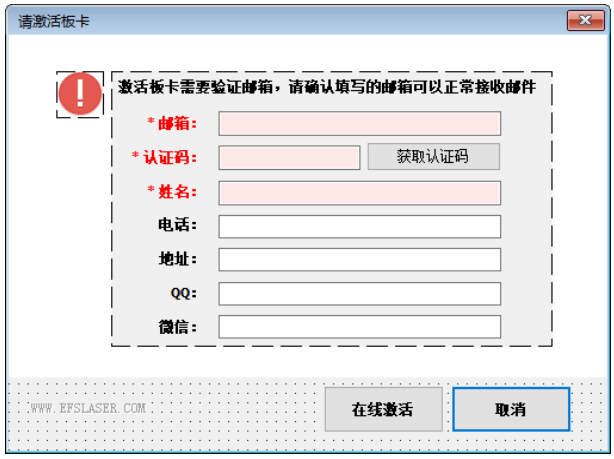
备注：

1、回调中只要收到：

MainCardRegisterError = 1008; //板卡注册失败
就要弹出板卡注册窗口，让用户输入板卡注册码

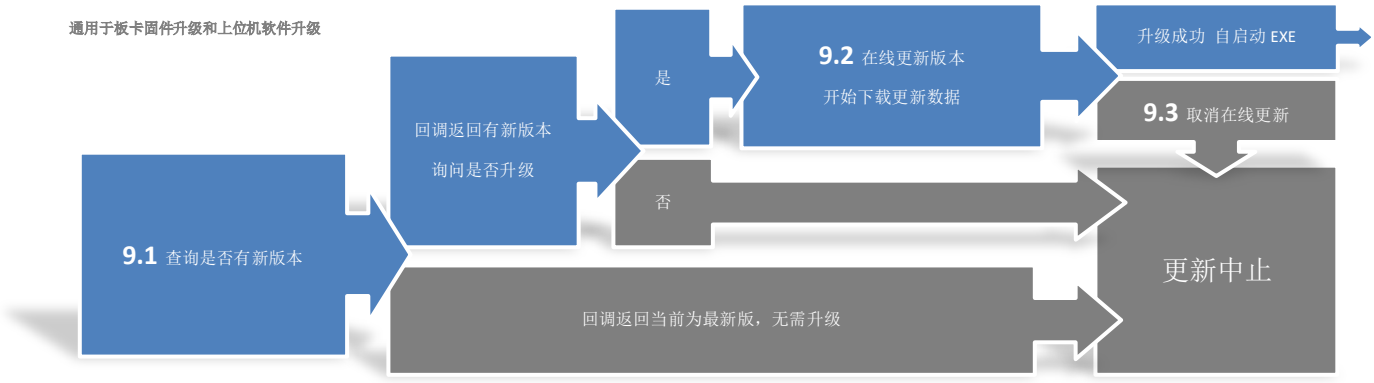
2、回调中只要收到：
MainCardIsPirate = 1009; //板卡未激活
就要调用 ActivationMainCardEx 函数；只要 ActivationMainCardEx 函数返回 1，就要弹出邮箱认证激活窗口

设计参考




```
if ActivationMainCard(//
  PChar(vEmail), //邮箱，必填
  PChar(vActiveCode), //认证码，必填
  PChar(vUserName), //姓名，必填
  //
  PChar(vTelphone), //电话
  PChar(vAddress), //地址
  PChar(vQQ), //QQ
  PChar(vWX), //WX
  //
  PChar(Country), //国家
  PChar(Distributor), //经销商
  PChar(Trademark), //品牌
  PChar(Model), //型号
  PChar(MachineID) //机器ID
) <> '' then
begin
  ActiveLabel.Caption := '激活成功...';
  ModalResult := mrOk;
end
else
begin
  ActiveLabel.Caption := '激活失败...';
end;
```

9 、在线升级



9.1 查询是否有新版本

接口：**CheckVersionUpdate**

原型： procedure CheckVersionUpdate(//查询是否有新版本
const IsFirmwareUpdate: Boolean; //是否固件更新，否为软件更新
const Flag: PChar; //标识符
const CurrentVersion: Integer; //当前版本号
const VersionNoteToJsonFile: PChar //Json 格式的更新日志文件名，值为空表示不返回也不生成文件
); stdcall;//**通过回调返回**有无新版本情况

参数：IsFirmwareUpdate 值 True 为固件，False 为上位机软件

Flag 标识符取值请参考 **9.2 在线更新软件版本**附表；

CurrentVersion 为当前版本号，格式为 yymmdd，**必须和编译日期同步**，此版本号（固件有效）用于和升级服务器记录的版本号作对比，如有更新版本，则新版本号比此版本号要大（作为整形数比较大小）。

VersionNoteToJsonFile 新版本更新日志文件名，有新版会生成此 JSON 文件，需自行解析并显示于弹窗中。

JSON 内容如下：

```
{
  "ApplicationName": "LaserController",
  "HomePage": "https://xxx.com",
  "CompanyName": "",
  "FileDescription": "",
  "FileVersion": "",
  "InternalName": "",
  "LegalCopyright": "",
  "LegalTrademarks": "",
  "OriginalFilename": "",
  "ProductName": "",
  "ProductVersion": "",
  "Comments": "",
  "ProgramID": "",
  "UpdateLog":" /*更新日志内容字段*/
新版本更新日志内容第 1 行
新版本更新日志内容第 2 行
新版本更新日志内容第 3 行
.....N 行
"
```

}

返回：在回调中返回升级状态

备注：板卡未激活不可在线升级同时在回调中返回消息

9.2 升级新版本

接口：**StartVersionUpdate**

原型：procedure StartVersionUpdate (

const IsFirmwareUpdate: Boolean; //是否固件更新，否为软件更新

const Flag: PChar; //标识符 区分升级方式

const LocalFile: PChar; //本地升级文件（**仅本地升级方式有效**）

const UpdateVersion: Integer; //要升级的版本 指定格式为 201201 则升级此版本，为 0 则升级到最新版

const BeforeRunType: Integer; //升级前运行的程序运行方式 0 运行并等待结束 1 运行后无需等待

const BeforeRunExeName: PChar; //升级前运行的程序文件名(.bat;.exe)，为空不运行

const AfterRunExeName: PChar //升级完成后运行的程序文件名(.bat;.exe)，为空不运行

); stdcall;

参数： IsFirmwareUpdate 值 True 为固件，False 为上位机软件

Flag 标识符，升级方式对应的值见下表

功能		Flag 取值	UpdateVersion
本地升级		{7ADDA8F6-69A6-410A-B17F-260733FA50D5}	LocalFile 为本地升级文件名
网络升级	内测版	{4A5F9C85-8735-414D-BCA7-E9DD111B23A8}	版本号格式为 yymmdd 如 201201 则升级此指定版本 若值为 0，则升级到最新版本
	公测版	{CC9094BA-2991-4CE2-A514-BF0EA3685D05}	

返回：在回调中返回升级状态

备注：1、通过网络升级时，**固件可指定版本，上位机软件只允许升级到最新版**；

2、板卡未激活不可在线升级同时在回调中返回消息

9.3 取消版本更新

接口：**AbortVersionUpdate**

原型：function AbortVersionUpdate: Integer; stdcall;

参数：无

返回：在**下载升级数据过程中**可随时中止，成功返回 0，失败返回-1

9.4 弹出软件升级窗口

接口：StartSoftUpdateWizard

原型：procedure StartSoftUpdateWizard; stdcall;

参数：无

9.5 弹出固件升级窗口

接口：StartFirmwareUpdateWizard

原型：procedure StartFirmwareUpdateWizard; stdcall;

参数：无

10、设备基础控制

10.1 回机械原点_XY 轴

接口：**LpenMoveToOriginalPoint**
原型：procedure LPenMoveToOriginalPoint(const cSpeed: Double); stdcall;
参数：cSpeed 速度值，Double 型，单位 mm/s
返回：无
备注：cSpeed 参数不再使用，以寄存器值为准

10.2 回机械原点_Z 轴

接口：**LPenMoveToOriginalPointZ**
原型：procedure LPenMoveToOriginalPointZ(const MoveTo: Integer); stdcall;
参数：MoveTo Z 轴复位后再移至目标坐标值 单位 μm
返回：无
备注：

10.3 快速移动到指定坐标点

接口：**LPenQuickMoveTo**
原型：procedure LPenQuickMoveTo(//
const X_MoveEnable: Boolean; //X 移动使能：True 移动 False 不移动
const X_MoveStyle: Boolean; //X 移动方式：True 绝对坐标 False 相对坐标
const X_MovePos: Integer; //X 目标坐标值 单位 μm
const Y_MoveEnable: Boolean; //Y 移动使能：True 移动 False 不移动
const Y_MoveStyle: Boolean; //Y 移动方式：True 绝对坐标 False 相对坐标
const Y_MovePos: Integer; //Y 目标坐标值 单位 μm
const Z_MoveEnable: Boolean; //Z 移动使能：True 移动 False 不移动
const Z_MoveStyle: Boolean; //Z 移动方式：True 绝对坐标 False 相对坐标
const Z_MovePos: Integer //Z 目标坐标值 单位 μm
); stdcall;
参数：见上
返回：无
备注：

10.4 开始加工

接口：**StartMachinin**
原型：procedure StartMachining(const PackIndex: LongWord = 0) stdcall; //开始加工
参数：PackIndex 数据包编号，从指定编号发数据，默认为 0 即可
返回：
备注：切割或雕刻数据加载后，执行 StartMachining 开始加工

10.5 暂停继续加工

接口：**PauseContinueMachining**
原型：function PauseContinueMachining(const cPause: Boolean): Integer; stdcall;
参数：cPause = True 为暂停加工，= False 为继续加工
返回：-1 失败 0 暂停 1 继续
备注：

10.6 停止加工

接口：**StopMachining**
原型：procedure StopMachining; stdcall;
参数：

返回：
备注：停止加工且不可继续（加工数据被清除）

10.7 加载卸载电机
接口：**ControlMotor**
原型：function ControlMotor(const cOpen: Boolean): Integer; stdcall;
参数：cOpen=True 为加载，=False 为卸载
返回：-1 失败 3 加载 4 卸载
备注：

10.8 激光点射测试
接口：**TestLaserLight**
原型：function TestLaserLight(const cOpen: Boolean): Integer; stdcall;
参数：cOpen=True 为开激光，=False 为关激光
返回：-1 失败 5 开激光 6 关激光
备注：

10.9 载入断点续传数据(暂由本库内部自行维护，外部无须调用)
接口：**LoadBreakPiontData**
原型：function LoadBreakPiontData(const IsLoad: Boolean): Boolean; stdcall;
参数：IsLoad 是否载入断点数据，布尔值，True 为载入，False 为取消续传（即取消未完成的加工）
返回：载入数据成功返回 True
备注：
1、断点数据由本库自行维护；
2、EXE 需要在回调 RequestAndContinue 事件中弹窗请求用户选择是否断点续传即可；
3、在弹窗未返回选择状态前，RequestAndContinue 事件是多次触发的，为避免多次弹窗请自行处理逻辑。

==========

10.10 启动超时检测(按键按下后)
接口：**CheckMoveLaserMotors**
原型：procedure CheckMoveLaserMotors (
const I_DelayTime: Word; //按键延迟检测时长，单位 ms
const X_MoveEnable: Boolean; //X 移动使能：True 移动 False 不移动
const X_MoveStyle: Boolean; //X 移动方式：True 绝对坐标 False 相对坐标
const X_MovePos: Integer; //X 目标坐标值
const Y_MoveEnable: Boolean; //Y 移动使能：True 移动 False 不移动
const Y_MoveStyle: Boolean; //Y 移动方式：True 绝对坐标 False 相对坐标
const Y_MovePos: Integer; //Y 目标坐标值
const Z_MoveEnable: Boolean; //Z 移动使能：True 移动 False 不移动
const Z_MoveStyle: Boolean; //Z 移动方式：True 绝对坐标 False 相对坐标
const Z_MovePos: Integer //Z 目标坐标值
); stdcall;
参数：坐标单位：μm 微米
返回：无
备注：窗口按钮按下事件中执行，在 I_DelayTime 内为短按，超出时长为长按

10.11 开始点动、连动模式移动激光头(按键弹起后)
接口：**StartMoveLaserMotors**
原型：procedure StartMoveLaserMotors; stdcall;

参数：无
返回：无
备注：窗口按钮弹起事件中执行，库内会自动处理长按短按两种情况。

10.12 载入加工数据 JSON 文件
接口：**LoadDataFromFile**
原型：function LoadDataFromFile(const FileName: PChar): Integer; stdcall;
参数：FileName 要加工的数据文件名，本库目前支持 JSON 格式（**附件五**）
返回：成功返回分包数，错误返回值<0
备注：

10.13 载入加工数据 JSON 流
接口：**ImportData**
原型：function ImportData(const InData: PByte; const Len: Integer): Integer; stdcall;
参数：InData JSON 流，Len 数组长度；本库目前支持 JSON 格式（**附件五**）
返回：Len 数组长度
备注

10.14 输出 JSON 文件
接口：**ExportData**
原型：function ExportData(const InData: PByte; const Len: Integer; const OutName: PChar): Integer; stdcall;
参数：InData JSON 流，Len 数组长度，OutName JSON 文件名全路径；本库目前支持 JSON 格式（**附件五**）
返回：返回 Len 数组长度
备注

10.15 获取设备当前运行信息
接口：**GetDeviceWorkState**
原型：procedure GetDeviceWorkState; stdcall;
备注：
1、通过回调消息返回 MachineWorking = 2039; //工作状态
2、返回字符串格式为：
指令码;包编号;运行状态;指令编号;X 坐标;Y 坐标;Z 坐标
以半角;隔断；
3、工作状态：<0 停止 1 运行 2 暂停>
4、XYZ 坐标单位：μm 微米

11、控制 CorelDRAW

11.1 导出到文件

接口：**ExportsFileFromCDR**

原型：function ExportsFileFromCDR(const FileName: PChar): Integer; stdcall;

参数：FileName 期望 CDR 输出的文件名，可自定义，为全路径。

导出的文件扩展名决定导出格式，如 D:\demo.svg，则导出格式为 SVG

支持的导出格式有.PLT;.BMP;.JPG;.JPEG;.PNG;.TTF;.PDF;.GIF;.SVG

返回：Integer 类型，具体定义如下

0:导出成功

-1:导出失败

-2:未启动 CorelDRAW

-3:不支持的导出格式

-4:图元数量为 0

-5:无活动页面

-6:导出数据时发生未知错误

11.2 获取正在运行的 CorelDRW.exe 的主窗口句柄

接口：**GetCDRMainFormHWND**

原型：function GetCDRMainFormHWND(const vExeFileNmae: PChar): Integer; stdcall;

参数：vExeFileNmae 当前正在运行的 CorelDRW.exe 全路径

返回：0 获取主窗口句柄失败 -1; EXE 文件不存在 -2; EXE 未运行

参考方案：

要实现的功能：把当前运行的 CDR 主窗口提到最前面(无论窗口是否正常显示或最小化到任务栏)

Pascal 代码示例：

```
---------------------------------------------------------------------------------------------------------------------------------------------
var
  vCDRHwnd: Integer; //CDR 主窗口句柄
begin
  vCDRHwnd :=
  GetCDRMainFormHWND (PChar('D:\Program Files\Corel\CorelDRAW Graphics Suite X8\Programs64\CorelDRW.exe')); //本库提供的函数，获取 CDR 主窗口实例的句柄

  if vCDRHwnd > 0 then //CDR 主窗口句柄窗口获取成功
  begin
    //第一步，显示窗口
    ShowWindow(//Windows 系统 API，该函数设置指定窗口的显示状态
      vCDRHwnd, //CDR 主窗口句柄
      SW_NORMAL//窗口的显示方式为：正常状态(无论是否显示或隐藏于任务栏，一律恢复到桌面显示)
    );

    //第二步，窗口提前
    BringWindowToTop(//Windows 系统 API，该函数将指定的窗口设置到 Z 序的顶部。如果窗口为顶层窗口，则该窗口被激活；如果窗口为子窗口，则相应的顶级父窗口被激活。
```

```
        vCDRHwnd //CDR 主窗口句柄
      ); //
    end
    else //获取失败后，返回值
      case vCDRHwnd of
        0:
          ShowMessage('获取 CDR 主窗口句柄失败！');
        -1:
          ShowMessage('指定 EXE 文件不存在！');
        -2:
          ShowMessage('指定 EXE 文件没有运行！');
      end;
end;
```

12、其它 API

12.1 毫秒格式化为字符串

接口：API_FormatMillisecondString
原型：function API_FormatMillisecondString(const Ms: Int64): PChar; stdcall;
参数：Ms 毫秒数
返回：秒数格式化后的字符串
备注：Ms= 82265，格式化返回 0 小时 1 分 22 秒

12.2 获取文件版本号

接口：API_GetFileVersionStr
原型：function API_GetFileVersionStr(const FileName: PChar): PChar; stdcall;
参数：FileName 要获取版本号的文件名
返回：版本号
备注：

12.3 设置程序自动运行

接口：API_ SetAutoStart
原型：procedure API_SetAutoStart(AppName, AppTitle: PChar; IsRegister: Boolean); stdcall;
参数：AppName 自动运行的文件名 AppTitle 标题 IsRegister 是否自动运行，False 为取消自动运行
返回：无
备注：

12.4 检测某个端口是否被占用

接口：API_ PortTCPIsOpen
原型：function API_PortTCPIsOpen(APort: Word): Boolean; stdcall;
参数：Aport 要检测的 TCP 端口
返回：返回是否被其它程序占用 True 为占用
备注：

12.5 获取 CPU 版本信息

接口：API_ GetProcessorVersion
原型：function API_GetProcessorVersion: PChar; stdcall;
参数：无
返回：CPU 版本信息
备注：

12.6 获取物理内存大小

接口：API_ GetPhysicsMemoryTotalSize
原型：function API_GetPhysicsMemoryTotalSize: Int64; stdcall;
参数：无
返回：物理内存大小
备注：

12.7 获取进程内存使用量

接口：API_ GetMemoryUsed
原型：function API_GetMemoryUsed(AProcess: THandle = 0): Int64; stdcall;
参数：Aprocess 进程的 PID
返回：内存使用量
备注：

12.8 获取进程线程数量

接口：API_ GetThreadCount
原型：function API_GetThreadCount(AProcessID: Cardinal = 0): Cardinal; stdcall;
参数：Aprocess 进程的 PID
返回：进程线程数量
备注：

12.9 检测文件是否是 64 位的

接口：API_ IsCompiledAsX64
原型：function API_IsCompiledAsX64(const AExeName: PChar): Boolean; stdcall;
参数：AexeName 要检测的 EXE 文件名
返回：True 是 64 位
备注：

12.10 格式化字节数为 xx B/KB/MB/GB/TB

接口：API_ FormatByteNumber
原型：function API_FormatByteNumber(ASize: Int64; const ADecimals: Byte = 1): PChar; stdcall;
参数：Asize 要格式化的字节数 Adecimals 小数位数
返回：xx B/KB/MB/GB/TB 的字符串
备注：

12.11 获取字符串 MD5 的值

接口：API_ GetStringMD5
原型：function API_GetStringMD5(const AInput: PChar): PChar; stdcall;
参数：Ainput 要计算 MD5 的字符串
返回：字符串 MD5 的值
备注：

12.12 十进制转换为二进制

接口：API_ IntToBin
原型：function API_IntToBin(Value: LongInt; Size: Integer): PChar; stdcall;
参数：Values 要转换的十进制数字；Size：转换为二进制的位数
返回：转换后的二进制字符串
备注：

12.13 二进制转换为十进制

接口：API_ BintoInt
原型：function API_BintoInt(Value: PChar): LongInt; stdcall;

参数：Value 二进制字符串
返回：转换后的十进制
备注：

12.14 字符串转换为十六进制

接口：API_ StringToHex
原型：function API_StringToHex(const Value: PChar): PChar; stdcall;
参数：Value 要转换十六进制的字符串
返回：十六进制字符串
备注：

12.15 十六进制转换为字符串

接口：API_ HexToString
原型：function API_HexToString(const Hex: PChar): PChar; stdcall;
参数：Hex 十六进制字符串
返回：字符串
备注：

12.16 十六进制字符串转换为二进制字符串

接口：API_ HexToBinString
原型：function API_HexToBinString(Hex: PChar): PChar; stdcall;
参数：Hex 十六进制字符串
返回：二进制字符串 如：3E8 -> 1111101000
备注：

12.17 把二进制串转换成十六进制串

接口：API_ BinStringToHex
原型：function API_BinStringToHex(Bin: PChar): PChar; stdcall;
参数：Bin 二进制字符串
返回：十六进制字符串 如：1111101000 -> 3E8
备注：

12.18 生成随机字符串（包含数字、字母）

接口：API_ GenerateRandomString
原型：function API_GenerateRandomString(const ALength: Integer = 6): PChar; stdcall;
参数：Alength 生成字符串的位数
返回：随机字符串
备注：

12.19 获取设备名称

接口：API_ GetDeviceModel
原型：function API_GetDeviceModel: PChar; stdcall;
参数：无
返回：设备名称的字符串

12.20 获取设备唯一 ID

接口：API_ GetDeviceId

原型：function API_GetDeviceId: PChar; stdcall;

参数：无

返回：设备唯一 ID 的字符串

备注：

12.21 获取操作系统语言信息

接口：API_ GetWindowsLanguage

原型：function API_GetWindowsLanguage(const LanType: Integer = 0): PChar; stdcall;

参数：LanType 信息类型，可取值范围 0~4，定义如下：

0: LOCALE_ILANGUAGE // language id,语言 ID，如 0804

1: LOCALE_SLANGUAGE // localized name of locale,用本地语言表示，如 中文(简体，中国)

2: LOCALE_SENGLANGUAGE // English name of language,用英文表示，如 Chinese (Simplified)

3: LOCALE_SABBREVLANGNAME // abbreviated language name,用英文表示缩写名称，如 CHS

4: LOCALE_SNATIVELANGNAME // native name of language,语言本名，如 中文(简体)

返回：系统语言信息字符串

12.22 获取文件大小

接口：API_FileSizeByName

原型：function API_FileSizeByName(const AFileName: PChar): Int64; stdcall;

参数：AfileName 文件名（全路径）

返回：指定文件名的大小，单位：字节

12.23 获取文件的时间属性

接口：API_GetFileTimeByName

原型：function API_GetFileTimeByName(const AFileName: PChar; AFlag: Byte): TDateTime;

参数：AfileName 文件名（全路径）Aflag 0 创建时间 1 修改时间 2 访问时间

返回：指定文件名的时间属性

13、经典排序算法

13.1 冒泡排序

接口：API_ BubbleSort

原型：procedure API_BubbleSort(var abc: array of Integer); stdcall;

13.2 摇动排序

接口：API_ShakerSort

原型：procedure API_ ShakerSort (var abc: array of Integer); stdcall;

13.3 梳子排序

接口：API_ CombSort

原型：procedure API_ CombSort (var abc: array of Integer); stdcall;

13.4 选择排序

接口：API_ SelectionSort

原型：procedure API_ SelectionSort (var abc: array of Integer); stdcall;

13.5 标准插入排序

接口：API_ InsertionSortStd

原型：procedure API_ InsertionSortStd (var abc: array of Integer); stdcall;

13.6 优化的插入排序

接口：API_ InsertionSort

原型：procedure API_ InsertionSort (var abc: array of Integer); stdcall;

13.7 希尔排序

接口：API_ ShellSort

原型：procedure API_ ShellSort (var abc: array of Integer); stdcall;

13.8 标准并归排序

接口：API_ MergeSortStd

原型：procedure API_ MergeSortStd (var abc: array of Integer); stdcall;

13.9 优化的并归排序

接口：API_ MergeSort

原型：procedure API_ MergeSort (var abc: array of Integer); stdcall;

13.10 标准快速排序

接口：API_ QuickSortStd

原型：procedure API_ QuickSortStd (var abc: array of Integer); stdcall;

13.11 无递归的快速排序

接口：API_ QuickSortNoRecurse

原型：procedure API_ QuickSortNoRecurse (var abc: array of Integer); stdcall;

13.12 随机的快速排序

接口：API_ QuickSortRandom

原型：procedure API_ QuickSortRandom (var abc: array of Integer); stdcall;

13.13 中间值的快速排序

接口：API_ QuickSortMedian

原型：procedure API_ QuickSortMedian (var abc: array of Integer); stdcall;

13.14 优化的插入快速排序

接口：API_ QuickSort

原型：procedure API_ QuickSort (var abc: array of Integer); stdcall;

13.15 堆排序

接口：API_ HeapSort

原型：procedure API_ HeapSort (var abc: array of Integer); stdcall;

〈更多接口有待开发〉

END

附件一：

```
//=============================================================================================
 SystemFatalError = 1000; //系统错误
 UnknownError = 1001; //未知错误
 //-------------------------------------------------------------
 InitializeError = 1002; //初始化错误
 UnInitializeError = 1003; //卸载错误
 //-------------------------------------------------------------
 ComPortNotAvailable = 1004; //通讯口故障
 GetComPortListError = 1005; //获取通讯端口失败
 //-------------------------------------------------------------
 EnLockerNotExists = 1006; //加密锁不存在
 EnLockerActivaDisabled = 1007; //加密锁激活功能已禁用
 MainCardRegisterError = 1008; //板卡注册失败
 MainCardIsPirate = 1009; //板卡未激活
 MainCardIDError = 1010; //板卡 ID 无效
 EnLockerIDError = 1011; //加密锁 ID 无效
 CardDongleBindErr = 1012; //板卡和加密锁绑定失败
 CardDongleBindDuplicate = 1013; //重复绑定
 CardDongleBindOutTimes = 1014; //绑定次数超限
 CardDongleBindinvalid = 1015; //非法绑定
 LaserTubeZeroClearingErr = 1016; //激光管清零失败
 //-------------------------------------------------------------
 FactoryPasswordInValid = 1017; //厂家密码错误
 FactoryPasswordLenthError = 1018; //厂家密码长度无效
 FactoryPasswordExpired = 1019; //厂家密码过期请重新验证
 InputPassWordOutTimes = 1020; //密码输入错误次数过多
 ChangeFactoryPasswordError = 1021; //厂家密码修改失败
 //-------------------------------------------------------------
 ReadSysParamFromCardError = 1022; //读系统参数错误
 WriteSysParamToCardError = 1023; //保存系统参数错误
 ReadUserParamFromCardError = 1024; //读用户参数错误
 WriteUserParamToCardError = 1025; //保存用户参数错误
 SaveParamsToServerErr = 1026; //配置参数保存到服务器失败
 ReadParamsFromServerErr = 1027; //从服务端读取配置参数失败
 //-------------------------------------------------------------
 FileNotExistsError = 1028; //指定文件不存在
 DataFormatError = 1029; //数据格式错误
 DecryptCommandError = 1030; //解密错误
 //-------------------------------------------------------------
 ImageDataError = 1031; //图像数据无效
 GraphicMinSizeError = 1032; //图形尺寸过小
 GraphicMaxSizeError = 1033; //图形尺寸过大(超出可加工范围)
 //-------------------------------------------------------------
```

```
NoneDataTransError = 1034; //无传输数据
TransTimeOut = 1035; //数据发送超时
TimeOutReSend = 1036; //超时重发
ReSendDataOutTimes = 1037; //数据重发超次
SendDataError = 1038; //数据发送失败
//-----------------------------------------------------------------
ReceiveWrongDataOutTimes = 1039; //连续接收到多次错误的数据
BreakpointDataError = 1040; //断点续传数据错误
//-----------------------------------------------------------------
CanNotDoOnWorking = 1041; //设备運行期间不支持此操作
//-----------------------------------------------------------------
PingServerFail = 1042; //网络故障
ConnectServerError = 1043; //登录服务器失败
ConnectFrequently = 1044; //登录过于频繁
   SubmitToServerErr = 1045; //提交信息失败
ServerAccessDenied = 1046; //服务端拒绝访问
ServerReturnEmpty = 1047; //返回结果为空
//-----------------------------------------------------------------
UpdateInfoFileNotExists = 1048; //升级信息文件不存在
UpdateFileNotExists = 1049; //升级包不存在
SoftUpdateFailed = 1050; //升级失败
DownloadFirmwareDataError = 1051; //固件下载失败
UpdateFirmwareTimeOut = 1052; //固件升级等待超时
InadequatePermissions = 1053; //权限不足
//-----------------------------------------------------------------
SendEmailFailed = 1054; //邮件发送失败
MailboxInValid = 1055; //邮箱认证失败
MailBoxAccountErr = 1056; //邮箱帐号创建失败
ACTIVECODE_INVALID = 1057; //激活码错误
VALIDATECODE_INVALID = 1058; //验证错误
MailboxName_INVALID = 1059; //邮箱地址错误
//-----------------------------------------------------------------
//=================================================================================================
GetComPortListOK = 2000; //获取通讯端口成功
ComPortOpened = 2001; //通讯已连接
ComPortClosed = 2002; //通讯已断开
//-----------------------------------------------------------------
USBArrival = 2003; //USB 设备已连接
USBRemove = 2004; //USB 设备已断开
//-----------------------------------------------------------------
DongleArrival = 2005; //加密锁已连接
DongleRemove = 2006; //加密锁已断开
//-----------------------------------------------------------------
MainCardRegisterOK = 2007; //板卡注册成功
```

MainCardIsGenuine = 2008; //板卡已激活

MainCardIsGenuineEx = 2009; //板卡已临时激活

MainCardMachineMoreInfo = 2010; //返回板卡及机器注册及激活信息

//----------

CardDongleBindOK = 2011; //板卡和加密锁绑定成功

LaserTubeZeroClearingOK = 2012; //激光管清零成功

//----------

ReadSysParamFromCardOK = 2013; //读系统参数后返回数据 OK

WriteSysParamToCardOK = 2014; //保存系统参数后返回数据 OK

//

ReadUserParamFromCardOK = 2015; //读用户参数后返回的数据 OK

WriteUserParamToCardOK = 2016; //保存用户参数后返回的数据 OK

//

ReadComputerParamFromCardOK = 2017; //读上位机参数后返回的数据 OK

WriteComputerParamToCardOK = 2018; //保存上位机参数后返回的数据 OK

//

SaveParamsToServerOK = 2019; //配置参数成功保存到服务器

ReadParamsFromServerOK = 2020; //从服务端读取配置参数成功

//----------

FactoryPasswordValid = 2021; //厂家密码正确

ChangeFactoryPasswordOK = 2022; //厂家密码修改成功

//----------

ReturnTextMsgFromCallBack = 2023; //回调返回的字符信息

//----------

ImportFromFile = 2024; //导入文件完成

CancelCurrentWork = 2025; //加工任务取消

TimeConsuming = 2026; //加工任务总计耗时

EstimatedWorkTime = 2027; //预计加工耗时

StartProcData = 2028; //处理数据

DataTransCompleted = 2029; //数据发送完成

RequestAndContinue = 2030; //请求续传数据

//----------

MotorLock = 2031; //加载电机

MotorUnLock = 2032; //卸载电机

LaserLightOn = 2033; //激光点射开

LaserLightOff = 2034; //激光点射关

//----------

StartWorking = 2035; //开始加工

PauseWorking = 2036; //暂停加工

ContinueWorking = 2037; //继续加工

StopWorking = 2038; //停止加工

//----------

MachineWorking = 2039; //工作状态

NotWorking = 2040; //空闲状态

WorkFinished = 2041; //加工完成

//----------

DeviceIDInfo = 2042; //本机唯一 ID

ClientAddrInfo = 2043; //本机的外网 IP 地址

//

ConnectedServer = 2044; //已连接服务

DisConnectedServer = 2045; //已断开服务

ConnectServerOK = 2046; //登录服务器成功

SubmitToServerOK = 2047; //提交信息成功

//

DownLoadBegin = 2048; //开始下载

DownLoadEnd = 2049; //下载完成

//----------

NewVersionChecking = 2050; //正在检测版本，请稍候

NewVersionCheckFinished = 2051; //版本检测完成

IsLatestVersion = 2052; //当前版本已经最新

ReadyToUpdateFile = 2053; //准备升级

DownLoadUpdateInfoFile = 2054; //正在获取升级信息

//----------

FoundSoftNewVersion = 2055; //软件发现新版本

DownLoadFileCounts = 2056; //准备更新的文件总数

DownLoadFileIndex = 2057; //正在下载第 X 个文件

DownloadSoftDataStart = 2058; //开始下载软件升级包

StartSoftUpdate = 2059; //开始升级

CancelSoftUpdate = 2060; //升级取消

SoftUpdateFinished = 2061; //升级成功

//----------

FoundFirmwareNewVersion = 2062; //固件发现新版本

DownloadFirmwareDataStart = 2063; //准备下载固件升级包

DownloadFirmwareDataEnd = 2064; //固件下载完成

SendFirmwareDataStart = 2065; //开始发送固件数据

SendFirmwareDataEnd = 2066; //固件数据发送完成

UpdateFirmwareStart = 2067; //开始更新固件

UpdateFirmwareEnd = 2068; //固件更新完成

UpdateFirmwareAbort = 2069; //固件更新已取消

//----------

UpdateFinishedExit = 2070; //升级完成需要退出主程序

//----------

SendEmailFinished = 2071; //邮件发送成功

MailboxValid = 2072; //邮箱认证通过

MailBoxAccountOK = 2073; //邮箱帐号创建成功

RequestProcessing = 2074; //面板通过板卡返回请求:开始加工

RequestLensFocus = 2075; //面板通过板卡返回请求:镜头对焦

RequestDrawBounding = 2076; //面板通过板卡返回请求:走边框

附件二：

说明：**系统寄存器**的地址和内容如下：

地址	内容	说明
9	系统运行时间 Sys_run_time	
10	激光管开光时间 Laser_run_time	
11	系统加工次数 Sys_run_Num	
12	X 轴加工幅面 X_max_length	该运动轴所能行走的最远距离，根据机器的实际情况而定。
13	X 轴方向极性 X_dir_phase	修改方向极性可使电机向反方向运动。修改的目标是使该轴在复位时向原点运动，若复位时该轴向远离原点的方向运动，则说明该轴方向极性设置错误，应做修改。
14	限位极性 X_limit_phase	用于设置限位信号的高低电平模式。若运动轴到达限位位置时，向主板输入一个低电平信号，则此时的限位极性应设置为负。
15	原点偏移 X_zero_dev	若该轴使能了硬限位保护，通常应设置该值为 2~5mm 值，若设置为 0，则该运动轴运行到最小坐标 0 处时，有可能使限位有效，这样会错误触发硬限位保护功能，使机器紧急停机。若未使能硬限位保护，可设置该值为 0~5mm。单位 1um
16	电机步距 X_step_Length	即电机的脉冲当量，向电机发送一个脉冲时，对应运动轴走过的绝对距离值。在该值正确设置之前，可让机器切割一个较大的矩形（图形较大，可使误差较小），通过图形长度和测量长度来自动计算电机步距。单位 0.001um
17	限位输入选择 X_limit_num	选择一个外部输入接口作为该轴的限位控制信号
18	复位使能 X_reset_en	若机器配置有该轴，则该轴“复位使能”选项应打开，若未配置该轴，则该轴“复位使能”选项应禁止。该参数的意义在于控制用户参数里的“开机复位”选项和功能键里的“各轴复位”功能，防止用户错误地对某个并不存在的运动轴进行复位。
19	电机驱动接口选择 X_motor_num	为该轴指定一个电机驱动接口
20	电机驱动电流 X_motor_i	设定板载驱动器工作电流，设定范围 0-100%，厂家可根据机器实际负载值调整驱动电流，
21	起跳速度 X_start_speed	运动轴从静止状态直接启动的速度，若该值过大，会导致电机丢步、抖动甚至产生啸叫，设置过小，会降低整个图形的运行速度。若运动轴的惯性较大（轴较重），可设置一个较小的起跳速度，若运动轴的惯性较小（轴较轻），则可适当加大起跳速度。典型值如 5~30mm/s。
22	最大速度 X_max_speed	该轴所能承受的最高极限运动速度。该参数与电机的驱动能力、运动轴的惯性以及传动比有关。典型值如 200~500mm/s，单位 0.001mm/s
23	最大加速度 X_max_a	运动轴在进行加减速运动时的最大加速度值，加速度设置过大，同样会导致电机丢步、抖动甚至产生啸叫，设置过小，会导致加速缓慢而降低整个图形的运行速度。对应惯性较大的轴，如横梁所对应的 Y 轴，一个典型的设置范围为 800～3000mm/s2，对应惯性较小的轴，如小车所对应的 X 轴，一个典型的设置范围为 8000～20000mm/s2。单位 0.001mm/s2；
24	急停加速度 X_urgent_a	若该轴使能了硬限位保护，则当该轴运动到限位位置时，会对该轴以急停加速度进行紧急减速停机操作。该值可取该轴最大加速度的 2~3 倍值

25	Y_max_length	
26	Y_dir_phase	
27	Y_limit_phase	
28	Y_zero_dev	
29	Y_step_Length	
30	Y_limit_num	
31	Y_reset_en	
32	Y_motor_num	
33	Y_motor_i	
34	Y_start_speed	
35	Y_max_speed	
36	Y_max_a	
37	Y_urgent_a	
38	Z_max_length	
39	Z_dir_phase	
40	Z_limit_phase	
41	Z_zero_dev	
42	Z_step_Length	
43	Z_limit_num	
44	Z_reset_en	
45	Z_motor_num	
46	Z_motor_i	
47	Z_start_speed	
48	Z_max_speed	
49	Z_max_a	
50	Z_urgent_a	
51	Laser_max_power	
52	Laser_min_power	
53	Laser_power_freq	
54	X_en_phase	使能相位
55	Y_en_phase	
56	Z_en_phase	
57	原点位置	0:左上 1:左下 2:右下 3:右上
59	Z 轴复位速度	(mm/s)

附件三：

用户寄存器的地址和内容如下：

地址	内容	说明
1	加速模式 Acc_mode	0：T 型加速，1-5：S 型曲线加速，值越大，加速越柔和
2	空移速度 Cut_move_speed	
3	空移加速度 Cut_move_a	
4	拐弯速度 Cut_turn_speed	
5	拐弯加速度 Cut_turn_a	
6	切割加速度 Cut_work_a	
7	空移加速倍率 Cut_move_speed_per	
8	切割加速倍率 Cut_work_speed_per	
9	光斑尺寸 0-1000 Cut_spot_size	
10	扫描 x 轴起跳速度 Scan_x_start_speed	
11	扫描 y 轴起跳速度 Scan_y_start_speed	
12	扫描 x 轴加速度 Scan_x_a	
13	扫描 y 轴加速度 Scan_y_a	
14	换行速度 Scan_row_speed	
15	开光频率 1000~10000	
16	扫描回差 Scan_return_err	
17	扫描激光功率 Scan_laser_power	
18	x 轴复位使能 X_reset_en	
19	y 轴复位使能 Y_reset_en	
20	z 轴复位使能 Z_reset_en	
21	复位速度 Reset_speed (mm/s²)	
22	复位位置 Return_pos (mm)	
23	x 轴反向间隙 Backlash_x	
24	y 轴反向间隙 Backlash_y	
25	z 轴反向间隙 Backlash_z	
26	默认加工速度 Def run speed (mm/s)	
27	默认最大切割功率 Def max cut laser	
28	默认最小切割功率 Def min cut laser	
29	默认扫描速度 Def scan speed (mm/s)	
30	扫描最大灰度比	
31	扫描最小灰度比	
32	切割开光延时 us	
33	切割关光延时 us	
34	点射功率	(0~999)

35	填充速度	(mm/s)
36	填充起跳速度	(mm/s)
37	填充加速度	(mm/s²)
38	最大填充功率	(0~1000)
39	最小填充功率	(0~1000)
40	填充加速倍率	(0~200)

附件四：


```
{
   "LaserLibInfo":{
      "Languge":1,
      "OSInfo":"Windows 7 Service Pack 1 (Version 6.1, Build 7601, 64-bit Edition)",
      "CPURAM":"Intel(R) Core(TM) i5-2400S CPU @ 2.50GHz,19.9GB RAM",
      "CorelVersion":"CorelDRAW X8 (64-Bit) - SuiteVersion: 18.0.0.448 Installed",
      "API":"API: V2.1.0.0(x32)_UE_Compile:2021-08-07 23:16:48",
      "MD5":"813F54C99D0433D3E8B7F28F50A23ACF",
      "DongleID":"BFF3DB4F4D3C3516A662F107BF9XXXXX",
      "Agency":"南宁东方光电科技有限责任公司",
      "AreaCode":"1680150789",
      "SupportPhone":"139788xxxxx",
      "WX":"139788xxxxx",
      "Email":"2891XXXXX@qq.com",
      "HomePage":"www.xxxx.com"
   }
}
```


附件五：

下发的数据格式：JSON

数据中包含矢量数据（坐标）和图像数据（JPG/PNG）及图层参数信息、文件基本信息（版本号、创建日期）

完整的 JSON 文件内容如下：

```
{
    "LaserDocumentInfo": {   /*文档信息*/
        "APIVersion": "2.1.0.0",/*支持的 API 版本*/
        "BoundingRect": { /*外接矩形框的左上、右下角坐标*/
            "x1": 0,
            "x2": 46458.33365723211,
            "y1": 0,
            "y2": 46458.33365723211
        },
        "CreateDate": "2021-09-18 20:25:22",/*文档创建日期*/
        "DeviceOrigin": 0,/* DeviceOrigin：设备原点，0-左上，1-右上，2-右下，3-左下*/
        "FinishRun": 0, /*全部加工完成后执行的动作*/
        "JobOrigin": 2, /*作业原点九宫格顺序，0-左上，1-上，2-右上，3-左，4-中，5-右，6-左下，7-下，8-右下*/
        "Origin": { /*文档原点坐标，包含 x 和 y 两个子属性*/
            "x": 0,
            "y": 0
        },
        "PrinterDrawUnit": 1016, /*绘图单位*/
        "StartFrom": 2 /*开始加工的方式，0-当前位置，1-用户原点，2-绝对坐标*/
    },
    "Layers":   /*层参数信息*/
    [
            {
                "Items": [
                    {
                        "Data": "PU0 0;PD0 20000;PD20000 20000;PD20000 0;PD0 0;", /*PLT 数据*/
                        "Name": "Rect_0",
                        "Type": "Rect"
                    }
                ],
                "Params": {
                    "CuttingParams": {/*切割参数*/
                        "LaserPower": 23, /*功率*/
                        "MinSpeed": 15000, /*低速*/
                        "RunSpeed": 60000, /*高速*/
                        "Type": 1
                    },
                    "EngravingParams": {/*雕刻参数*/
                    }
```

```
                }
            },
            {
                "Items": [
                    {
                        "Data":"8k4AAHq1AAAgTgAAcAIAAMIIYQQwghh ",/*BMP 数据*/
                        "Height": 26.458335876464844, /*高度*/
                        "ImageType": "PNG",
                        "Name": "Bitmap_0",
                        "Type": "Bitmap",
                        "Width": 26.458335876464844/*宽度*/
                    }
                ],
                "Params": {
                    "CuttingParams": {/*切割参数*/
                    },
                    "EngravingParams": {/*雕刻参数*/
                        "CarveForward": true,
                        "CarveStyle": 0,
                        "ErrorX": 0,
                        "HStep": 7,
                        "LStep": 0,
                        "LaserPower": 8,
                        "MinSpeed": 15000,
                        "MinSpeedPower": 70,
                        "RunSpeed": 60000,
                        "RunSpeedPower": 100,
                        "Type": 0
                    }
                }
            }
        ]
    }
```

==========

关于 FinishRun 参数取值的说明：

FinishRun ，加工完成后执行的功能动作，使用 2 字节 0xFFFF 表示，取值时需要注意：

低位字节按 10 进制取值，为单选动作使用，共可控制 0xFF = 256 种动作；

0、停在当前位置

1、释放电机

2、回机械原点

3、回加工原点 1

4、回加工原点 2

5、回加工原点 3

例：0-5 共 6 个动作选项，且为单选，低位字节取值即所在编号（释放电机编号为 1，回机械原点编号为 2）

回机械原点用二进制表示为:10 = 0x02

高位字节按 2 进制位取值，为多选动作使用，共可控制 0xFF = 8 个电器 = 2*8=16 种开关状态；

下述功能各占一个 BIT 位，0 为关（或取消）1 为开（或选择）：

0、开关 1 号继电器//此位为最低 BIT 位

1、开关 2 号继电器

2、开关 3 号继电器

例 1：开 1、3 号继电器，二进制表示为:00000101 = 0x05

例 2:开 1、3 号继电器并回机械原点，则 FinishRun 相应二进制位表示为:0000010100000010 = 0x502，即 FinishRun = 1282

计算方法：FinishRun = 0x05 * 256 + 0x02 = 0x502 = 1282

==========