WiMi-net 无线自组网管理平台

Win32 SDK 说明书

（初级版）

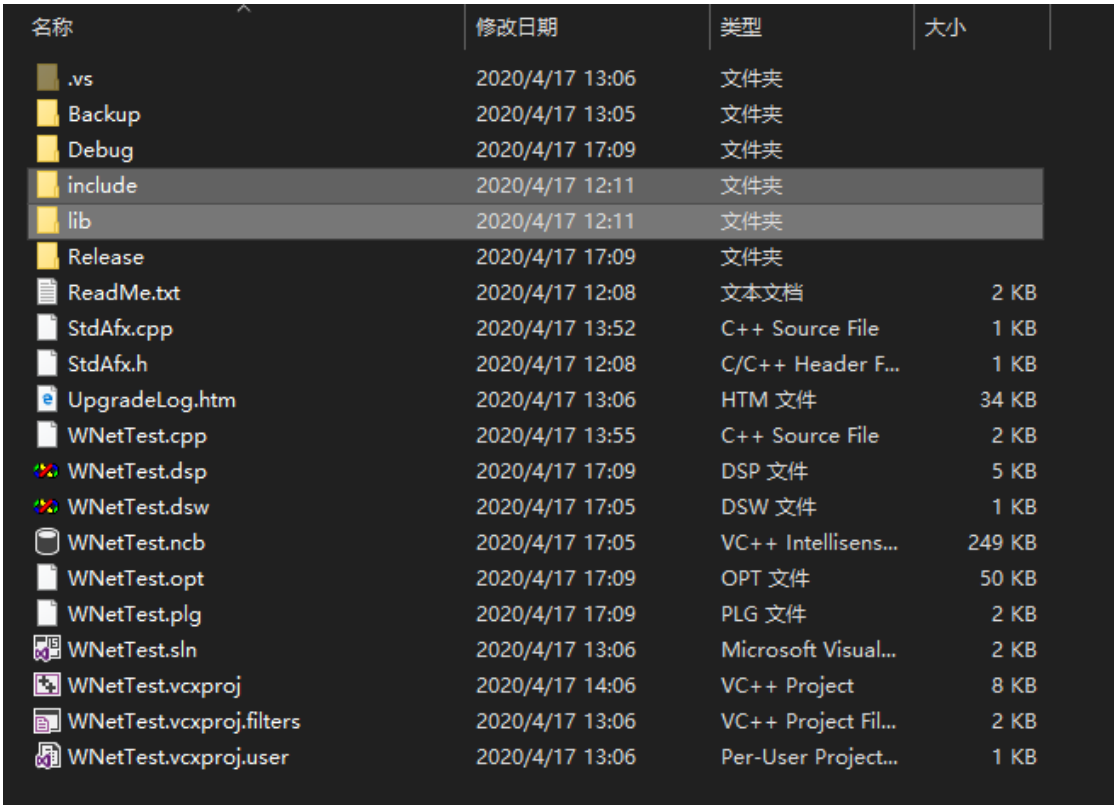
微网高通(北京)无线技术有限公司

目　录

1. 建立安装和开发环境

1.1 准备工作

(1)建立目标工程，比如“**W:\WNetTest**”



(2)将“**WiMinet.dll**”和“**WirelessTCP.dll**”动态链接库文件放入目标 EXE 程序的搜索路径中，有两个路径是应用程序必须会搜索的：

a) Windows 操作系统的公共路径，比如“**C:\Windows\System32**”

b) 可以执行程序的目录，比如： “**W:\WNetTest**”

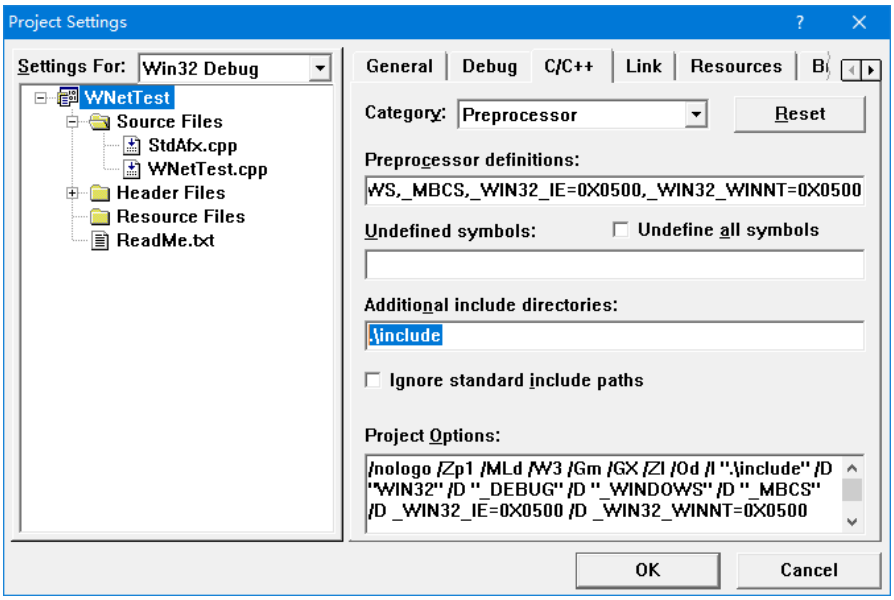
(3)在工程目录“**W:\WNetTest**”建立 include 子目录“**W:\WNetTest\include**”，将文件“**API-Message.h**”和“**API-WiMinet.h**”文件放入目录“**W:\WNetTest \include**”

(4)在工程目录“**W:\WNetTest**”建立 lib 子目录“**W:\WNetTest\lib**”，将静态链接库文件“**WiMinet.lib**”和“**WirelessTCP.lib**”放入目录“**W:\WNetMgr\lib**”

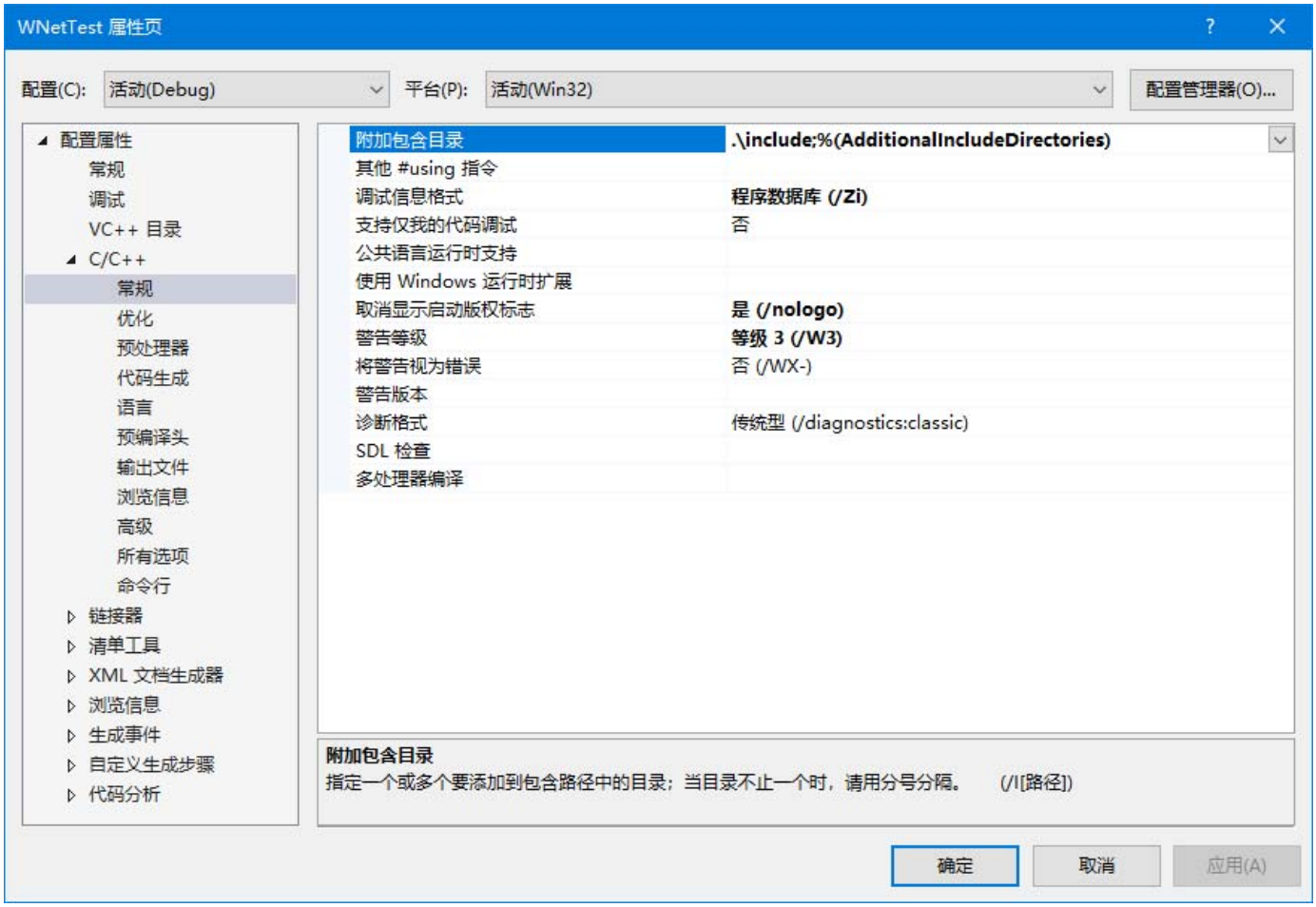
1.2 设置头文件包含路径

将头文件的路径“W:\WNetTest\include”加入工程的包含路径中。

下图是 Microsoft Visual Studio 6.0 的头文件包含路径的设置。



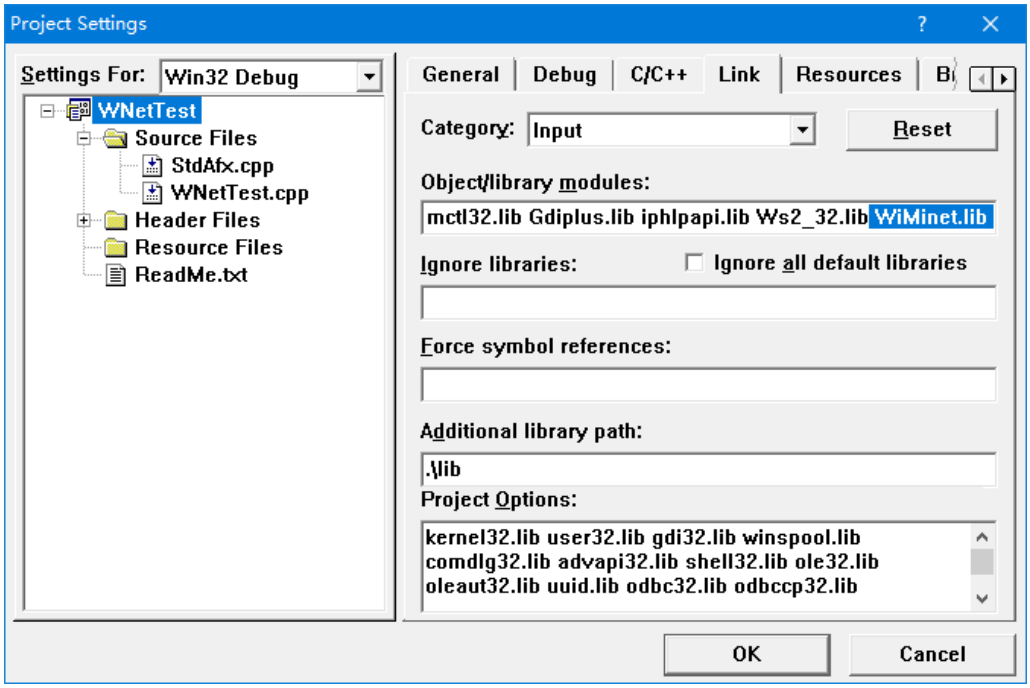
下图是 Microsoft Visual Studio 2017 的头文件包含路径的设置



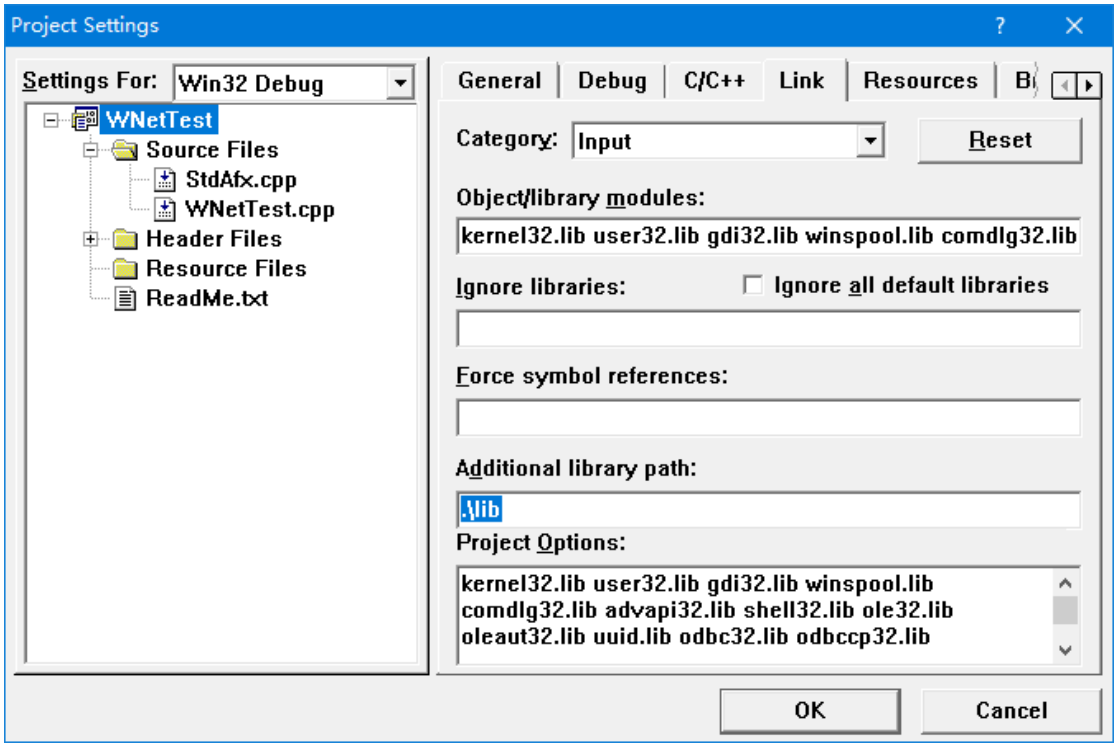
1.3 设置库文件引用路径

将 lib 文件的路径“W:\WNetTest\lib”加入工程的库文件路径中

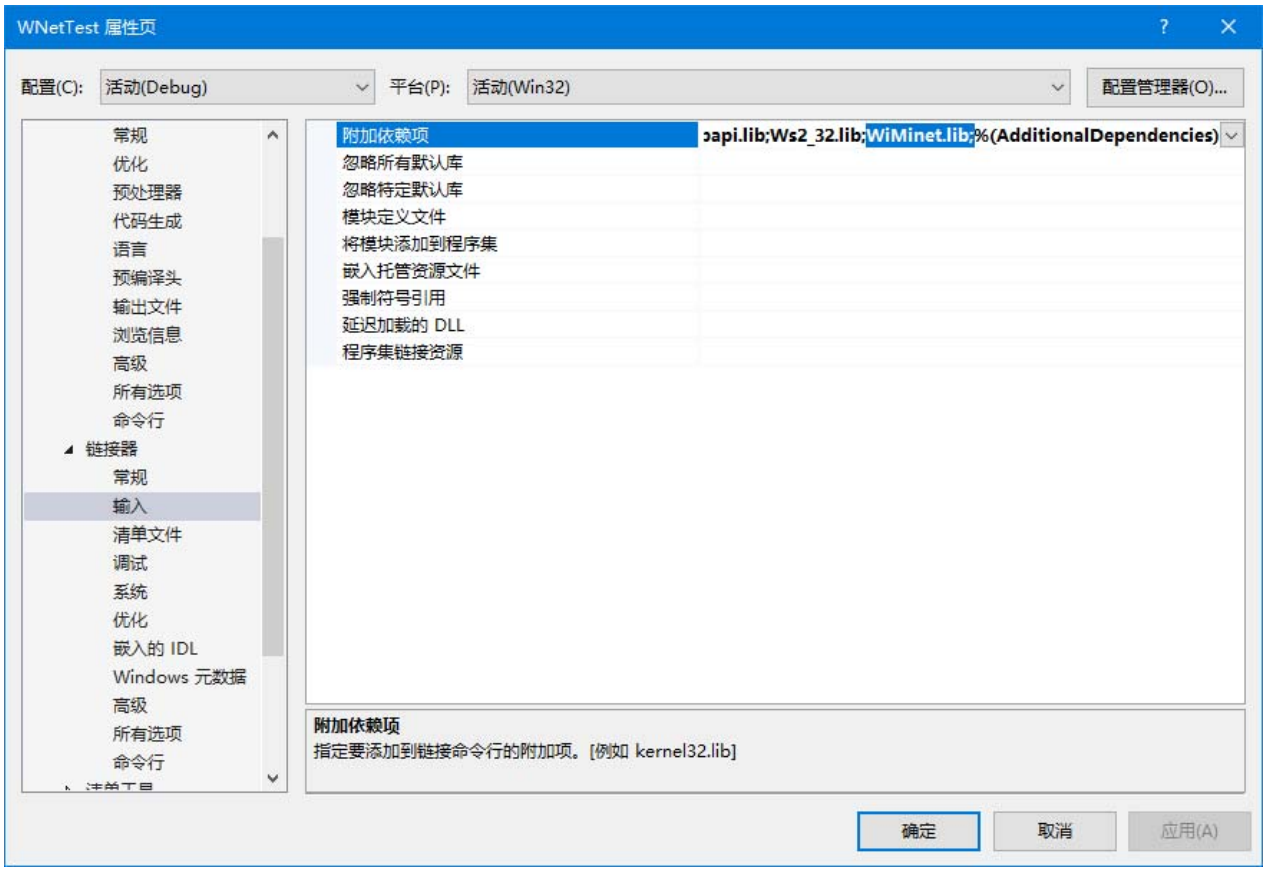
下图是 Microsoft Visual Studio 6.0 的库文件名称设置



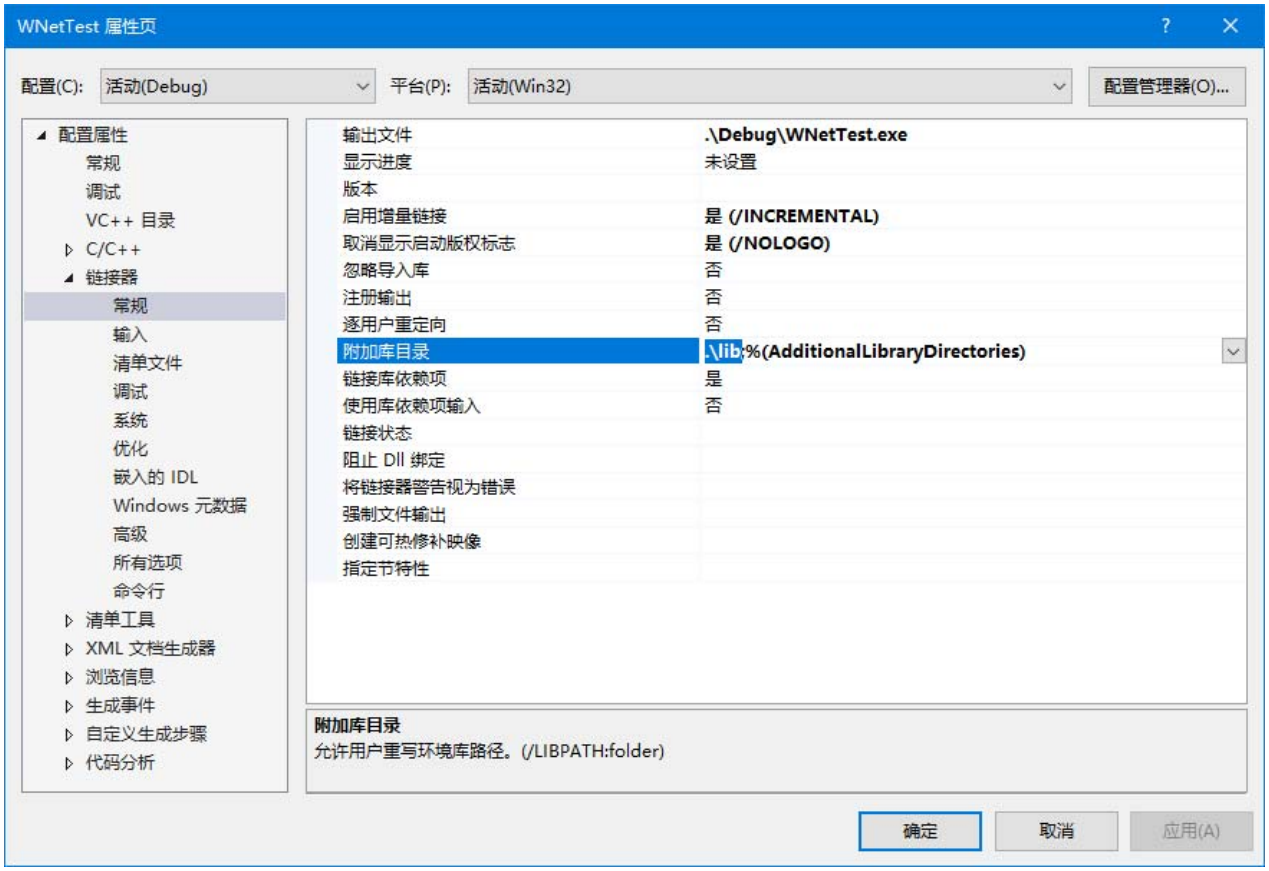
下图是 Microsoft Visual Studio 6.0 的库文件路径设置



下图是 Microsoft Visual Studio 2017 的库文件的名称设置



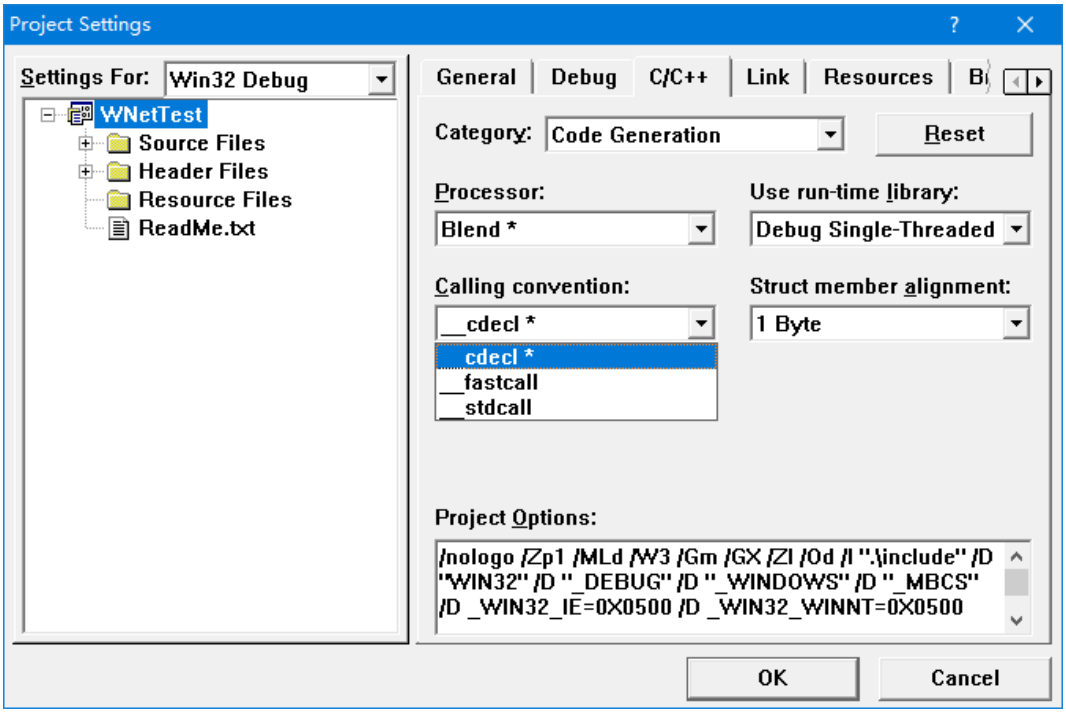
下图是 Microsoft Visual Studio 2017 的库文件的路径设置



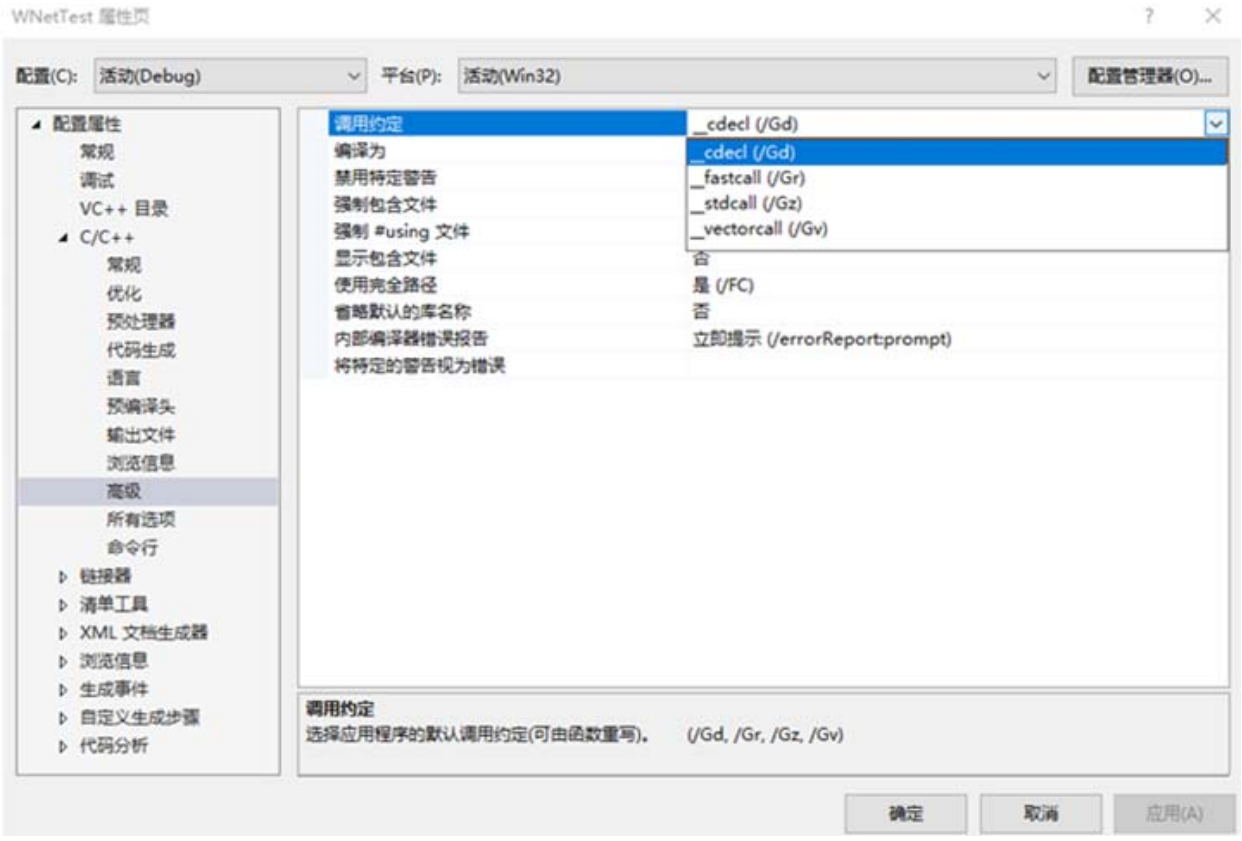
1.4 设置函数调用规范

(1)设置 DLL 的调用规范为 __cdecl * 的格式

下图是 Microsoft Visual Studio 6.0 的设置



下图是 Microsoft Visual Studio 2017 的截图



1.5 同步更新 Debug 和 Release 版本

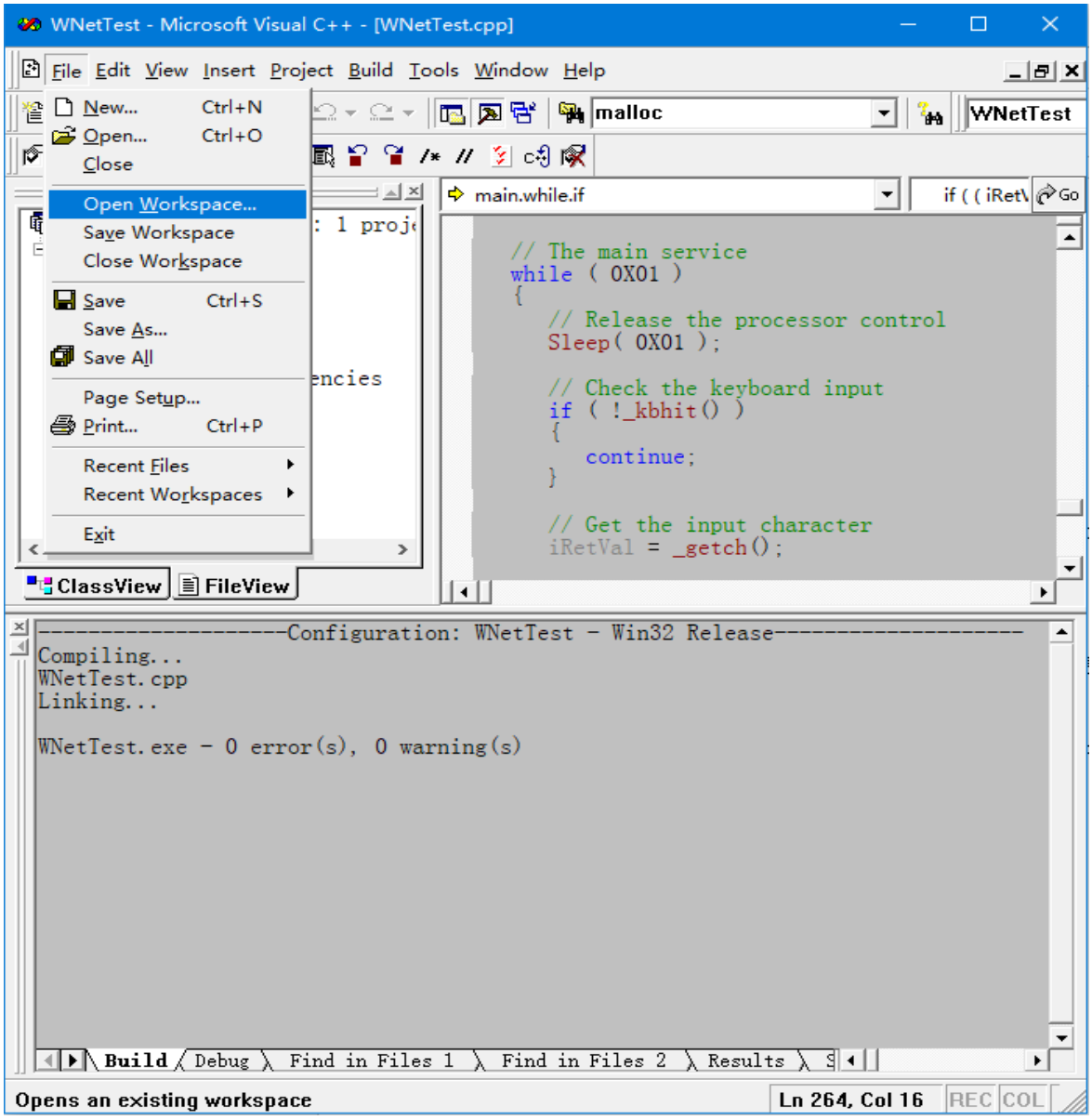
针对 Debug 版本和 Release 版本，执行上述步骤 1.2，1.3，1.4 三个章节的配置

1.6 Visual Studio 6.0 和 2017 的工程管理

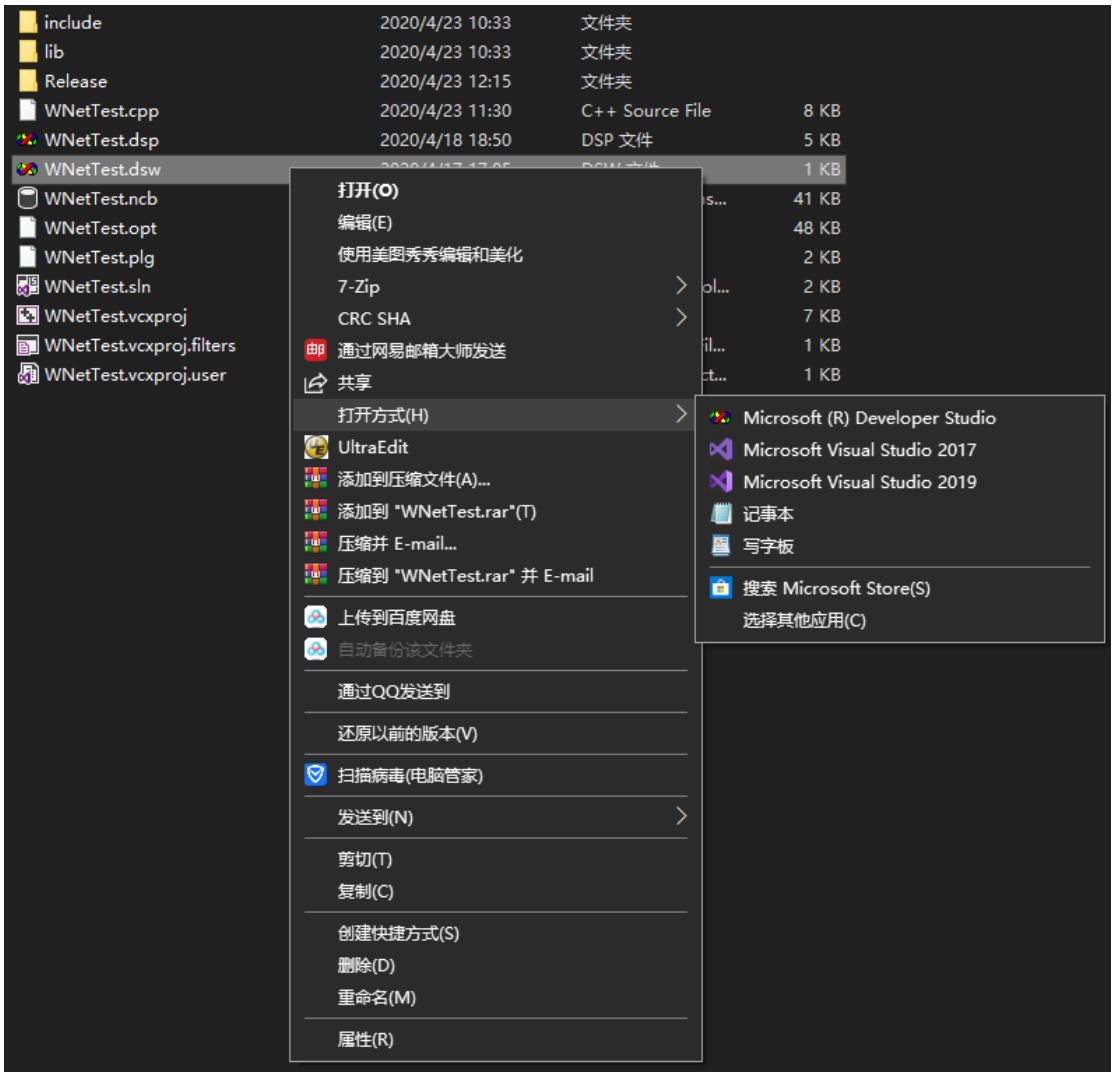
Microsoft Visual Studio 6.0 的工程文件尾缀是 *.dsp 以及 *.dsw；Microsoft Visual Studio 2017 的工程文件的尾缀是 *.sln。

由于 Windows 操作系统文件关联的设置，有可能双击 *.dsp 或者 *.dsw 工程文件的时候，仍然会采用 Microsoft Visual Studio 2017 打开。精确控制打开的集成开发环境，有下述两种方法：

(1) 先启动 Microsoft Visual Studio 6.0，然后从菜单“**File**”中选择“**Open Workspace**”打开工程。

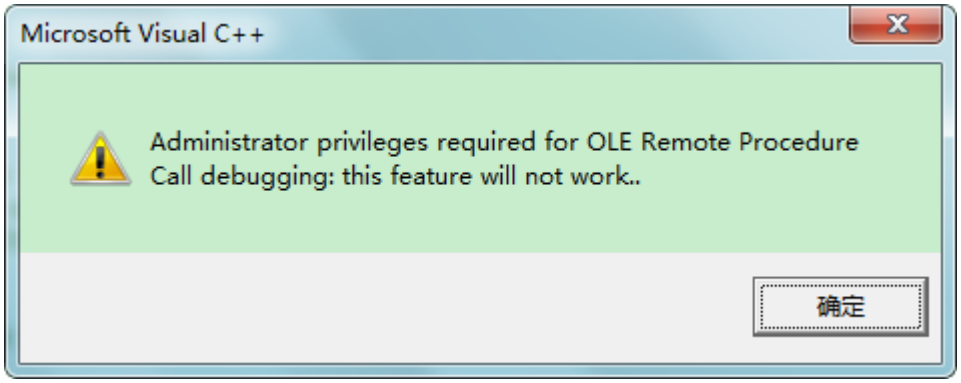


(2) 右键单击 *.dsp 或者*.dsw 文件，选择“**打开方式**”，然后通过“**打开方式**”的二级子菜单选择“**Microsoft(R) Developer Stduio**”打开 Microsoft Visual Studio 6.0 集成开发环境。



1.7 注意事项

Microsoft Visual C++ 6.0 在 Windows 10 操作系统上运行的时候，有权限方面的兼容性问题。在单步运行 Step into/over 的时候，会弹出来“OLE Remote Procedure Call”的警告，此时需要以管理员身份运行 Microsoft Visual C++ 6.0 的程序组件即可。



具体如下：选中 Microsoft Visual C++ 6.0 的程序图标，右键弹出菜单，选中“以管理员身份运行”即可。

1.8 清除 Visual Stdudio 临时文件

Microsoft Visual Stdudio 集成开发环境会在编译过程中产生临时文件，为了清除这些文件，可以执行根目录的批处理文件“**del.VCTemp.bat**”，用记事本打开（Win+R 调出“**运行**”命令，输入 notepad 并回车确认，然后用鼠标将该文件拖入 notepad 空白处并释放鼠标），显示其内容如下：

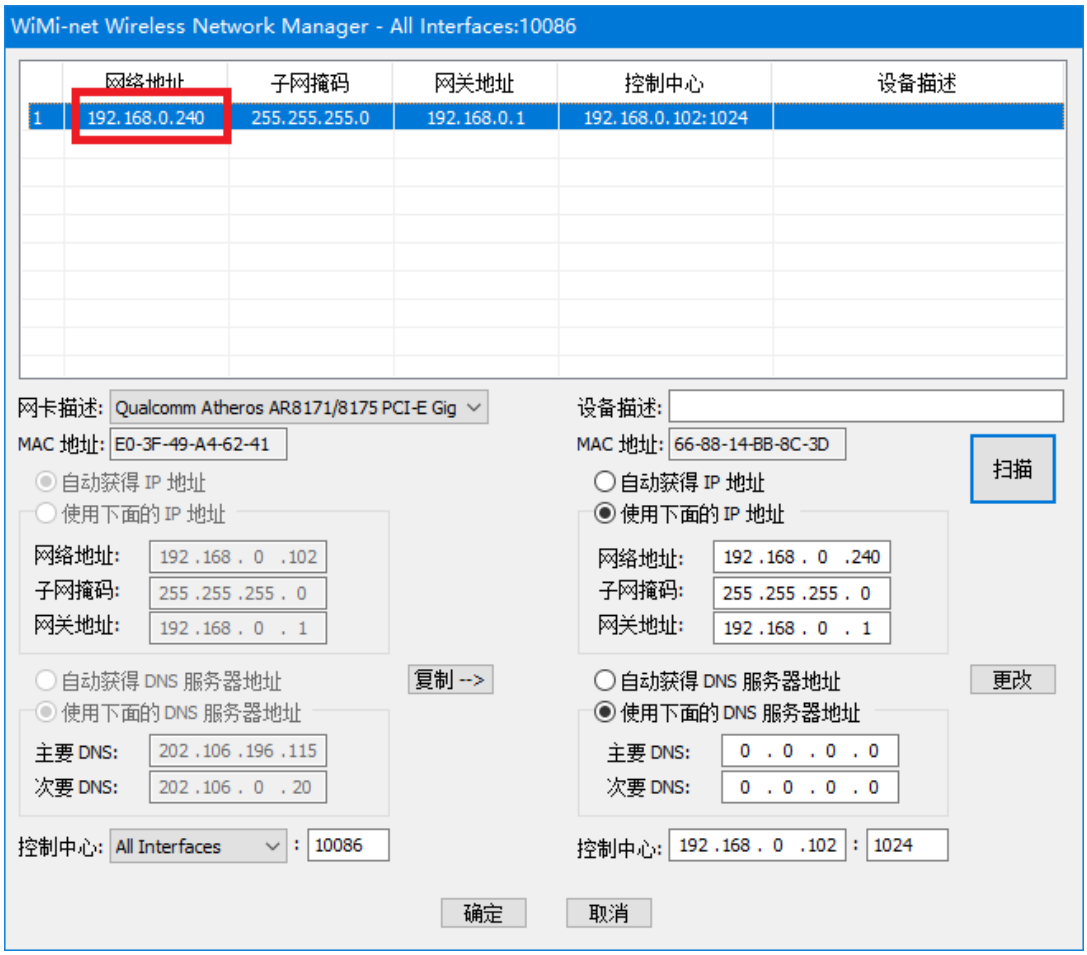
```
del /s *.pdb
del /s *.pch
del /s *.ilk
del /s *.log
del /s *.tlog
del /s *.ncb
del /s *.plg
del /s *.opt
del /s *.obj
del /s *.htm

rem Delete [Version].....
rmdir /S/Q    Version\Backup
rmdir /S/Q    Version\.vs
rmdir /S/Q    Version\Debug\WNetTest.tlog
rmdir /S/Q    Version\Release\WNetTest.tlog

rem Delete [Send].....
rmdir /S/Q    Send\Backup
rmdir /S/Q    Send\.vs
rmdir /S/Q    Send\Debug\WNetTest.tlog
```

```
rmdir /S/Q    Send\Release\WNetTest.tlog

rem Delete [Recv]......
rmdir /S/Q    Recv\Backup
rmdir /S/Q    Recv\.vs
rmdir /S/Q    Recv\Debug\WNetTest.tlog
rmdir /S/Q    Recv\Release\WNetTest.tlog

rem Delete [eWOR]......
rmdir /S/Q    eWOR\Backup
rmdir /S/Q    eWOR\.vs
rmdir /S/Q    eWOR\Debug\WNetTest.tlog
rmdir /S/Q    eWOR\Release\WNetTest.tlog

rem Delete [BMP]......
rmdir /S/Q    BMP\Backup
rmdir /S/Q    BMP\.vs
rmdir /S/Q    BMP\Debug\WNetTest.tlog
rmdir /S/Q    BMP\Release\WNetTest.tlog

rem Delete [[App-01].eWOR+Send]......
rmdir /S/Q    [App-01].eWOR+Send\Backup
rmdir /S/Q    [App-01].eWOR+Send\.vs
rmdir /S/Q    [App-01].eWOR+Send\Debug\WNetTest.tlog
rmdir /S/Q    [App-01].eWOR+Send\Release\WNetTest.tlog
```

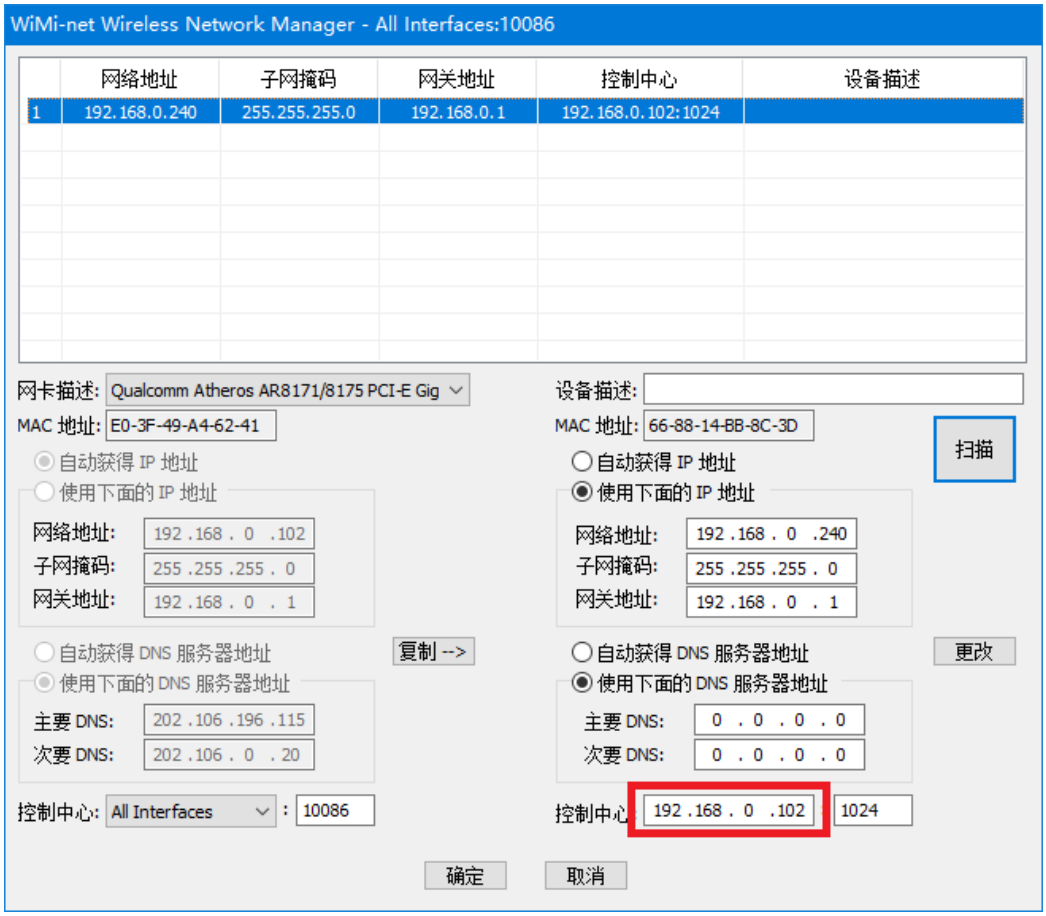
2. 连接和断开无线设备

2.1 打开通讯端口

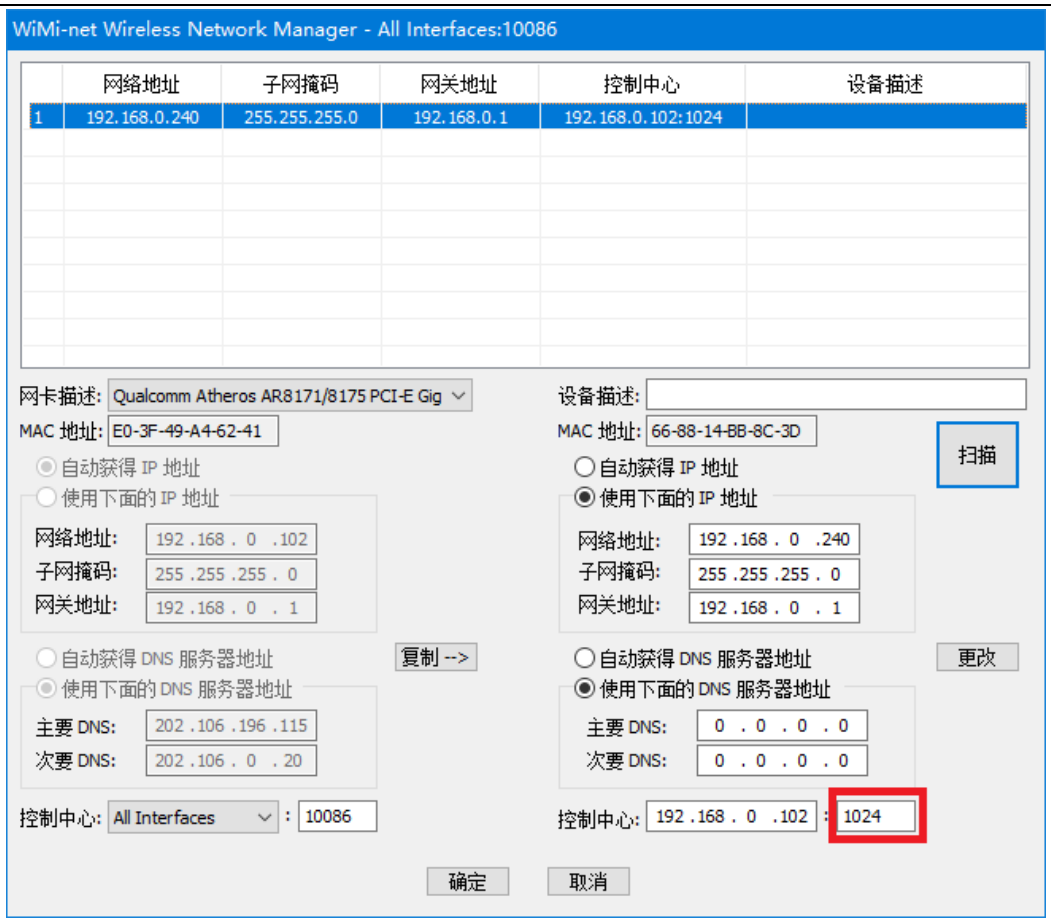
函数名	**char OpenWiMinetShell** （char * pDevice, long dwParam, char iBlockMode）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char * pDevice	目标设备的通讯端口名称，以字符串形式输入，分为串口和网口两种接口端口类型，共计三种模式： a）串口通讯，就用“**COM1**”，“**COM2**”等名称，不区分大小写，也成“**com1**”，“”“**com2**”等也可以。 b）网口通讯，基站作为 TCP Server，填写基站的 IP 地址，比如“**192.168.0.240**”，见下图中“**TCP Server 模式**”红色框住的部分。 c）网口通讯，基站作为 TCP Client，填写运行当前目标程序的 PC 或者云服务器的 IP 地址，比如 PC 或者云服务器的 IP 地址，同时需要确保基站配置软件中控制中心的 IP 地址需要填写为该地址，见下图中“**TCP Client 模式**”红色框住的部分
参数二	**long dwParam**	跟通讯端口相关的参数，针对串口和网口两种接口类型，有下述三种设定： a）串口通讯：填写基站的串口波特率，默认是 115200 b）网口通讯，基站做 TCP Server 模式，填写基站的 TCP 通讯端口，填写固定数值 12580 c）网口通讯，基站做 TCP Client，模式，填写运行该目标程序的 PC 或者云服务器的 TCP Server 的端口号码，该数值需要和基站的设定一致，见下图“**TCP 端口号码**”红色方框所示
参数三	**char iBlockMode**	通讯模式，同步阻塞模式，填写 0X01，异步非阻塞模式，填写 0X00，下面是两种模式的使用方式以及具体差异： a）同步模式下：所有的操作命令，发完了请求报文之后，需要读取返回结果，经过了一定的时间如果没有收到应答，返回失败。 b）异步模式下：所有的操作命令，发送完了请求报文之后，不需要读取返回结果，等待返回结果到达之后，发送通知消息给应用程序的窗体或者其他资源句柄，通知其来读取命令结果。
返回值	0X01=操作成功，0X00=操作失败	
示例一	打开串口通讯 OpenWiMinetShell ("COM3",115200, 0X01)	
示例二	打开网口通讯 OpenWiMinetShell ("192.168.0.240",12580, 0X01)	



TCP Server 模式



TCP Client 模式



TCP 端口号码

2.2 关闭通讯端口

函数名	**char StopWiMinetShell （char iShell）**	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
返回值	0X01=操作成功，0X00=操作失败	

3. 读取固件的版本号码

3.1 函数说明

函数名	**char GetModuleVersion** （char iShell， VersionInfo * pVersion）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	**VersionInfo * pVersion**	指向一个已经分配好实体内存空间的 VersionInfo 结构体，详细定义见“**VersionInfo**”结构体说明
返回值	0X01=操作成功，0X00=操作失败	

3.2 测试例程：固件版本信息 VersionInfo 结构体

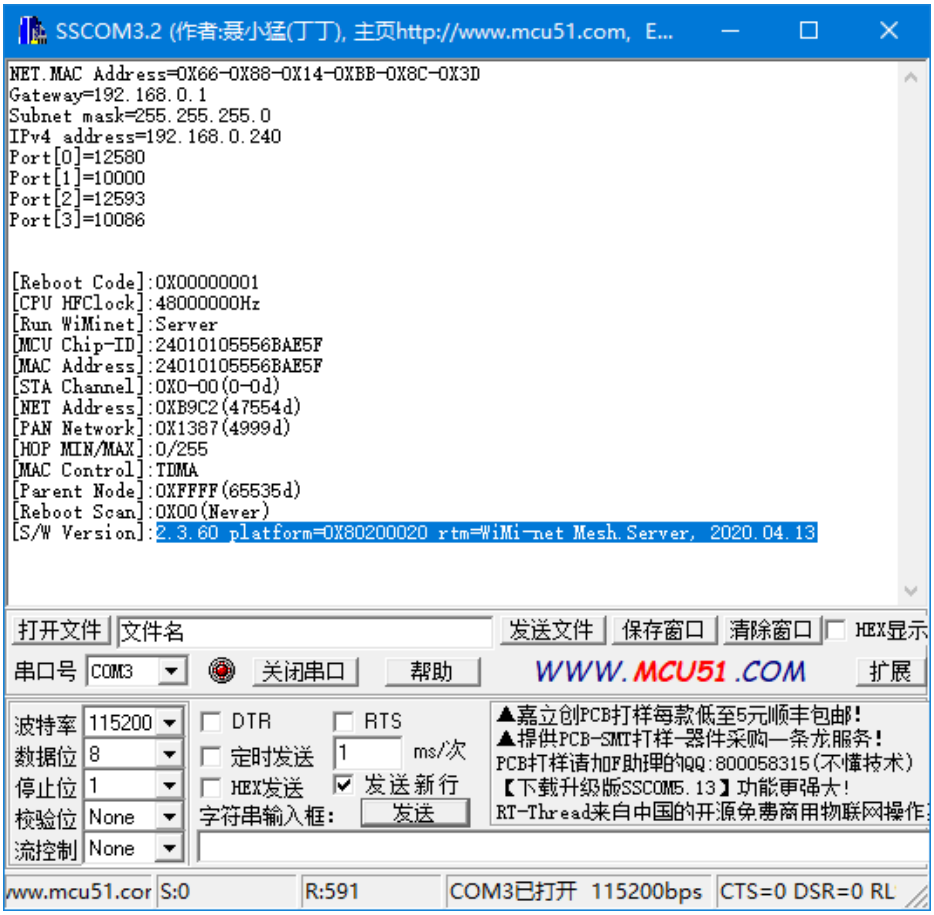
```
// ----------------------------------------------------------------------
// DESCRIPTION:
// ----------------------------------------------------------------------
typedef struct _VersionInfo
{
    // ----------------------------------------------------------------------
    // DESCRIPTION:
    // ----------------------------------------------------------------------
    unsigned long                   m_dwInfoSize;

    // ----------------------------------------------------------------------
    // DESCRIPTION:
    // ----------------------------------------------------------------------
    unsigned long                   m_dwMajorVersion;

    // ----------------------------------------------------------------------
    // DESCRIPTION:
    // ----------------------------------------------------------------------
    unsigned long                   m_dwMinorVersion;

    // ----------------------------------------------------------------------
    // DESCRIPTION:
    // ----------------------------------------------------------------------
    unsigned long                   m_dwBuildNumber;

    // ----------------------------------------------------------------------
    // DESCRIPTION:
    // ----------------------------------------------------------------------
    unsigned long                   m_dwPlatformId;

    // ----------------------------------------------------------------------
```

```
    // DESCRIPTION:
    // --------------------------------------------------------------------
    char                                    m_pVersion[0X80];

} VersionInfo;
```

名称	VersionInfo
数值	**说明**
m_dwInfoSize	描述信息的实际长度，32 位长整型
m_dwMajorVersion	固件的次要版本号码，32 位长整型
m_dwMinorVersion	固件的次要版本号码，32 位长整型
m_dwPlatformId	设备的硬件平台标识，32 位长整型
m_pVersion	设备的固件描述信息，字符串，最大长度 128 字节

3.3 上电自举与固件升级

设备的固件版本信息，在上电自举完成之后，会从串口打印出来，其中最后一行“[S/W Version]”一栏就是固件的信息。可以将该 API 读取到的信息与打印信息对比验证正确性。另外设备在做完固件升级之后，其版本号码会发生变化。通过对比版本号码的变化可以检验升级成功与否。



4. 快速上手：测试和设备的连接

4.1 测试步骤

(1)调用函数 OpenWiMinetShell 打开通讯端口，确保操作成功

(2)调用函数 GetModuleVersion 读取固件的版本函数，如果可以正确读取版本号码，就说明通讯端口打开是正确的，和设备的连接是正常的。

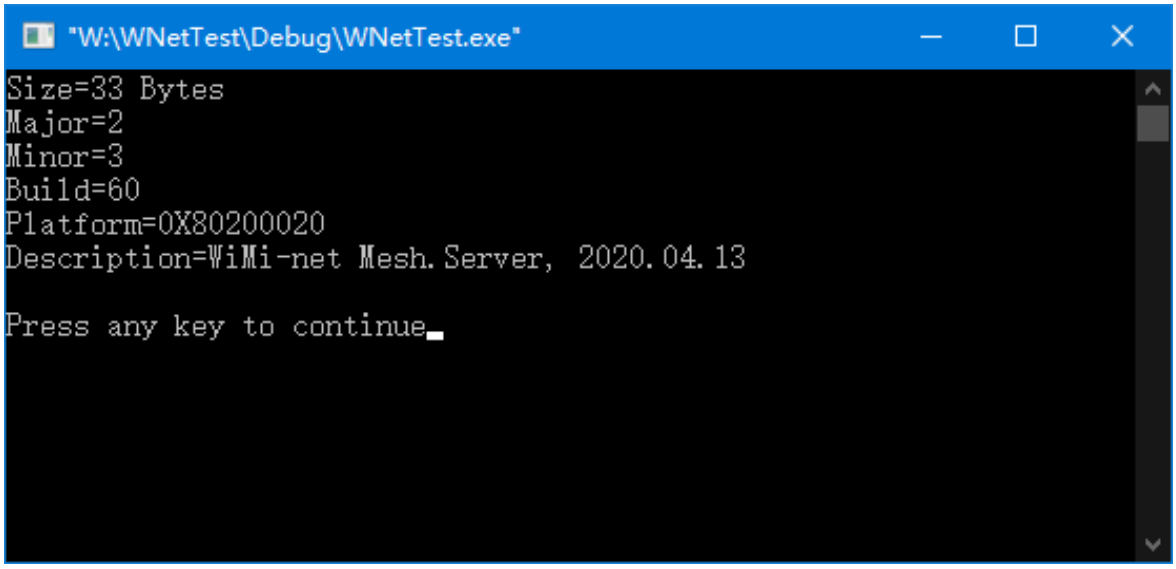
(3)使用本手册中的其他函数，展开测试和集成工作。

4.2 测试例程：获取固件版本信息

```
#include <stdio.h>
#include <Winsock2.h>
#include "API-WiMinet.h"

int main( int argc, char* argv[] )
{
    char iRetVal;
    VersionInfo version;

    // COM port interface
    iRetVal = OpenWiMinetShell( "COM3",115200, 0X01 );

    // Ethernet interface
    //iRetVal = OpenWiMinetShell( "192.168.0.240",12580, 0X01 );

    // Validate the shell open interface
    if ( !iRetVal )
    {
        printf( "Open shell failed!\r\n" );
        return 0X00;
    }

    // Get the module firmware version
    iRetVal = GetModuleVersion(0X00, &version);

    // Validate the operation status
```

```
    if ( !iRetVal )
    {
        printf( "GetModuleVersion failed!\r\n" );
    }

    // The version information
    printf( "Size=%d Bytes\r\n", ntohl( version.m_dwInfoSize ) );
    printf( "Major=%d\r\n", version.m_dwMajorVersion );
    printf( "Minor=%d\r\n", version.m_dwMinorVersion );
    printf( "Build=%d\r\n", version.m_dwBuildNumber );
    printf( "Platform=0X%08lX\r\n", version.m_dwPlatformId );
    printf( "Description=%s\r\n\r\n", version.m_pVersion );
    return 0X01;
}
```

4.3 运行结果与分析



该程序首先打开串口或者网口，建立和设备的连接。在确保连接成功之后，读取设备的固件版本信息，并打印出版本信息的各个字段，具体详见各个字段的定义。

5. 发送无线数据

5.1 查询发送状态

函数名	**char QueryTxMsgStatus** （char iShell, char * pStatus）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	char * pStatus	指向一个已分配好实体内存空间的单字节变量，用于记录发送状态，详细定义见“**发送状态**”表所示。
返回值	0X01=操作成功，0X00=操作失败	

数值名称	发送状态说明
0XFF	系统空闲，没有发送任务
0-100	正在发送报文，当前数值为发送的百分进度比，比如 35 代表发送了总长度 35%的报文
0X81	发送成功，且已经附加了前导描述信息，报文总长度，32 位的 CRC 等校验信息
0X82	发送失败
0X83	发送成功，但是没有附加前导描述信息

5.2 以 TCP 发送数据报文

函数名	**char SendHiQoSMessage**（char iShell, short iNode, char * pBuffer, long dwSize）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	short iNode	报文的目标接收节点 ID 号码
参数三	char * pBuffer	报文数据块的内存地址
参数四	long dwSize	报文数据块的长度
备注	途径的所有的传输节点，都采用 TCP 方式传输数据	
返回值	0X01=操作成功，0X00=操作失败	

5.3 以 UDP 发送数据报文

函数名	**char SendNoQoSMessage**（char iShell, short iNode, char * pBuffer, long dwSize）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	short iNode	报文的目标接收节点 ID 号码
参数三	char * pBuffer	报文数据块的内存地址
参数四	long dwSize	报文数据块的长度
备注	途径的所有的传输节点，都采用 UDP 方式传输数据	
返回值	0X01=操作成功，0X00=操作失败	

5.4 以 TCP/UDP 发送数据报文

函数名	**char SendMxQoSMessage**（char iShell, short iNode, char * pBuffer, long dwSize）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	short iNode	报文的目标接收节点 ID 号码
参数三	char * pBuffer	报文数据块的内存地址
参数四	long dwSize	报文数据块的长度
备注	接收报文的目标节点采用 UDP 传输方式，其余节点采用 TCP 传输方式	
返回值	0X01=操作成功，0X00=操作失败	

5.5 设置 UDP 发送等级

函数名	**char SetTxPerformance** （char iShell, unsigned char iSize）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	unsigned char iSize	UDP 发送的等级，取值范围 0-255，数值越大，发送的等级越高，可靠性越强。0 代表采用设备默认的配置，最低数值 0X03
返回值	0X01=操作成功，0X00=操作失败	

5.6 读取 UDP 发送等级

函数名	**char GetTxPerformance** （char iShell）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
返回值	UDP 发送的等级，取值范围 0-255，数值越大，发送的等级越高，可靠性越强。 0 代表采用设备默认的配置，最低数值 0X03	

5.7 测试例程：向远程节点发送数据

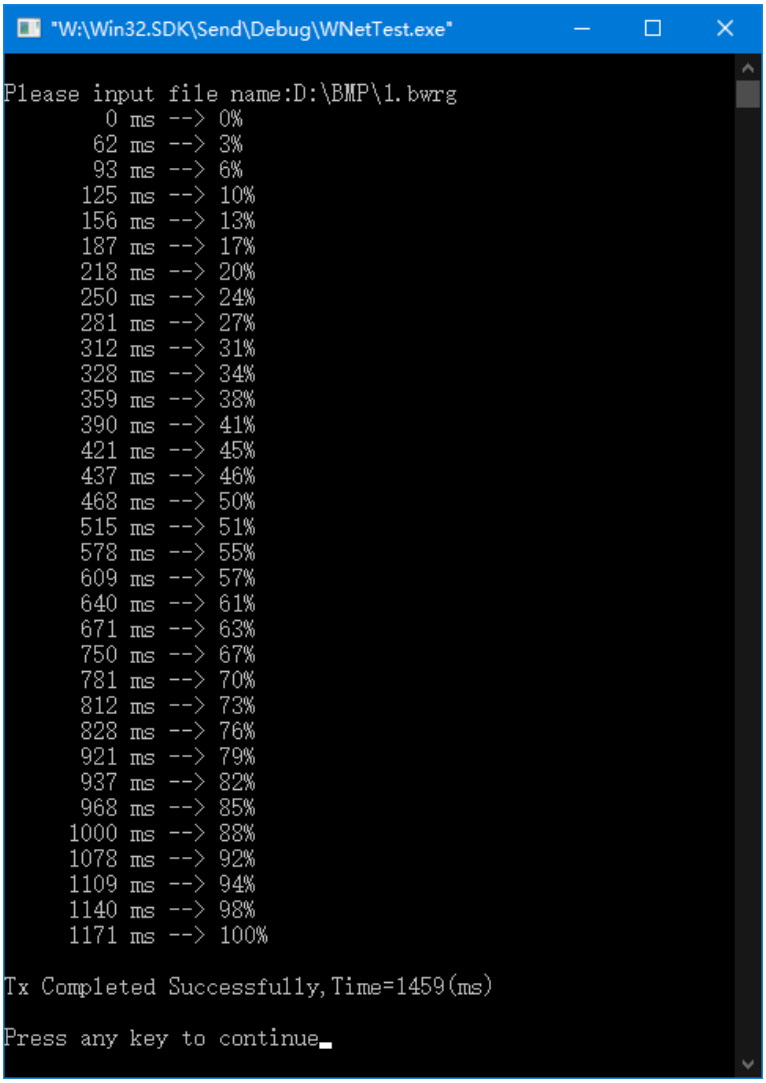
```
#include <stdio.h>
#include <stdlib.h>
#include "API-WiMinet.h"

int main( int argc, char* argv[] )
{
    char iRetVal;
    char * pBuffer;
    FILE * pFile;
    char     pFileName[1024];
```

```
unsigned char iStatus;
unsigned char iProgress;
unsigned long dwSize;
unsigned long dwTimerA;
unsigned long dwTimerB;
unsigned long dwTimerX;

// COM port interface
iRetVal = OpenWiMinetShell( "COM3",115200, 0X01 );

// Ethernet interface
//iRetVal = OpenWiMinetShell( "192.168.0.240",12580, 0X01 );

// Validate the shell open interface
if ( !iRetVal )
{
    printf( "Open shell failed!\r\n" );
    return 0X00;
}

// Query the TxStatus
while ( 0X01 )
{
    // Query the Tx status
    QueryTxMsgStatus( 0X00, ( char * )&iStatus );

    // Check if end of the Tx procedure
    if ( iStatus == TXD_TASK_STATUS_JOB_WAITING )
    {
        break;
    }

    // Waiting the task is free
    printf( "." );

    // Release the control of the processor
    Sleep( 0X01 );
}

// The input file name
printf( "\r\nPlease input file name:" );

// Clear the string
memset( pFileName, 0X00, sizeof( pFileName ) );

// The input file name
scanf( "%s", pFileName );

// Open the file
pFile = fopen( pFileName, "rb" );

// Validate the file pointer
if ( !pFile )
{
    printf( "Failed To Open File!\r\n" );
```

```
}

// Seek to the file end
fseek( pFile, 0X00, SEEK_END );

// The file size
dwSize = ftell( pFile );

// Seek to the file head
fseek( pFile, 0X00, SEEK_SET );

// allocate the buffer
pBuffer = ( char * )malloc( dwSize );

// Read this file contents
fread( pBuffer, 0X01, dwSize, pFile );

// Close this file
fclose( pFile );

// Send out this packet
SendHiQoSMessage( 0X00, ( short )0XA117, pBuffer, dwSize );

// delete this buffer
free( pBuffer );

// The initial time counter
dwTimerA = GetTickCount();

// The default Tx progress
iProgress = 0XFF;

// Query the TxStatus
while ( 0X01 )
{
    // Release the control of the processor
    Sleep( 0X01 );

    // Query the Tx status
    QueryTxMsgStatus( 0X00, ( char * )&iStatus );

    // Check if end of the Tx procedure
    if ( iStatus & TXD_TASK_STATUS_REPORTTABLE )
    {
        break;
    }

    // Compare the progress value
    if ( iStatus == iProgress )
    {
        continue;
    }

    // Current time counter
    dwTimerB = GetTickCount();
```

```
        // The offset timer
        dwTimerX = dwTimerB - dwTimerA;

        // Update the progress value
        iProgress = iStatus;

        // Current Tx progress
        printf( "%9lu ms --> %d%%\r\n", dwTimerX, iStatus );
    }

    // Query the Tx timer
    GetTxMessageTime( 0X00, &dwTimerX );

    // The task completed status
    switch ( iStatus )
    {
    case TXD_TASK_STATUS_END_SUCCESS:
        {
            printf( "\r\nTx Completed Successfully,Time=%lu(ms)\r\n\r\n", dwTimerX );
        }
        break;

    case TXD_TASK_STATUS_END_FAILURE:
        {
            printf( "\r\nTx Completed With Error,Time=%lu(ms)\r\n\r\n", dwTimerX );
        }
        break;

    case TXD_TASK_STATUS_END_NOCRC32:
        {
            printf( "\r\nTx Completed Without CRC32,Time=%lu(ms)\r\n\r\n", dwTimerX );
        }
        break;

    default:
        {
            printf( "\r\nTx Completed With Unknown Error,Time=%lu(ms)\r\n\r\n", dwTimerX );
        }
        break;
    }

    // Stop the shell
    StopWiMinetShell( 0X00 );

    // Exit this main program
    return 0X01;
}
```

5.8 测试程序说明



程序基本的逻辑如下：

(1) 查询发送队列，是否处于空闲状态，如果没有需要等待当前发送任务完成

(2) 提示输入需要传输的文件名称

(3) 读取文件的总长度

(4) 分配相应长度的内存块

(5) 将文件内容加载到内存块

(6) 关闭文件

(7) 将内存块中的内容提交给发送队列

(8) 释放之前分配的内存块

(9) 查询发送状态，打印发送时间和进度信息

(10)查询发送任务是否已经结束

(11)打印发送的结果信息

6. 接收无线数据

6.1 查询接收状态

函数名	**char QueryRxMsgStatus** （char iShell, char * pStatus）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	char * pStatus	指向一个已分配好实体内存空间的单字节变量，用于记录接收状态。 详细定义见“**接收状态**”表所示。
返回值	0X01=操作成功，0X00=操作失败	

数值名称	接收状态说明
0XFF	系统空闲，没有接收任务
0-100	正在接收报文，当前数值为接收的百分进度比，比如 35 代表接收了总长度 35%的报文
0X81	接收成功，且已经附加了前导描述信息，报文总长度，32 位的 CRC 等校验信息
0X82	接收失败
0X83	接收成功，但是没有附加前导描述信息

6.2 读取报文发送站点

函数名	**char GetRxMessageNode** （char iShell， short * pNode）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	short * pNode	指向一个已经分配好实体内存空间的双字节变量，用于记录报文的原始发送站点
返回值	0X01=操作成功，0X00=操作失败	

6.3 读取接收报文属性

函数名	**char GetRxMessageAttr** （char iShell, char * pAttr）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	char * pAttr	指向一个已经分配好实体内存空间的单字节变量，用于记录报文的属性
返回值	0X01=操作成功，0X00=操作失败	

6.4 读取接收报文长度

函数名	**char GetRxMessageSize** （char iShell, unsigned long * pSize）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	unsigned long * pSize	指向一个已经分配好实体内存空间的四字节变量，用于记录报文的长度
返回值	0X01=操作成功，0X00=操作失败	

6.5 读取接收报文内容

函数名	**char ReadInputMessage** （char iShell, char * pBuffer, long dwSize）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	char * pBuffer	指向一个已经分配好实体内存空间的内存块地址，该内存块用于保存需要发送的数据
参数三	long dwSize	内存块中的数据长度
返回值	0X01=操作成功，0X00=操作失败	

6.6 测试例程：接收远程节点的数据

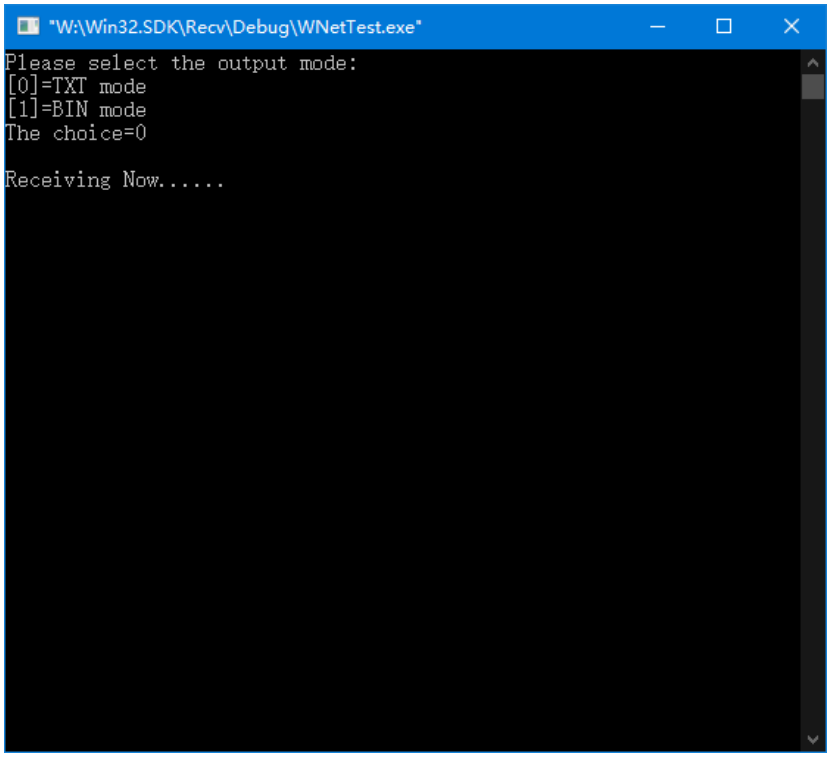
```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include "API-WiMinet.h"

int main( int argc, char* argv[] )
{
    char iRetVal;
    char * pBuffer;
    unsigned short iSource;
    unsigned char iAttr;
    unsigned char iStatus;
    unsigned char iProgress;
    unsigned long dwBinMode;
    unsigned long dwTimerA;
    unsigned long dwTimerB;
    unsigned long dwTimerX;
    unsigned long dwIndex;
    unsigned long dwSize;

    // COM port interface
    iRetVal = OpenWiMinetShell( "COM3",115200, 0X01 );

    // Ethernet interface
    //iRetVal = OpenWiMinetShell( "192.168.0.240",12580, 0X01 );

    // Validate the shell open interface
    if ( !iRetVal )
    {
        printf( "Open shell failed!\r\n" );
        return 0X00;
    }

    // The output mode
    printf( "Please select the output mode:\r\n" );
    printf( "[0]=TXT mode\r\n" );
    printf( "[1]=BIN mode\r\n" );
    printf( "The choice=" );
    scanf( "%d", &dwBinMode );

    // The default progrss
    iProgress = 0XFF;

    // The exit information
    printf( "\r\nReceiving Now.....\r\n" );

    // Get the user input key
    while ( !_kbhit() )
    {
        // Release the processor control
        Sleep( 0X01 );

        // Query the Rx status
        QueryRxMsgStatus( 0X00, ( char * )&iStatus );
```

```
// No received packet arrived
if ( iStatus == RXD_TASK_STATUS_JOB_WAITING )
{
    // Reset the initial timer
    dwTimerA = GetTickCount();
    continue;
}

// Check if ready to report
if ( !( iStatus & RXD_TASK_STATUS_REPORTTABLE ) )
{
    // Update the receive progress
    if ( iStatus != iProgress )
    {
        // Update the receive progress
        iProgress = iStatus;

        // Current timer
        dwTimerB = GetTickCount();

        // The timer offset
        dwTimerX = dwTimerB - dwTimerA;

        // Current Tx progress
        printf( "%9lu ms --> %d%%\r\n", dwTimerX, iStatus );
    }
    continue;
}

// The Rx timer
GetRxMessageTime( 0X00, &dwTimerX );

// The receive status
switch ( iStatus )
{
case RXD_TASK_STATUS_END_SUCCESS:
    {
        printf( "\r\nRx Completed Successfully,Time=%lu(ms)\r\n\r\n", dwTimerX );
    }
    break;

case RXD_TASK_STATUS_END_FAILURE:
    {
        printf( "\r\nRx Completed With Error,Time=%lu(ms)\r\n\r\n", dwTimerX );
    }
    break;

case RXD_TASK_STATUS_END_NOCRC32:
    {
        printf( "\r\nRx Completed Without CRC32,Time=%lu(ms)\r\n\r\n", dwTimerX );
    }
    break;

default:
    {
```

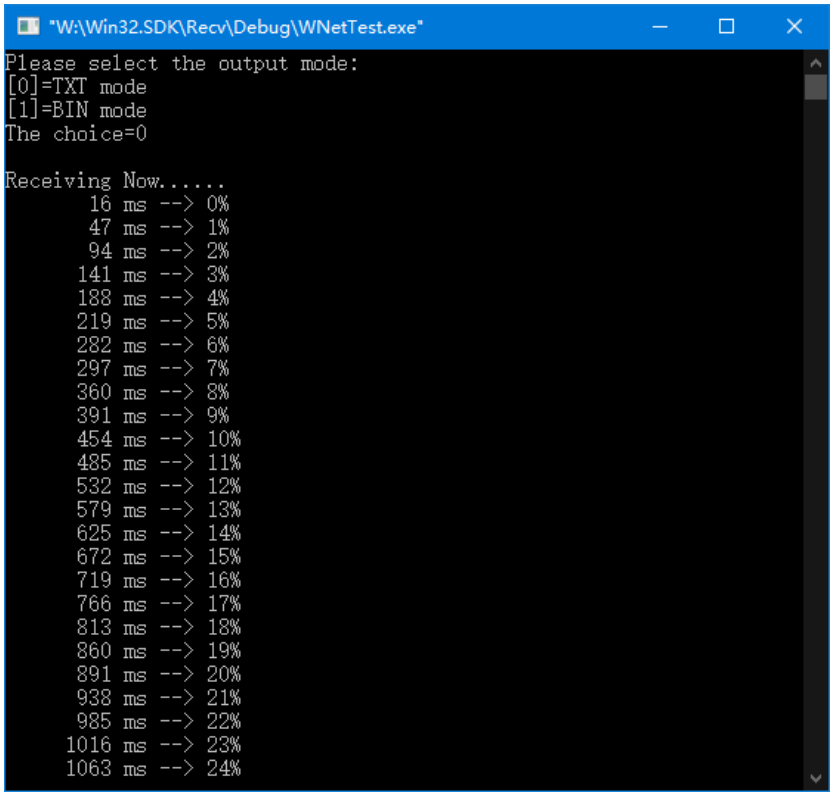
```
                printf( "\r\nTx Completed With Unknown Error,Time=%lu(ms)\r\n\r\n", dwTimerX );
            }
            break;
        }

        // The source node address
        GetRxMessageNode( 0X00, ( short * )&iSource );

        // The packet attribute
        GetRxMessageAttr( 0X00, ( char * )&iAttr );

        // The packet size
        GetRxMessageSize( 0X00, &dwSize );

        // Allocate the buffer for the message
        pBuffer = ( char * )malloc( dwSize );

        // Read this message
        ReadInputMessage( 0X00, pBuffer, dwSize );

        // Report the message: source node address
        printf( "\r\nSource=0X%04X\r\n", iSource );

        // Report the message: attribute
        printf( "Attribute=0X%02X\r\n", iAttr );

        // Report the message: size
        printf( "Size=%lu bytes\r\n", dwSize );

        // Report the message: contents
        printf( "Contents:\r\n\r\n" );

        // Check if print mode
        if ( dwBinMode )
        {
            // The byte contents of the input buffer
            for ( dwIndex = 0X00; dwIndex < dwSize; dwIndex++ )
            {
                printf( "%02X ", (unsigned char)pBuffer[dwIndex] );
            }
        }
        else
        {
            printf( "%s\r\n", pBuffer );
        }

        // free the buffer
        free( pBuffer );
    }

    // Stop the shell
    StopWiMinetShell( 0X00 );

    // Exit this main program
    return 0X01;
}
```

6.7 测试程序说明

首先询问收到报文之后的内容显示格式，输入 0 代表输入为字符串型可打印字符，以文本格式显示；输入 1 位二进制不可打印字符，以十六进制显示。(如下图所示)

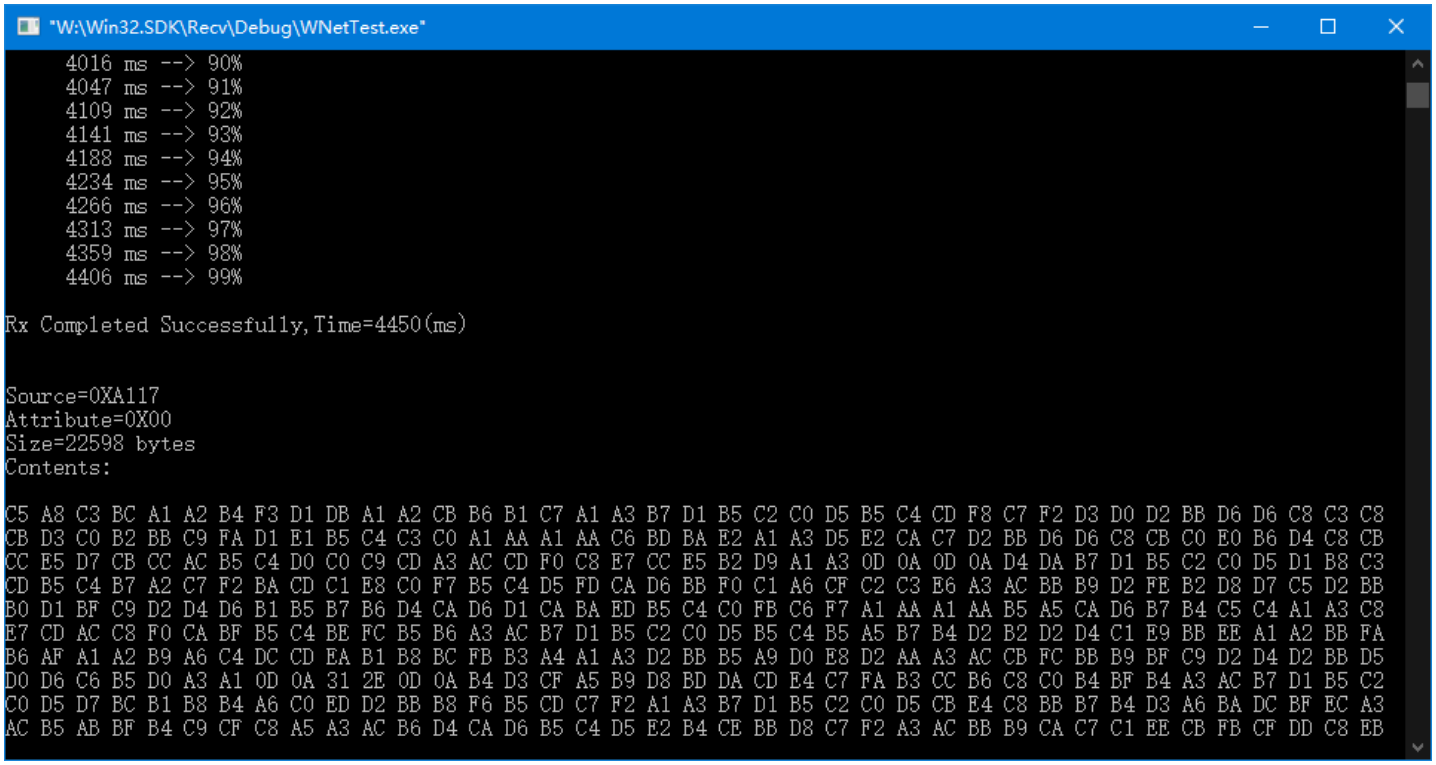


在开始接收之后，会显示时间和接收进度的信息。(如下图所示)



二进制格式显示模式：接收完成之后，以显示接收的结果，原站点地址，报文属性，报文大小；最后以二进制格

式显示报文内容。(如下图所示)



文本格式显示模式：接收完成之后，以显示接收的结果，原站点地址，报文属性，报文大小；最后以文本格式显示报文内容。(如下图所示)

7. 电磁波唤醒节点

7.1 发送电磁波唤醒报文

函数名	**char SetWakeupRequest** （unsigned char iShell, short iObject, short iTime, char iAck, char * pBuffer, char iSize）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	short iObject	需要唤醒的目标节点地址
参数三	short iTime	目标节点被唤醒之后，再次休眠之前需要驻留的时间，单位毫秒； 如果超过了该时间长度，而且后续没有下发任何数据报文，目标节点将会重新进入睡眠状态。
参数四	char iAck	是否需要目标节点做出应答
参数五	char * pBuffer	指向一个已经分配好实体内存空间的内存块，该内存块中的内容会在唤醒报文中附加过去
参数六	long dwSize	内存块中的数据长度，超过了可以附加的最大长度，会被截尾，最小长度为 9 字节
返回值	0X01=操作成功，0X00=操作失败	

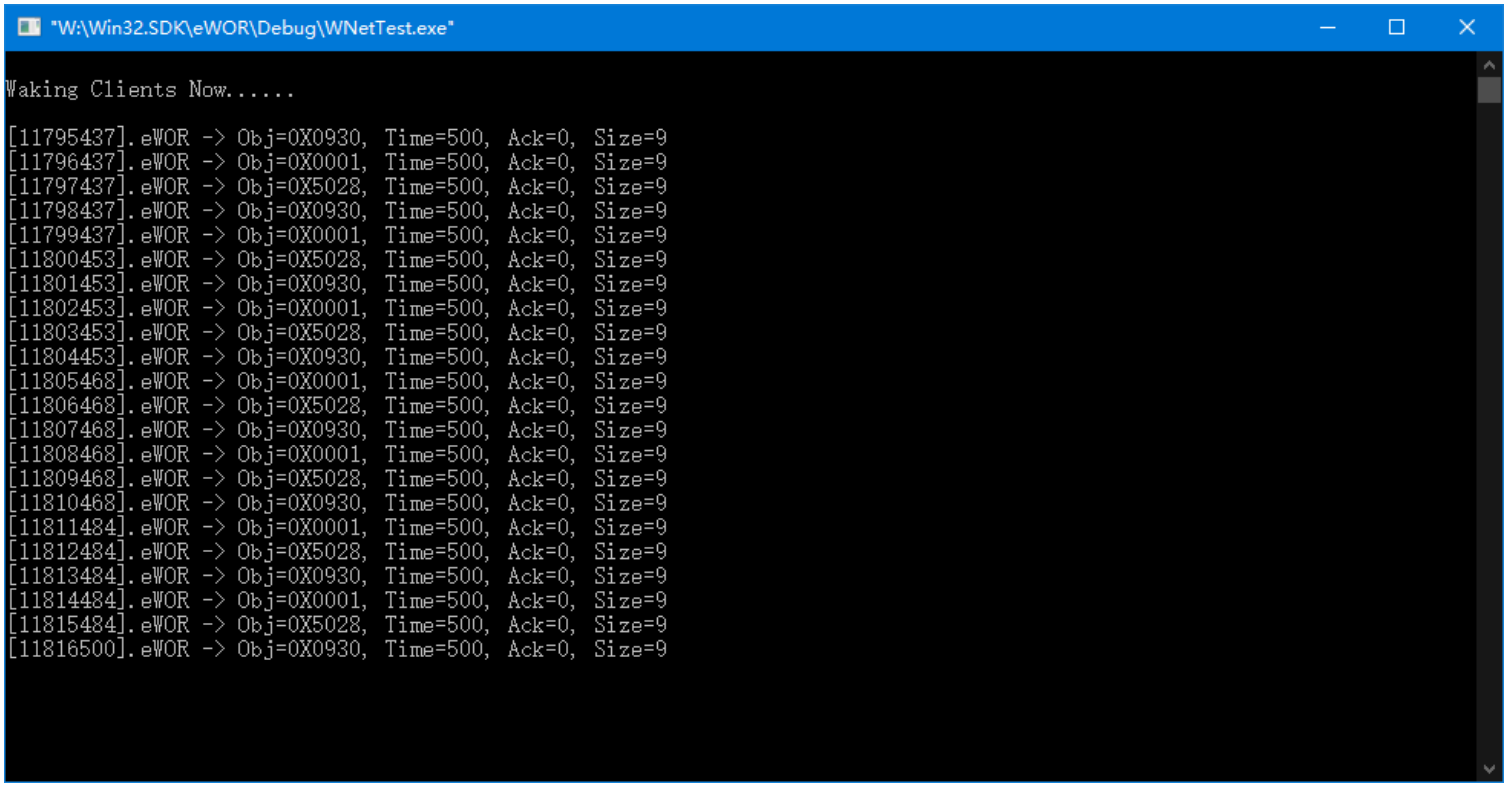
7.2 测试例程：电磁波唤醒目标节点

```
#include <stdio.h>
#include <conio.h>
#include "API-WiMinet.h"

int main( int argc, char* argv[] )
{
    char iRetVal;
    unsigned char iAck;
    unsigned char iSize;
    unsigned short iTime;
    unsigned short iObject;
    char buffer[0XFF];
    unsigned char index;

    // The wakeup object table
    short pObject[] = {
        ( short )0X0930,
        ( short )0X0001,
        ( short )0X5028 };
```

```
// COM port interface
iRetVal = OpenWiMinetShell( "COM3",115200, 0X01 );

// Ethernet interface
//iRetVal = OpenWiMinetShell( "192.168.0.240",12580, 0X01 );

// Validate the shell open interface
if ( !iRetVal )
{
    printf( "Open shell failed!\r\n" );
    return 0X00;
}

// The exit information
printf( "\r\nWaking Clients Now.....\r\n\r\n" );

// The keep timer for client after wakeup, unit is minisecond
iTime = 500;

// NOT need ack from the client
iAck = 0X00;

// The buffer contents
memset( buffer, 0XAA, sizeof( buffer ) );

// The buffer size
iSize = 0X09;

// The initial item index
index = 0X00;

// The wakeup exit information
while ( !_kbhit() )
{
    // The task interval
    Sleep( 1000 );

    // Validate the object index
    if ( index >= sizeof( pObject ) / sizeof( pObject[0X00] ) )
    {
        index = 0X00;
    }

    // The object address
    iObject = pObject[index++];

    // Set the wakeup request
    SetWakeupRequest( 0X00, iObject, iTime, iAck, buffer, iSize );

    // The runtime information
    printf(
        "[%lu].eWOR -> Obj=0X%04X, Time=%d, Ack=%d, Size=%lu\r\n",
        GetTickCount(),
        iObject,
```

```
            iTime,
            iAck,
            iSize );
    }

    // Stop the shell
    StopWiMinetShell( 0X00 );

    // Exit this main program
    return 0X01;
}
```

7.3 单元测试例程说明



该程序根据目标地址列表中的地址，间隔一段时间循环发送唤醒请求报文，并打印出唤醒的参数信息。观察目标节点可以看到被唤醒的标志，比如 LED 指示灯亮起，并持续一段时间后再次熄灭。

7.4 测试例程：向休眠节点发送文件

```
#include <conio.h>


#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include "API-WiMinet.h"

// -------------------------------------------------------------------------
// DESCRIPTION:
// -------------------------------------------------------------------------
#define TIME_OUT_TIMER                                          20UL

// -------------------------------------------------------------------------
// DESCRIPTION: Convert from 100ns to mili-second
// -------------------------------------------------------------------------
#define X64_MIS_TIMER_CONST                                     ( 10000UL )

// -------------------------------------------------------------------------
// DESCRIPTION: Convert from 100ns to mili-second, and then to second
// -------------------------------------------------------------------------
#define X64_SEC_TIMER_CONST                                     ( X64_MIS_TIMER_CONST * 1000UL )


// -------------------------------------------------------------------------
// DESCRIPTION:
// -------------------------------------------------------------------------
char * pOutputFileName[] = {
        "D:\\BMP\\1.bwrg.mLZO",
        "D:\\BMP\\2.bwrg.mLZO",
        "D:\\BMP\\3.bwrg.mLZO",
        "D:\\BMP\\4.bwrg.mLZO",
        "D:\\BMP\\5.bwrg.mLZO",
        "D:\\BMP\\6.bwrg.mLZO",
        "D:\\BMP\\7.bwrg.mLZO",
        "D:\\BMP\\8.bwrg.mLZO",
        "D:\\BMP\\9.bwrg.mLZO",
        "D:\\BMP\\10.bwrg.mLZO",
        "D:\\BMP\\11.bwrg.mLZO",
        "D:\\BMP\\12.bwrg.mLZO",
        "D:\\BMP\\13.bwrg.mLZO",
        "D:\\BMP\\14.bwrg.mLZO" };


/*
// -------------------------------------------------------------------------
// DESCRIPTION:
// -------------------------------------------------------------------------
char * pOutputFileName[] = {
    "D:\\BMP\\1.bmp.mLZO",
    "D:\\BMP\\2.bmp.mLZO",
```

```
    "D:\\BMP\\3.bmp.mLZO",
    "D:\\BMP\\4.bmp.mLZO",
    "D:\\BMP\\5.bmp.mLZO",
    "D:\\BMP\\6.mp.mLZO",
    "D:\\BMP\\7.bmp.mLZO",
    "D:\\BMP\\8.bmp.mLZO",
    "D:\\BMP\\9.bmp.mLZO",
    "D:\\BMP\\10.bmp.mLZO",
    "D:\\BMP\\11.bmp.mLZO",
    "D:\\BMP\\12.bmp.mLZO",
    "D:\\BMP\\13.bmp.mLZO",
    "D:\\BMP\\14.bmp.mLZO" };
*/


/*
// ------------------------------------------------------------------------
// DESCRIPTION:
// ------------------------------------------------------------------------
char * pOutputFileName[] = {
    "D:\\BMP\\1.bwrg",
    "D:\\BMP\\2.bwrg",
    "D:\\BMP\\3.bwrg",
    "D:\\BMP\\4.bwrg",
    "D:\\BMP\\5.bwrg",
    "D:\\BMP\\6.bwrg",
    "D:\\BMP\\7.bwrg",
    "D:\\BMP\\8.bwrg",
    "D:\\BMP\\9.bwrg",
    "D:\\BMP\\10.bwrg",
    "D:\\BMP\\11.bwrg",
    "D:\\BMP\\12.bwrg",
    "D:\\BMP\\13.bwrg",
    "D:\\BMP\\14.bwrg" };
*/

/*
// ------------------------------------------------------------------------
// DESCRIPTION:
// ------------------------------------------------------------------------
    // The file table
    char * pOutputFileName[] = {
        "D:\\BMP\\1.bmp",
        "D:\\BMP\\2.bmp",
        "D:\\BMP\\3.bmp",
        "D:\\BMP\\4.bmp",
        "D:\\BMP\\5.bmp",
        "D:\\BMP\\6.bmp",
        "D:\\BMP\\7.bmp",
        "D:\\BMP\\8.bmp",
        "D:\\BMP\\9.bmp",
        "D:\\BMP\\10.bmp",
        "D:\\BMP\\11.bmp",
        "D:\\BMP\\12.bmp",
```

```
        "D:\\BMP\\13.bmp",
        "D:\\BMP\\14.bmp" };
*/

// ****************************************************************************
// Design Notes:
// ----------------------------------------------------------------------------
unsigned long Translate_Output_File_Number( unsigned long dwChar )
{
    // Select the file ID
    if ( ( dwChar >= '0' ) && ( dwChar <= '9' ) )
    {
        // Skip off the base value
        dwChar -= '0';
    }
    else if ( ( dwChar >= 'a' ) && ( dwChar <= 'f' ) )
    {
        // Skip off the base value
        dwChar -= 'a';

        // Add the basic value
        dwChar += 0X0A;
    }
    else if ( ( dwChar >= 'A' ) && ( dwChar <= 'F' ) )
    {
        // Skip off the base value
        dwChar -= 'A';

        // Add the basic value
        dwChar += 0X0A;
    }
    else
    {
        dwChar = 0XFFFFFFFF;
    }

    // The new output file number
    return dwChar;
}

// ****************************************************************************
// Design Notes:
// ----------------------------------------------------------------------------
char Validate_Output_File_Number( unsigned long dwFile )
{
    // Validate the file index
    if ( dwFile >= ( sizeof( pOutputFileName ) / sizeof( pOutputFileName[0X00] ) ) )
    {
        return 0X00;
    }

    // The file name
    return 0X01;
}
```

```
// ****************************************************************************
// Design Notes:
// ----------------------------------------------------------------------------
void Wait_Tx_Active_Ready( void )
{
    unsigned char iStatus;

    // Query the TxStatus
    while ( 0X01 )
    {
        // Query the Tx status
        QueryTxMsgStatus( 0X00, ( char * )&iStatus );

        // Check if end of the Tx procedure
        if ( iStatus & TXD_TASK_STATUS_REPORTTABLE )
        {
            break;
        }

        // Waiting the task is free
        printf( "%02X ", iStatus );

        // Release the control of the processor
        Sleep( 0X01 );
    }
}

// ****************************************************************************
// Design Notes:
// ----------------------------------------------------------------------------
void Print_Tx_Task_Status( unsigned long dwIndex )
{
    float fTimer;
    unsigned char iStatus;
    unsigned char iProgress;
    unsigned long dwTimerA;
    unsigned long dwTimerB;
    unsigned long dwTimerX;

    // The initial time counter
    dwTimerA = GetTickCount();

    // The default Tx progress
    iProgress = 0XFF;

    // Query the TxStatus
    while ( 0X01 )
    {
        // Release the control of the processor
        Sleep( 0X01 );

        // Query the Tx status
        QueryTxMsgStatus( 0X00, ( char * )&iStatus );
```

```
    // Check if end of the Tx procedure
    if ( iStatus & TXD_TASK_STATUS_REPORTTABLE )
    {
        break;
    }

    // Compare the progress value
    if ( iStatus == iProgress )
    {
        continue;
    }

    // Current time counter
    dwTimerB = GetTickCount();

    // The offset timer
    dwTimerX = dwTimerB - dwTimerA;

    // Update the progress value
    iProgress = iStatus;

    // Current Tx progress
    printf( "%9lu ms --> %d%%\r\n", dwTimerX, iStatus );
}

// The seconds timer
fTimer = 0.001f * GetTickCount();

// Query the Tx timer
GetTxMessageTime( 0X00, &dwTimerX );

// The task completed status
switch ( iStatus )
{
case TXD_TASK_STATUS_END_SUCCESS:
    {
        printf(
            "\r\n[%.3f]+%lu Tx Completed Successfully,Time=%lu(ms)\r\n\r\n",
            fTimer,
            dwIndex,
            dwTimerX );
    }
    break;

case TXD_TASK_STATUS_END_FAILURE:
    {
        printf(
            "\r\n[%.3f]+%lu Tx Completed With Error,Time=%lu(ms)\r\n\r\n",
            fTimer,
            dwIndex,
            dwTimerX );
    }
    break;
```

```
    case TXD_TASK_STATUS_END_NOCRC32:
        {
            printf(
                "\r\n[%.3f]+%lu Tx Completed Without CRC32,Time=%lu(ms)\r\n\r\n",
                fTimer,
                dwIndex,
                dwTimerX );
        }
        break;

    default:
        {
            printf(
                "\r\n[%.3f]+%lu Tx Completed With Unknown Error,Time=%lu(ms)\r\n\r\n",
                fTimer,
                dwIndex,
                dwTimerX );
        }
        break;
    }
}

// ****************************************************************************
// Design Notes:
// ----------------------------------------------------------------------
char Tx_Message_To_Client( unsigned short iObject, unsigned long dwFile )
{
    char * pBuffer;
    FILE * pFile;
    unsigned long dwSize;

    // The total file size
    dwSize = sizeof( pOutputFileName ) / sizeof( pOutputFileName[0X00] );

    // Regulate the index value
    dwFile %= dwSize;

    // Open the file
    pFile = fopen( pOutputFileName[dwFile], "rb" );

    // Validate the file pointer
    if ( !pFile )
    {
        printf( "Failed To Open File!\r\n" );
        return 0X00;
    }

    // Seek to the file end
    fseek( pFile, 0X00, SEEK_END );

    // The file size
    dwSize = ftell( pFile );
```

```
    // Seek to the file head
    fseek( pFile, 0X00, SEEK_SET );

    // allocate the buffer
    pBuffer = ( char * )malloc( dwSize );

    // Read this file contents
    fread( pBuffer, 0X01, dwSize, pFile );

    // Close this file
    fclose( pFile );

    // Set the message format
    // Parameter-1: Shell Number
    // Parameter-2: Media Type
    //                  0X00 = Common Data
    //                  0X01 = Firmware
    //
    // Parameter-3: Compress Mode
    //                  0X00 = None Compress
    //                  0X01 = Compress Mode
    SetMessageFormat( 0X00, 0X00, 0X01 );

    // The task status
    printf( "[WiMinet TxSize=%lu]:Object=0X%04X\r\n", dwSize, iObject );

    // Send message via TCP on all nodes
    SendHiQoSMessage( 0X00, iObject, pBuffer, dwSize );

    // Send message via UDP on the end node, TCP on other nodes
    //SendMxQoSMessage( 0X00, iObject, pBuffer, dwSize );

    // Send message via UDP on all nodes
    //SendNoQoSMessage( 0X00, iObject, pBuffer, dwSize );

    // delete this buffer
    free( pBuffer );

    // The operation status
    return 0X01;
}

// ****************************************************************************
// Design Notes:
// ----------------------------------------------------------------------------
unsigned char Print_WiMinet_Notice( void )
{
    char buffer[0XFF];
    unsigned char iExit;
    unsigned char iSize;
    unsigned char iError;
    unsigned char iEvent;
    unsigned char iRetVal;
    unsigned short iObject;
```

```
ULARGE_INTEGER qwTimerA;
ULARGE_INTEGER qwTimerB;
WiMinet_TxReport * pTxReport;

// The exit condition
iExit = 0X00;

// The error code
iError = 0X00;

// Update the start timer
GetSystemTimeAsFileTime( ( LPFILETIME )&qwTimerA );

// Wait some timer
while ( iExit != 0X03 )
{
    // Update the start timer
    GetSystemTimeAsFileTime( ( LPFILETIME )&qwTimerB );

    // Get the offset value
    qwTimerB.QuadPart -= qwTimerA.QuadPart;

    // Check the time out value
    if ( qwTimerB.QuadPart >= 0X02UL * X64_SEC_TIMER_CONST )
    {
        // Convert to mini-second timer
        qwTimerB.QuadPart /= X64_MIS_TIMER_CONST;

        // The error status
        printf( "\r\n[WiMinet Notice=%I64u]:Time Out Error\r\n", qwTimerB.QuadPart );
        return 0XFF;
    }

    // The max buffer size
    iSize = sizeof( buffer );

    // Get the wiminet notice
    iRetVal = GetWiMinetNotice( 0X00, &iEvent, &iObject, buffer, &iSize );

    // Check the status
    if ( !iRetVal )
    {
        continue;
    }

    // Convert to mini-second timer
    qwTimerB.QuadPart /= X64_MIS_TIMER_CONST;

    // The notice header
    printf( "\r\n[WiMinet Notice=%I64u]:\r\n", qwTimerB.QuadPart );

    // The event of the object
    switch ( iEvent )
    {
```

```
case WIMINET_EVENT_COMMUTE_END:
    {
        iExit |= 0X02;
        printf( "     [1] Event=Commute Success\r\n" );
        printf( "     [2] Object=0X%04X\r\n", iObject );

        // The TxReport status
        pTxReport = ( WiMinet_TxReport * )buffer;

        // The report status
        printf( "     [3] Size=%d Bytes\r\n", iSize );
        printf( "     [4] Error=0X%02X", pTxReport->m_iQError );

        // The detailed error code: No Header
        if ( pTxReport->m_iQError & WIMINET_IO_REPORT_NO_HEADER )
        {
            printf( "[No Header]" );
        }

        // The detailed error code: Invalid Size
        if ( pTxReport->m_iQError & WIMINET_IO_REPORT_ER_AMOUNT )
        {
            printf( "[Invalid Size]" );
        }

        // The detailed error code: Invalid CRC32
        if ( pTxReport->m_iQError & WIMINET_IO_REPORT_ER_CRCODE )
        {
            printf( "[Invalid CRC32]" );
        }

        // The end of this line
        printf( "\r\n" );

        // The Tx data size
        printf( "     [5] Count=%lu\r\n", pTxReport->m_dwCount );

        // The error code
        iError |= pTxReport->m_iQError;
    }
    break;

case WIMINET_EVENT_COMMUTE_ERR:
    {
        iExit |= 0X02;
        printf( "     [1] Event=Commute Failure\r\n" );
        printf( "     [2] Object=0X%04X\r\n", iObject );

        // The error code
        iError |= 0X10;
    }
    break;

case WIMINET_EVENT_CONNECT_ERR:
```

```
            {
                iExit |= 0X02;
                printf( "     [1] Event=Connect Failure\r\n" );
                printf( "     [2] Object=0X%04X\r\n", iObject );

                // The error code
                iError |= 0X20;
            }
            break;

        case WIMINET_EVENT_WOR_COMMUTE:
            {
                iExit |= 0X01;
                printf( "     [1] Event=eWOR Commute\r\n" );
                printf( "     [2] Object=0X%04X\r\n", iObject );

                // The TxReport status
                pTxReport = ( WiMinet_TxReport * )buffer;

                // The report status
                printf( "     [3] Size=%d Bytes\r\n", iSize );

                // The Tx data size
                printf( "     [4] Count=%lu\r\n", pTxReport->m_dwCount );
            }
            break;

        default:
            {
                // The message code
                printf( "     [1] Code=0X%02X\r\n", iEvent );
                printf( "     [2] Object=0X%04X\r\n", iObject );
            }
            break;
        }
    }

    // The unknown error code
    return iError;
}

// *****************************************************************************
// Design Notes:
// -----------------------------------------------------------------------
unsigned char Print_WiMinet_BATVol( void )
{
    unsigned char iExit;
    unsigned char iAttr;
    unsigned char iStatus;
    unsigned long dwSize;
    unsigned short iSource;
    ULARGE_INTEGER qwTimerA;
    ULARGE_INTEGER qwTimerB;
    WiMinet_RxReport report;
```

```
// The exit condition
iExit = 0X00;

// Update the start timer
GetSystemTimeAsFileTime( ( LPFILETIME )&qwTimerA );

// Wait some timer
while ( !iExit )
{
    // Update the start timer
    GetSystemTimeAsFileTime( ( LPFILETIME )&qwTimerB );

    // Get the offset value
    qwTimerB.QuadPart -= qwTimerA.QuadPart;

    // Check the time out value
    if ( qwTimerB.QuadPart >= 0X02UL * X64_SEC_TIMER_CONST )
    {
        // Convert to mini-second timer
        qwTimerB.QuadPart /= X64_MIS_TIMER_CONST;

        // The error status
        printf( "\r\n[WiMinet Report=%I64u]:Time Out Error\r\n", qwTimerB.QuadPart );
        return 0XFF;
    }

    // Query the Rx status
    QueryRxMsgStatus( 0X00, ( char * )&iStatus );

    // No received packet arrived
    if ( iStatus == RXD_TASK_STATUS_JOB_WAITING )
    {
        continue;
    }

    // Check if ready to report
    if ( !( iStatus & RXD_TASK_STATUS_REPORTTABLE ) )
    {
        continue;
    }

    // Convert to mini-second timer
    qwTimerB.QuadPart /= X64_MIS_TIMER_CONST;

    // The notice header
    printf( "\r\n[WiMinet Report=%I64u]:\r\n", qwTimerB.QuadPart );

    // The source node address
    GetRxMessageNode( 0X00, ( short * )&iSource );

    // The packet size
    GetRxMessageSize( 0X00, &dwSize );
```

```
// The packet attribute
GetRxMessageAttr( 0X00, ( char * )&iAttr );

// Clear this object
memset( &report, 0X00, sizeof( report ) );

// Read this message
ReadInputMessage( 0X00, ( char * )&report, sizeof( report ) );

// The report status
printf( "     [1] Size=%lu\r\n", dwSize );
printf( "     [2] Attr=0X%02X\r\n", iAttr );
printf( "     [3] Source=0X%04X\r\n", iSource );
printf( "     [4] Device=0X%02X(%d)\r\n", report.m_iDevice, report.m_iDevice );
printf( "     [5] Version=0X%08lX", report.m_dwBuild );

// The build version for firmware update
printf(
    " --> [20%u.%u.%u+R%u]\r\n",
    ( unsigned char )( report.m_dwBuild >> 0X18 ),
    ( unsigned char )( report.m_dwBuild >> 0X10 ),
    ( unsigned char )( report.m_dwBuild >> 0X08 ),
    ( unsigned char )( report.m_dwBuild >> 0X00 ) );

// The received byte size by the remote endpoint
dwSize = ( ( unsigned long )report.m_iMBSize << 0X10 ) + report.m_iLBSize;

// The client received byte size
printf( "     [6] Counter=%lu Bytes\r\n", dwSize );

// The battery voltage
printf( "     [7] Battery=%.3f(V)\r\n", report.m_iBATVol * 0.001f );

// The RxRSSI voltage
printf( "     [8] RxRSSI=%d(dBm)\r\n", report.m_iRxRSSI );

// The RxRSSI voltage
printf( "     [9] TxRSSI=%d(dBm)\r\n", report.m_iTxRSSI );

// The client receive error code
printf( "     [A] Error=0X%02X", report.m_iQError );

// The detailed error code: Rx Time Out
if ( report.m_iQError & WIMIET_IO_REPORT_RX_TIMEOUT )
{
    printf( " [Rx Time Out]" );
}

// The detailed error code: No Header
if ( report.m_iQError & WIMINET_IO_REPORT_NO_HEADER )
{
    printf( "[No Header]" );
}
```

```
        // The detailed error code: Invalid Size
        if ( report.m_iQError & WIMINET_IO_REPORT_ER_AMOUNT )
        {
            printf( "[Invalid Size]" );
        }

        // The detailed error code: Invalid CRC32
        if ( report.m_iQError & WIMINET_IO_REPORT_ER_CRCODE )
        {
            printf( "[Invalid CRC32]" );
        }

        // The end of this line
        printf( "\r\n\r\n" );

        // The error code
        return report.m_iQError;
    }

    // The unknown error code
    return 0X80;
}

// ****************************************************************************
// Design Notes:
// ----------------------------------------------------------------------
int main( int argc, char* argv[] )
{
    char iRetVal;
    char iAutoMode;
    char iTRequest;
    char iShiftBMP;
    unsigned char   iErrorA;
    unsigned char   iErrorB;
    unsigned long   dwChar;
    unsigned long   dwFile;
    unsigned long   dwThis;
    unsigned long   dwSize;
    unsigned long   dwIndex;
    unsigned short iObject;
    ULARGE_INTEGER qwTimerA;
    ULARGE_INTEGER qwTimerB;
    unsigned short pObject[] = { 0X34F1 };   //0X0930   0X5028

    // COM port interface
    //iRetVal = OpenWiMinetShell( "COM6",115200, 0X01 );

    // Ethernet interface
    //iRetVal = OpenWiMinetShell( "192.168.0.240",12580, 0X01 );

    // Ethernet interface
    iRetVal = OpenWiMinetShell( "192.168.1.240",12580, 0X01 );

    // Validate the shell open interface
```

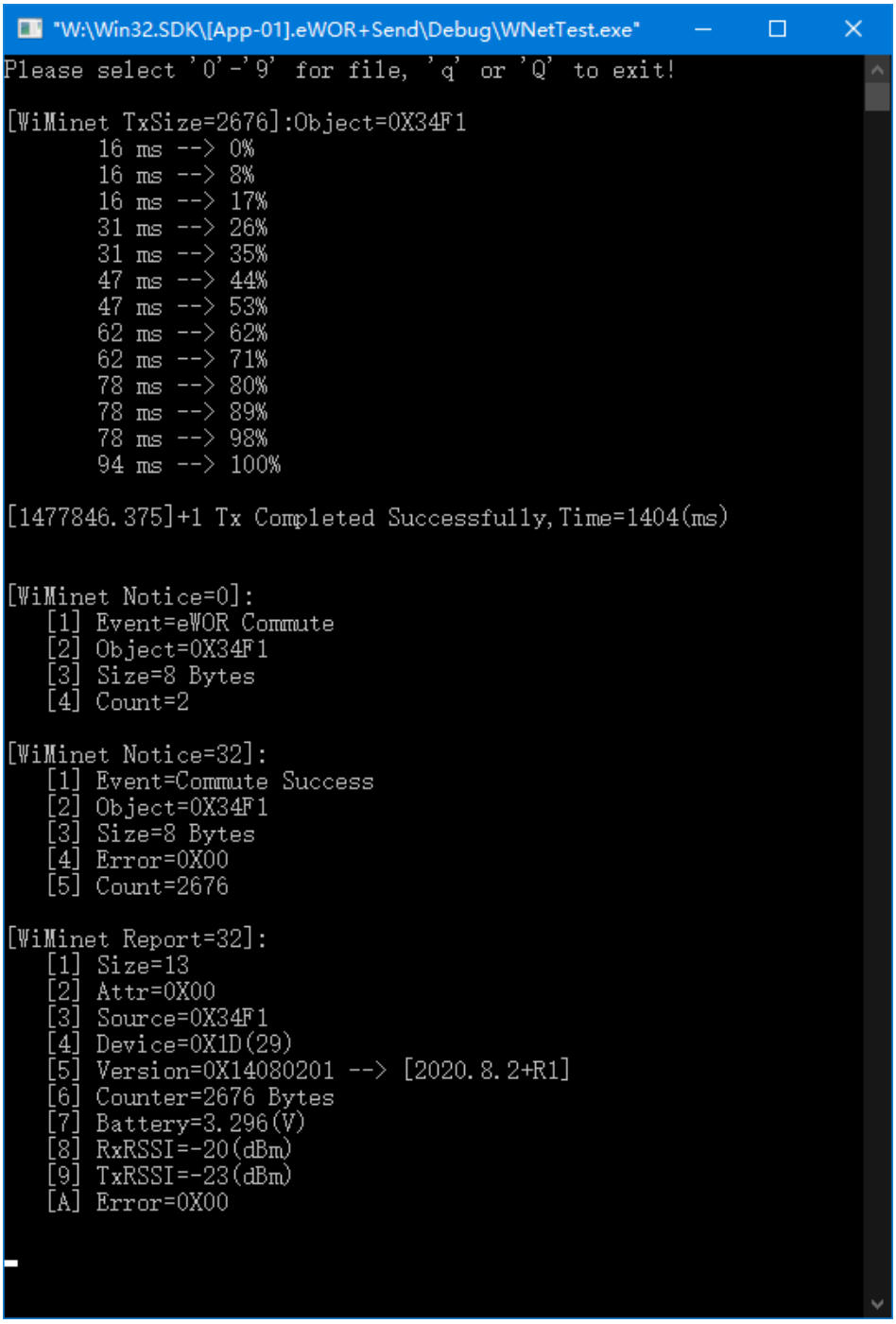
```
if ( !iRetVal )
{
    printf( "Open shell failed!\r\n" );
    return 0X00;
}

// The default Tx status report
SetTxStateReport( 0X00, 0X01 );

// The Tx performance
SetTxPerformance( 0X00, 0X05 );

// The notice for user input
printf( "Please select '0'-'9' for file, 'q' or 'Q' to exit!\r\n\r\n" );

// Auto mode or step mode
iAutoMode = 0X00;

// The initial request status
iTRequest = 0X00;

// The BMP status
iShiftBMP = 0X00;

// The main service
while ( 0X01 )
{
    // Release the processor control
    Sleep( 0X01 );

    // Check the keyboard input
    if ( _kbhit() )
    {
        // Get the input character
        dwChar = _getch();

        // Check if time to exit this process
        if ( ( dwChar == 'q' ) || ( dwChar == 'Q' ) )
        {
            break;
        }
        else if ( ( dwChar == 't' ) || ( dwChar == 'T' ) )
        {
            iAutoMode = 0X01;
            printf( "CMD:Auto Mode\r\n" );
            continue;
        }
        else if ( ( dwChar == 'm' ) || ( dwChar == 'M' ) )
        {
            iAutoMode = 0X00;
            printf( "CMD:Manual Mode\r\n" );
            continue;
        }
        else if ( ( dwChar == 'y' ) || ( dwChar == 'Y' ) )
```

```
        {
            iShiftBMP = 0X01;
            printf( "CMD:Dynamic BMP\r\n" );
            continue;
        }
        else if ( ( dwChar == 's' ) || ( dwChar == 'S' ) )
        {
            iShiftBMP = 0X00;
            printf( "CMD:Static BMP\r\n" );
            continue;
        }

        // Translate the input character
        dwFile = Translate_Output_File_Number( dwChar );

        // The request counter
        dwIndex = 0X00;

        // Start the new request
        iTRequest = 0X01;

        // Get the start timer
        GetSystemTimeAsFileTime( ( LPFILETIME )&qwTimerA );

        // Cut off the initial timer
        qwTimerA.QuadPart -= ( TIME_OUT_TIMER * X64_SEC_TIMER_CONST );
    }

    // Check if there is a new request
    if ( !iTRequest )
    {
        continue;
    }

    // Get the output file name
    if ( !Validate_Output_File_Number( dwFile ) )
    {
        // The error message
        printf( "Invalid File Number\r\n" );

        // Clear the request
        iTRequest = 0X00;
        continue;
    }

    // Get current timer
    GetSystemTimeAsFileTime( ( LPFILETIME )&qwTimerB );

    // Check if in auto mode
    if ( iAutoMode )
    {
        // Get the offset value
        qwTimerB.QuadPart -= qwTimerA.QuadPart;
```

```
    // Convert 100ns to miliseconds, then seconds
    qwTimerB.QuadPart /= X64_SEC_TIMER_CONST;

    // Validate the timeout timer
    if ( qwTimerB.QuadPart < TIME_OUT_TIMER )
    {
        continue;
    }
}

// Wait for the Tx Ready
Wait_Tx_Active_Ready();

// The counter of the object
dwSize = sizeof( pObject ) / sizeof( pObject[0X00] );

// The object index value
iObject = pObject[dwIndex % dwSize];

// Set the Wakeup Request: Parameter-4=iAck
// 0X00: Dont Send Report
// 0X01: Need Send Report
//
// ++++++ IMPORTANT NOTES ++++++
// (1) if iACK=0, then client will never send out the UDP report
// (2) if iACK=0, then Print_WiMinet_BATVol will encounter time out error
//
SetWakeupRequest( 0X00, iObject, 5000, 0X01, NULL, 0X00 );

// The file index
dwThis = dwFile;

// Check if in auto mode and dynamic BMP
if ( iShiftBMP && iAutoMode )
{
    dwThis += dwIndex;
}

// Send File to Remote Client
iRetVal = Tx_Message_To_Client( iObject, dwThis );

// Validate the Operation Result
if ( !iRetVal )
{
    // Clear the request
    iTRequest = 0X00;
    continue;
}

// The request counter
dwIndex++;

// Print the Tx Task status
Print_Tx_Task_Status( dwIndex );
```

```
        // The wiminet notice
        iErrorA = Print_WiMinet_Notice();

        // The client voltage
        iErrorB = Print_WiMinet_BATVol();

        // Check if there are errors
        if ( iErrorA || iErrorB )
        {
            // Clear the auto mode
            iAutoMode = 0X00;

            // Report the error status
            printf( "Error:A=0X%02X,B=0X%02X\r\n", iErrorA, iErrorB );
        }

        // Check if in auto mode
        if ( !iAutoMode )
        {
            // Clear the request
            iTRequest = 0X00;
            continue;
        }

        // Update the start timer
        GetSystemTimeAsFileTime( ( LPFILETIME )&qwTimerA );
    }

    // Stop the shell
    StopWiMinetShell( 0X00 );

    // Exit this main program
    return 0X01;
}
```

7.5 综合测试例程说明



该程序主要流程如下：

(1) 打开通讯端口

(2) 开启发送状态报告，设置 UDP 的发送等级为 5

(3) 输入文件序号：0-9，从 10 个磁盘文件中选择一个用于传输测试，按键 q 或者 Q 退出

(4) 向目标节点发送唤醒指令，目标节点 ID 保存在双字节变量 iObject 中

(5) 以 TCP，UDP 或者 TCP/UDP 方式向目标节点传送文件

(6) 检查报文发送的状态，等待发送结束

(7) 读取系统通知消息，检查发送的状态报告

(8) 读取目标节点的报告信息，打印电池电量和接收状态报告

特别说明：

(1) 如果需要基站送出发送状态报告，需要通过函数 SetTxStateReport 来开启，DLL 初始化的时候默认是开启的。如果关闭状态报告，函数会超时退出

(2) 如果需要从站节点接收到数据之后，上报电池电压和接收状态，需要在函数 SetWakeupRequest 的第四个参数 iAck 中填写 0X01，通知从站上报，如果填写 0X00，从站就不报告电池电压和接收状态，此时 Print_WiMinet_BATVol 会超时退出。

(3) 检查 WiMinetNotice 的通知消息的时候，有两类消息需要检查，第一类是 eWOR 的电磁波唤醒确认消息；第二类是 TCP/UDP 发送的通知消息，而第二类消息又有三种结果；第一种是连接失败消息（WIMINET_EVENT_CONNECT_ERR）；第二种连接成功之后通讯失败（WIMINET_EVENT_COMMUTE_ERR）；第三种是链接之后通讯也成功（WIMINET_EVENT_COMMUTE_END）。

8. 技术支持

邮箱：dingyg99@126.com

电话：13911821802 （微信同号）

座机：010-57222007

企业微信公众号