WiMi-net 无线自组网管理平台

Win32 SDK 说明书

（中级版）

微网高通（北京)无线技术有限公司

目 录

1. BMP 文件格式转换

1.1 将 16 灰度 BMP 文件转换为 4 灰度 BMP 文件

函数名	unsigned long Convert_BMP_File (char * pInFile, char * pOutFile, unsigned long * pSize)	
头文件	API-WiMiFile.h	
静态库	WiMiFile.lib	
动态库	WiMiFile.dll	
	形式	说明
参数一	char * pInFile	需要转换的原始 BMP 文件名称
参数二	char * pOutFile	指向一个已分配好实体内存空间的内存块，用于存储转换后的 4 灰度 BMP 文件名，文件的尾缀是 bwrg
参数三	unsigned long * pSize	指向一个已分配好实体内存空间的 4 字节长整型变量，用于保存转换后文件名的有效长度，需要初始化为参数二“pOutFile”的实体内存空间的大小
返回值	0X00=操作成功 0X01=源文件名无效 0X02=打开源文件失败 0X03=源文件不是有效的 BMP 文件 0X04=文件调色板错误，不是 16 灰度的 BMP 文件 0X05=打开 4 灰度的 bwrg 文件失败，可能是用户操作权限不够	

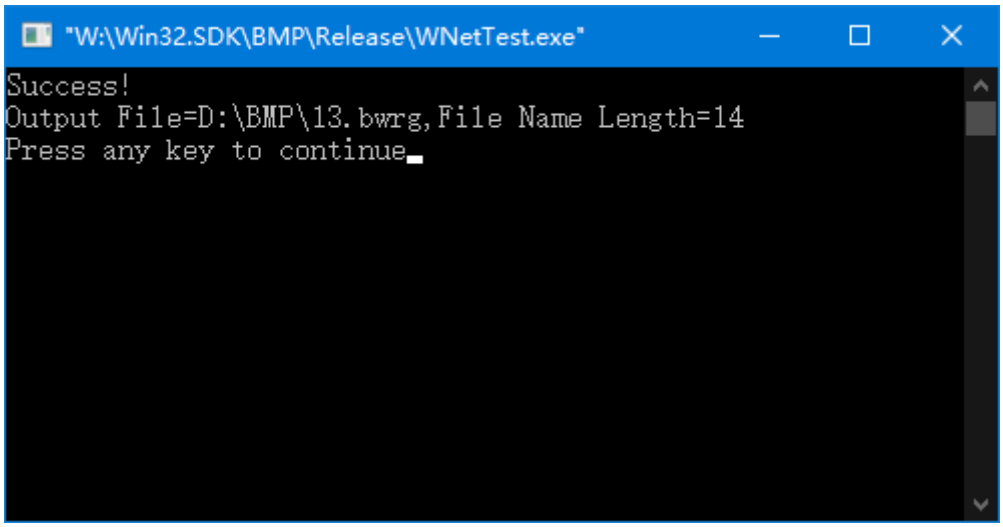
1.2 设置目标 BMP 文件的颜色位数

函数名	void SetBMPImageColor （unsigned char iColor）	
头文件	API-WiMiFile.h	
静态库	WiMiFile.lib	
动态库	WiMiFile.dll	
	形式	说明
参数一	unsigned char iColor	目标 BMP 文件的颜色位数 0X01：1-bit 颜色，仅仅支持黑白两种颜色 0X02：2-bit 颜色，仅仅支持黑色，白色，红色，三种颜色
返回值	无	

1.3 测试例程：BMP 文件格式转换

```
#include <stdio.h>
#include "API-WiMiFile.h"

int main( int argc, char* argv[] )
{
    char buffer[0XFF];
    unsigned long dwSize;
    unsigned long dwRetVal;

    // The default input buffer size
    dwSize = sizeof( buffer );

    // Convert the BMP file
    dwRetVal = Convert_BMP_File( "D:\\BMP\\1.BMP", buffer, &dwSize );

    // Validate the operation status
    if ( dwRetVal != ERROR_SUCCESS )
    {
        printf( "Convert BMP File Failed,Error Code=0X%08lX\r\n", dwRetVal );
        return 0X00;
    }

    // The output file name and size
    printf( "Success!\r\nOutput File=%s,File Name Length=%lu\r\n", buffer, dwSize );

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

1.4 测试说明：BMP 文件格式转换



该程序首先转换一个 BMP 文件，如果失败，就打印出失败代码，如果成功，就打印出转换后的文件名称以及文件名称的长度。

2. miniLZO 文件压缩

2.1 设置最大的文件尺寸

函数名	**void SetMaxFileVolume** （unsigned long dwSize）	
头文件	API-WiMiFile.h	
静态库	WiMiFile.lib	
动态库	WiMiFile.dll	
	形式	说明
参数一	Unsigned long dwSize	压缩文件和解压缩文件的最大总尺寸，单位字节
返回值	无	

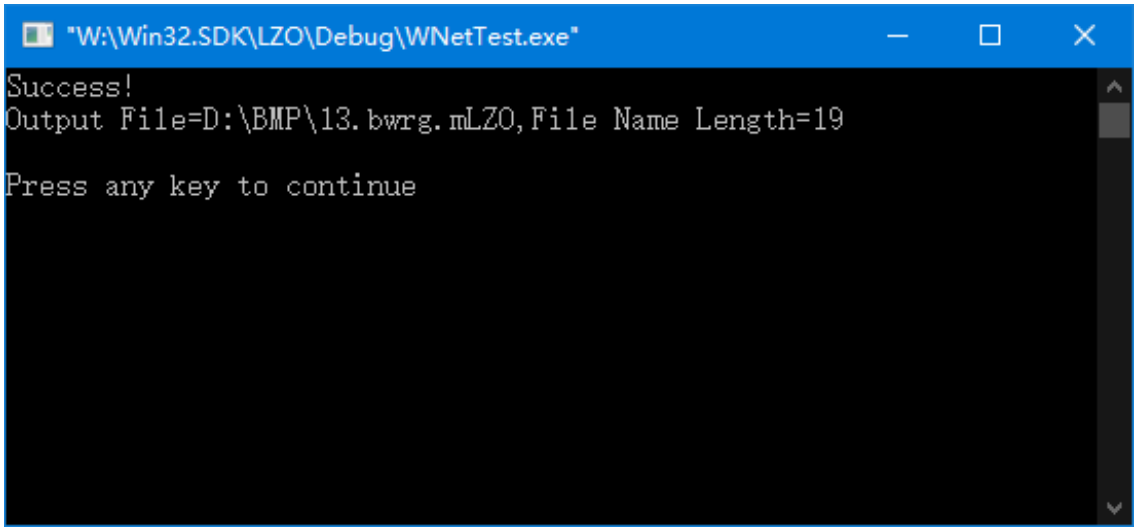
2.2 将文件以 miniLZO 协议压缩

函数名	**unsigned long LZOCompress_File** （char * pInFile， char * pOutFile， unsigned long * pSize）	
头文件	API-WiMiFile.h	
静态库	WiMiFile.lib	
动态库	WiMiFile.dll	
	形式	说明
参数一	char * pInFile	需要压缩的原始文件名称
参数二	char * pOutFile	指向一个已分配好实体内存空间的内存块，用于存储压缩后的文件名，文件的尾缀是 mLZO
参数三	unsigned long * pSize	指向一个已分配好实体内存空间的 4 字节长整型变量，用于保存转换后文件名的有效长度，需要初始化为参数二“pOutFile”的实体内存空间的大小
返回值	0X00=操作成功 0X01=源文件名无效 0X02=打开源文件失败 0X05=打开 mLZO 文件失败，可能是用户操作权限不够 0X06= mLZO 初始化失败 0X07= mLZO 压缩失败 0X08= mLZO 解压缩失败 0X09= mLZO 解压缩尺寸和源文件不匹配 0X0A= mLZO 解压缩内容和源文件不匹配	

2.3 测试例程：miniLZO 文件压缩

```
#include <stdio.h>
#include "API-WiMiFile.h"

int main( int argc, char* argv[] )
{
    char buffer[0XFF];
    unsigned long dwSize;
    unsigned long dwRetVal;

    // The default input buffer size
    dwSize = sizeof( buffer );

    // The max file volume
    SetMaxFileVolume( 12000UL );

    // miniLZO the source file
    dwRetVal = LZOCompress_File( "D:\\BMP\\13.bwrg", buffer, &dwSize );
    //dwRetVal = LZOCompress_File( "D:\\BMP\\5B.bmp", buffer, &dwSize );

    // Validate the operation status
    if ( dwRetVal != ERROR_SUCCESS )
    {
        printf( "Error!\r\nCode=0X%08lX\r\n\r\n", dwRetVal );
        return 0X00;
    }

    // The output file name and size
    printf( "Success!\r\nOutput File=%s,File Name Length=%lu\r\n\r\n", buffer, dwSize );

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

2.4 测试说明：miniLZO 文件压缩



该程序首先设置最大的压缩文件的尺寸为 12K 字节，然后对文件进行 miniLZO 压缩，如果失败，就打印出失败代码，如果成功，就打印出压缩后的文件名称以及文件名称的长度。

3. 设置传输文件的参数

设置传输文件的类型

函数名	**char SetMessageFormat** （char iShell, char iFirmware, char iCompress）	
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 iFirmware	传输文件的类型，0X00 是普通文件，0X01 是目标设备的二进制固件文件，用于固件升级。
参数三	char iCompress	传输文件的压缩格式，0X00 是非压缩格式，0X01 是 mLZO 压缩格式
返回值	固定数值 0X01	

4. 读取注册节点列表

4.1 读取节点的网络模式

函数名	**char GetNetworkWiMode** （char iShell, short iObject, unsigned char * pMode）	
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	通讯的目标节点网络地址，本机地址填写 0X00
参数三	unsigned char * pMode	指向单字节内存地址的指针，用于存放该节点的网络模式
返回值	0X01=操作成功，0X00=操作失败	

4.2 读取节点的 16 位网络地址

函数名	**char GetX16NETAddress** （char iShell, short iObject, unsigned short * pAddr）
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	通讯的目标节点网络地址，本机地址填写 0X00
参数三	unsigned char * pAddr	指向双字节内存地址的指针，用于存放该节点的 16 位网络地址
返回值	0X01=操作成功，0X00=操作失败	

4.3 读取节点的 64 位 MAC 地址

函数名	char GetX64MACAddress （char iShell, short iObject, unsigned char * pX64MAC, 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	short iObject	通讯的目标节点网络地址，本机地址填写 0X00
参数三	unsigned char * pX64MAC	指向至少八字节内存地址的指针，用于存放该节点的 64 位 MAC 地址
参数四	unsigned char iSize	内存地址块 pX64MAC 的最大长度
返回值	0X01=操作成功，0X00=操作失败	

4.4 读取注册表的最大数量

函数名	char GetRegistryCount （char iShell, unsigned short * pCount）
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 short * pCount	指向双字节内存地址块的指针，用于存放注册表最大条目数量
返回值	0X01=操作成功，0X00=操作失败	

4.5 读取注册表的条目

函数名	**char GetRegistryTable** （char iShell, short index, WiMinet_MeshItem * pInfo）	
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 index	注册表的条目序号
参数三	WiMinet_MeshItem * pInfo	注册表的条目内容
返回值	0X01=操作成功，0X00=操作失败	

4.6 测试例程：读取注册节点列表

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

// ********************************************************************************
// Design Notes:
// --------------------------------------------------------------------
char Print_WiMode_Address( void )
{
    char iRetVal;
    unsigned char   iMode;

    // Get the network mode
    iRetVal = GetNetworkWiMode( 0X00, 0X00, &iMode );

    // Validate the shell operation status
    if ( !iRetVal )
    {
        printf( "GetNetworkWiMode failed!\r\n" );
        return 0X00;
    }

    // The node network mode
    printf( "[0] The Node Mode is:" );

    // The WiMode status
    switch ( iMode )
```

```
    {
    case WIMODE_NETCLIENT:
        {
            printf( "Client\r\n" );
        }
        break;

    case WIMODE_NETSERVER:
        {
            printf( "Server\r\n" );
        }
        break;

    case WIMODE_AUTO_TREE:
        {
            printf( "Router\r\n" );
        }
        break;

    default:
        {
        }
        break;
    }

    return 0X01;
}

// *****************************************************************************
// Design Notes:
// -----------------------------------------------------------------------
char Print_X16NET_Address( void )
{
    char iRetVal;
    unsigned short iAddr;

    // Get the X16 NET address
    iRetVal = GetX16NETAddress( 0X00, 0X00, &iAddr );

    // Validate the shell operation status
    if ( !iRetVal )
    {
        printf( "GetX16NETAddress failed!\r\n" );
        return 0X00;
    }
```

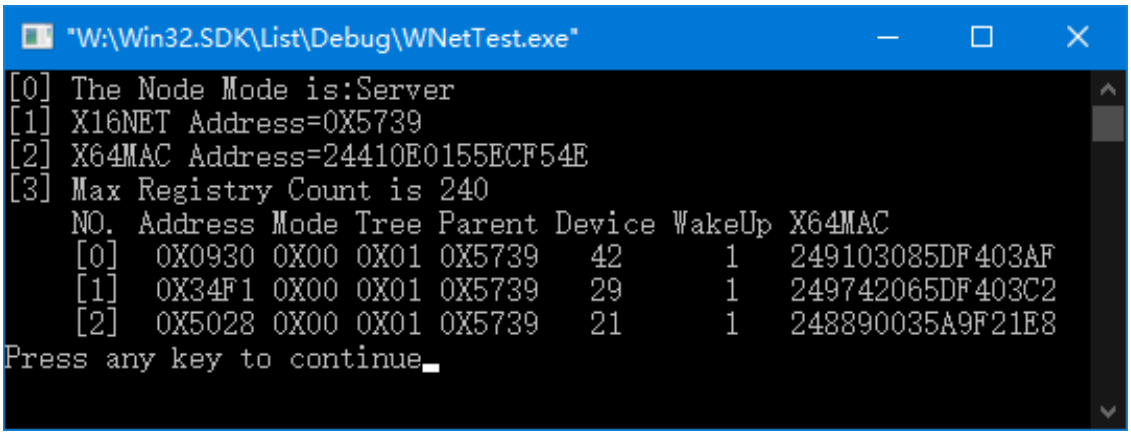
```
    // The X16NET address
    printf( "[1] X16NET Address=0X%04X\r\n", iAddr );
    return 0X01;
}

// *****************************************************************************
// Design Notes:
// -----------------------------------------------------------------------------
char Print_X64MAC_Address( void )
{
    char iRetVal;
    char pX64MAC[0X08];
    unsigned char   index;

    // Get the X64MAC address
    iRetVal = GetX64MACAddress( 0X00, 0X00, pX64MAC, sizeof( pX64MAC ) );

    // Validate the shell operation status
    if ( !iRetVal )
    {
        printf( "GetX64MACAddress failed!\r\n" );
        return 0X00;
    }

    // The X64MAC header
    printf( "[2] X64MAC Address=" );

    // The X64MAC address
    for ( index = 0X00; index < 0X08; index++ )
    {
        printf( "%02X", ( unsigned char )pX64MAC[index] );
    }
    printf( "\r\n" );
    return 0X01;
}

// *****************************************************************************
// Design Notes:
// -----------------------------------------------------------------------------
char Print_Register_Table( void )
{
    unsigned char   iRetVal;
    unsigned char   iStep;
    unsigned short index;
```

```
unsigned short iCount;
WiMinet_MeshItem item;

// The max registry count
iRetVal = GetRegistryCount( 0X00, &iCount );

// Validate the shell operation status
if ( !iRetVal )
{
    printf( "GetRegistryCount failed!\r\n" );
    return 0X00;
}

// The max registry count
printf( "[3] Max Registry Count is %u\r\n", iCount );

// The registry header
printf( "      NO. Address Mode Tree Parent Device WakeUp X64MAC\r\n" );

// Get all the item in this table
for ( index = 0X00; index < iCount; index++ )
{
    // Get the mesh item from the registry table
    GetRegistryTable( 0X00, index, &item );

    // Validate the node address
    if ( !item.m_Address.m_iTxAddr )
    {
        break;
    }

    // The device index
    printf( "      [%d] ", index );

    // The node information
    printf( " 0X%04X", item.m_Address.m_iTxAddr );

    // The node mode
    printf( " 0X%02X", item.m_Address.m_iTxMode );

    // The tree level
    printf( " 0X%02X", item.m_Address.m_iTxTree );

    // The parent information
    printf( " 0X%04X", item.m_RunTime.m_iParent );
```

```
        // The class information
        printf( "     %u     ", item.m_Address.m_iDevice );

        // The eWOR device status
        printf( "     %u     ", item.m_Address.m_iWakeUp );

        // The X64MAC address
        for ( iStep = 0X00; iStep < 0X08; iStep++ )
        {
            printf( "%02X", ( unsigned char )item.m_Address.m_pX64MAC[iStep] );
        }

        // The __I64 format has a revered byte order
        // printf( "     %I64X", *( ( unsigned __int64 * )item.m_Address.m_pX64MAC ) );

        // End of this item
        printf( "\r\n" );
    }
    return 0X01;
}

// ****************************************************************************
// Design Notes:
// ----------------------------------------------------------------------
int main( int argc, char* argv[] )
{
    char iRetVal;

    // COM port interface
    //iRetVal = OpenWiMinetShell( "COM6",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 WiMode status
    Print_WiMode_Address();
```

```
    // The X16NET address
    Print_X16NET_Address();

    // The X64MAC address
    Print_X64MAC_Address();

    // The Register Table
    Print_Register_Table();

    // Stop the shell
    StopWiMinetShell( 0X00 );

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

4.7 测试说明：读取注册节点列表



程序先读取该节点的工作模式，16 位网络地址，64 位网络地址，然后接着读取注册表的最大数量。接下来打印注册表的信息表头，然后依次读取每一条注册表记录信息。

其中 Device 字段是注册到该站点上的节点的类别号码，比如 21 就是 2.13 吋的电子纸设备 EPD，29 就是 2.9 吋的电子纸设备 EPD，42 就是 4.2 吋的电子纸设备 EPD。

5. 事件与消息通知

5.1 设置发送状态报告

函数名	**char SetTxStateReport** （char iShell, char iReport）	
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 iReport	是否需要回读发送状态报告，0X01=需要，0X00=不需要
返回值	0X01=操作成功，0X00=操作失败	

5.2 读取系统的事件与消息通知

函数名	**char GetWiMinetNotice** （char iShell , unsigned char * pEvent , unsigned short * pObject , char * pBuffer , unsigned char * 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 char * pEvent	指向单字节内存地址的指针，用于存放事件号码
参数三	unsigned short * pObject	指向双字节内存地址的指针，用于存放事件对应网络节点号码
参数四	char * pBuffer	用于存放事件描述信息的内存块地址
参数五	unsigned char * pSize	用于存放事件描述信息的长度的指针，需要初始化长度
返回值	0X01=操作成功，0X00=操作失败	

6. 基站信标管理

6.1 设置基站的信标开启与关闭

函数名	char SetWiMinetBeacon （ char iShell， short iObject， char iEnable ）	
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	需要控制的目标基站的网络地址
参数三	char iEnable	基站的信标开启与关闭状态，1=开启，0=关闭
返回值	0X01=操作成功，0X00=操作失败	

6.2 读取基站的信标状态

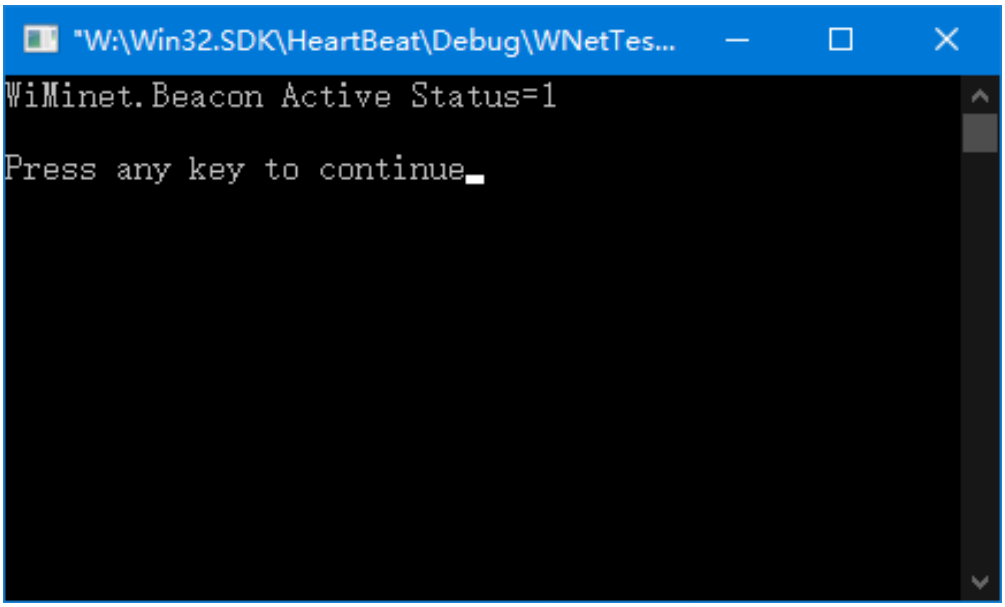
函数名	char GetWiMinetBeacon （char iShell， short iObject， char * pEnable）	
头文件	API-WiMinet.h	
静态库	WiMinet.lib	
动态库	WiMinet.dll	
	形式	说明
参数一	char iShell	通讯端口的编号，填写固定数值 0X00
参数二	short iObject	需要控制的目标基站的网络地址
参数三	char * pEnable	指向单字节的内存变量，用于保存基站的信标开启与关闭状态，1=开启，0=关闭
返回值	0X01=操作成功，0X00=操作失败	

6.3 测试例程：基站信标管理

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

// ********************************************************************************
// Design Notes:
// --------------------------------------------------------------------
int main( int argc, char* argv[] )
{
    char iRetVal;
    char iEnable;

    // COM port interface
    //iRetVal = OpenWiMinetShell( "COM6",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;
    }

    // Set the wiminet beacon status
    iRetVal = SetWiMinetBeacon( 0X00, 0X00, 0X01 );

    // Validate the operation status
    if ( !iRetVal )
    {
        printf( "Error: SetWiMinetBeacon\r\n" );
    }

    // Get the wiminet beacon status
    iRetVal = GetWiMinetBeacon( 0X00, 0X00, &iEnable );

    // Validate the operation status
    if ( !iRetVal )
    {
        printf( "Error: GetWiMinetBeacon\r\n" );
    }

    // Current wiminet beacon active status
    printf( "WiMinet.Beacon Active Status=%d\r\n\r\n", iEnable );

    // Stop the shell
    StopWiMinetShell( 0X00 );

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

6.4 测试说明：基站信标管理



Status=1 表示基站信标状态已开启

7. 基站信道管理

7.1 读取基站的信道配置模式

函数名	**char GetChannelSelect** （char iShell， short iObject， unsigned char * pMode）	
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	需要控制的目标基站的网络地址
参数三	unsigned char * pMode	指向单字节的内存指针，用于保存基站的信道配置模式 0X00：固定不变的信道设置，信道冲突不跳频 0X01：恢复出厂配置时搜索，信道冲突会跳频 0X02~0XFF：每次开机都自动搜索，信道冲突会跳频
返回值	0X01=操作成功，0X00=操作失败	

7.2 设置基站的信道配置模式

函数名	**char SetChannelSelect** （char iShell， short iObject， unsigned char iMode）
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	需要控制的目标基站的网络地址
参数三	unsigned char iMode	基站的信道配置模式 0X00：固定不变的信道设置，信道冲突不跳频 0X01：恢复出厂配置时搜索，信道冲突会跳频 0X02~0XFF：每次开机都自动搜索，信道冲突会跳频
返回值	0X01=操作成功，0X00=操作失败	

7.3 读取当前的工作信道

函数名	char GetModuleChannel （char iShell, short iObject, unsigned char * pBand, unsigned char * pChannel）	
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	需要控制的目标基站的网络地址
参数三	unsigned char * pBand	指向单字节的内存指针，用于保存当前的工作频带
参数四	unsigned short * pChannel	指向双字节的内存指针，用于保存当前的工作信道
返回值	0X01=操作成功，0X00=操作失败	

7.4 自动搜索信道

函数名	char ScanChannelState （char iShell， short iObject， char iApply）	
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	需要控制的目标基站的网络地址
参数三	char iApply	搜索基站所有信道的工作状态 0X00：在搜索结束之后，基站设备不应用最优信道 0X01：在搜索结束之后，基站设备需应用最优信道
返回值	0X01=操作成功，0X00=操作失败	

7.5 读取最大信道的数量

函数名	**char GetChannelAmount** （char iShell， short iObject， unsigned char * pBand， unsigned char * pChannel， unsigned 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	short iObject	需要控制的目标基站的网络地址
参数三	unsigned char * pBand	指向单字节的内存指针，用于保存最大频带数量
参数四	unsigned char * pChannel	指向双字节的内存指针，用于保存最大信道数量
参数五	unsigned char * pStatus	预留参数扩展
返回值	0X01=操作成功，0X00=操作失败	

7.6 读取基站的信道状态

函数名	**char GetActiveChannel** （ char iShell， short iObject， short index， unsigned char * pCounter， char * pBuffer， 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	short iObject	需要控制的目标基站的网络地址
参数三	short index	起始的读取信道编号
参数四	unsigned char * pCounter	输入：需要读取的最大信道个数 输出：本次实际读取的信道个数
参数五	char * pBuffer	指向单字节的内存指针，用于保存信道的状态
参数六	unsigned char iSize	内存数组的最大长度，参考结构体 WiMinet_NoiseTable 定义： m_iUsed：该信道是否被其他基站占用 m_iBand：工作频带 m_iGood：是否是优选的符合国家无委会规定的工作信道 m_iZERO：预留的扩展字段 m_iChannel：工作信道 m_iMaxVal：最大信道噪声，单位 dBm m_iAvgVal：平均信道噪声，单位 dBm
返回值	0X01=操作成功，0X00=操作失败	

7.7 测试例程：信道管理

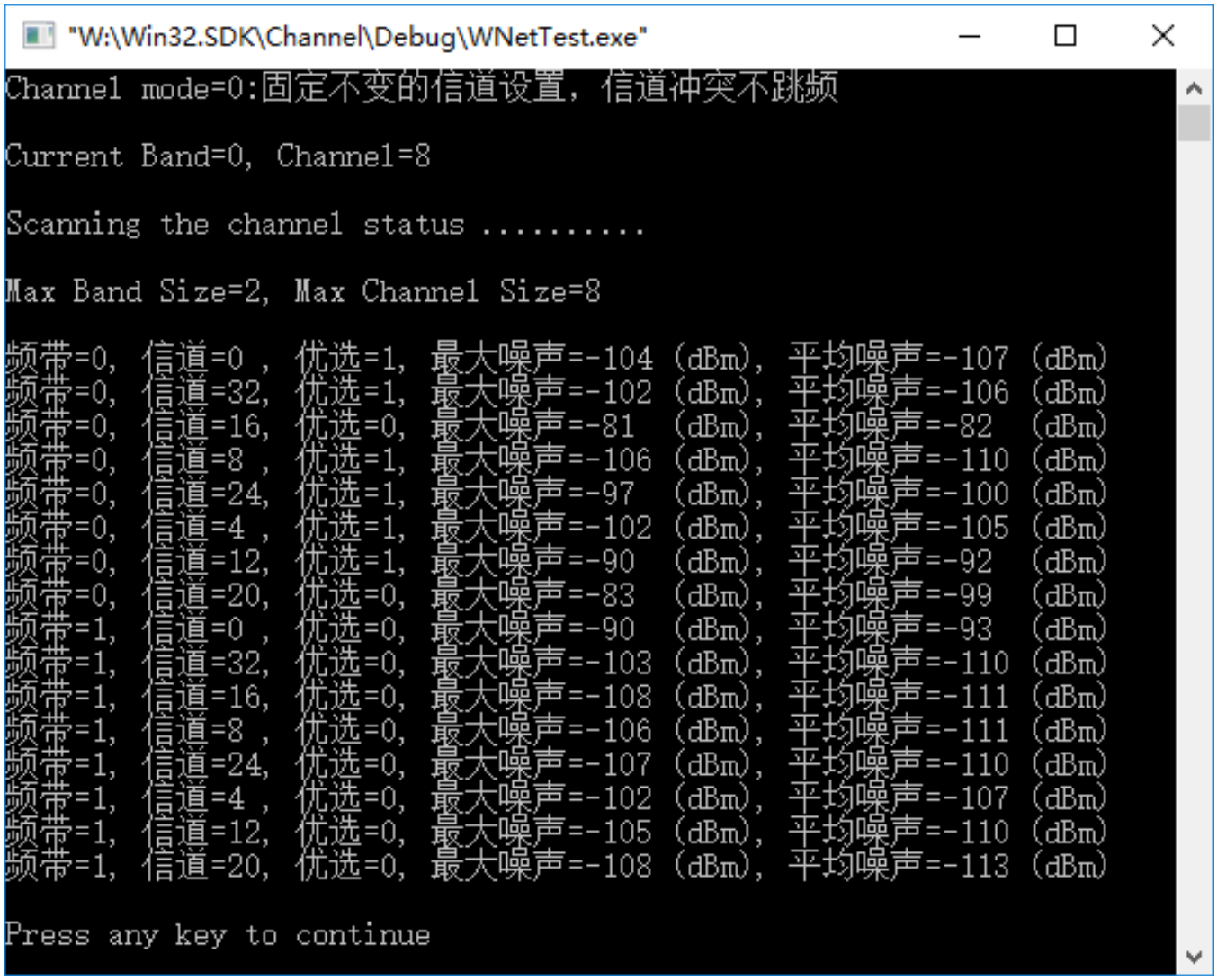
```
#include <stdio.h>
#include <Winsock2.h>
#include "API-WiMinet.h"

int main( int argc, char* argv[] )
{
    char iRetVal;
    unsigned char iMode;
    unsigned char iBand;
    unsigned char iChannel;
    unsigned char iBandSize;
    unsigned char iChannelSize;
    unsigned char iCounter;
    unsigned char iUnit;
    unsigned short iQMax;
    unsigned short index;
    char buffer[0XFF];
    WiMinet_NoiseTable * pTable;

    // 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 channel select mode
    iRetVal = GetChannelSelect( 0X00, 0X00, &iMode );

    // ++++++ IMPORTANT NOTES ++++++
    //
    // Set: SetChannelSelect( 0X00, 0X00,    iMode )
    // Get: GetChannelSelect( 0X00, 0X00, &iMode )
    //

    // Validate the operation status
    if ( !iRetVal )
    {
        printf( "GetChannelSelect failed!\r\n" );
        return 0X00;
    }

    // The channel mode
    switch( iMode )
    {
    case 0X00:
        {
            printf( "Channel mode=%d:固定不变的信道设置，信道冲突不跳频\r\n\r\n", iMode );
        }
      break;
```

```
case 0X01:
    {
        printf( "Channel mode=%d:恢复出厂配置时搜索，信道冲突会跳频\r\n\r\n", iMode );
    }
    break;

default:
    {
        printf( "Channel mode=%d:每次开机都自动搜索，信道冲突会跳频\r\n\r\n", iMode );
    }
    break;
}

// Set the chennel selection mode
iRetVal = SetChannelSelect( 0X00, 0X00,    iMode );

// Validate the operation status
if ( !iRetVal )
{
    printf( "SetChannelSelect failed!\r\n" );
    return 0X00;
}


// Get the module band and channel
GetModuleChannel( 0X00, 0X00, &iBand, &iChannel );

// ++++++ IMPORTANT NOTES ++++++
//
// Set: SetServerChannel( 0X00, 0X00, &iBand, &iChannel )
// Get: GetModuleChannel( 0X00, 0X00, &iBand, &iChannel )
//
// Note-1: ONLY the server device only needs to set the channel
// Note-2: The SET function will return current values by iBand/Channel parameter

// The module band and channel
printf( "Current Band=%d, Channel=%d\r\n\r\n", iBand, iChannel );

// Set the chennel selection mode
iRetVal = SetServerChannel( 0X00, 0X00, &iBand, &iChannel );

// Validate the operation status
if ( !iRetVal )
{
    printf( "SetServerChannel failed!\r\n" );
    return 0X00;
}


// ++++++ IMPORTANT NOTES ++++++
//
// (1) Notify the channel to scan all the channels for one time
// (2) If already scan, then read the channel status
// (3) Once triggered the scan procedure, then should wait some time
// (4) The noise table will keep until reboot or next scan
// (5) If the noise table is not required, this step CAN be skipped
//
ScanChannelState( 0X00, 0X00, 0X00 );
```

```
// Wait the device to complete the channel scan
printf( "Scanning the channel status " );

// The animation for channel scan
for ( index = 0X00; index < 0X0A; index++ )
{
    Sleep( 500 );
    printf( "." );
}
printf( "\r\n\r\n" );



// Get the channel count
GetChannelAmount( 0X00, 0X00, &iBandSize, &iChannelSize, NULL );

// The max band and channel size
printf( "Max Band Size=%d, Max Channel Size=%d\r\n\r\n", iBandSize, iChannelSize );

// The max band and channel size
iQMax = iBandSize * iChannelSize;

// Get all the band and channel table
index = 0X00;

// Get all the band and channel
while ( index < iQMax )
{
    // Max channel table to read
    iCounter = 0XFF;

    // Get the channel size
    GetActiveChannel( 0X00, 0X00, index, &iCounter, buffer, sizeof( buffer ) );

    // Update the index number
    index += iCounter;

    // The channel noise table
    pTable = ( WiMinet_NoiseTable * )buffer;

    // The channel table contents
    for ( iUnit = 0X00; iUnit < iCounter; iUnit++ )
    {
        // The channel information
        printf(
            "频带=%d, 信道=%-2d, 优选=%d, 最大噪声=%-4d (dBm), 平均噪声=%-4d (dBm)\r\n",
            pTable->m_iBand,
            pTable->m_iChannel,
            pTable->m_iGood,
            pTable->m_iMaxVal,
            pTable->m_iAvgVal );

        // Switch to the next item
        pTable++;
    }
}

// The end of the table
printf( "\r\n" );
```

```
    // Stop the shell
    StopWiMinetShell( 0X00 );

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

7.8 测试说明：信道管理



程序第 1 步：通过函数 GetChannelSelect 读取基站的信道配置模式（0X00）

程序第 2 步：通过函数 SetChannelSelect 设置基站的信道配置模式

程序第 3 步：通过函数 GetModuleChannel 读取基站的当前工作频带（0X00）和信道（0X08）

程序第 4 步：通过函数 SetServerChannel 设置基站的工作频带和信道

程序第 5 步：通过函数 ScanChannelState 设置自动搜索信道，然后延时一段时间等待完成信道搜索

程序第 6 步：通过函数 GetChannelAmount 读取最大的频带和信道数量

程序第 7 步：通过函数 GetActiveChannel 读取每一个信道的详细状态

8. 休眠节点管理

8.1 读取休眠节点的容量

函数名	**char GetTotalWakeSize** （char iShell， short iObject， unsigned char * pUnit， unsigned short * pTotal）	
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	需要控制的目标基站的网络地址
参数三	unsigned char * pUnit	指向单字节的内存指针，用于保存休眠信息的长度
参数四	unsigned short * pTotal	指向双字节的内存指针，用于保存最大的休眠节点的个数
返回值	0X01=操作成功，0X00=操作失败	

8.2 读取休眠节点信息

函数名	**char GetOneWakeMember** （char iShell， short iObject， short index，unsigned char * pCounter， char * pBuffer， 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	short iObject	需要控制的目标基站的网络地址
参数三	short index	起始记录的序列号码
参数三	unsigned char * pCounter	指向单字节的内存指针，用于保存需要读取的记录个数，该变量需要提前初始化
参数四	char * pBuffer	指向多字节的内存指针，用于保存读回来的休眠变量信息
参数五	unsigned char iSize	内存指针所指向的内存长度
返回值	0X01=操作成功，0X00=操作失败	

8.3 删除休眠节点信息

函数名	**char DeleteWakeMember** (char iShell, short iObject, char * pX16NET, unsigned short 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	需要控制的目标基站的网络地址
参数三	char * pX16NET	指向多字节的内存指针，用于保存休眠节点的网络地址，高字节在前，低字节在后
参数四	unsigned short iSize	内存地址的长度
返回值	0X01=操作成功，0X00=操作失败	

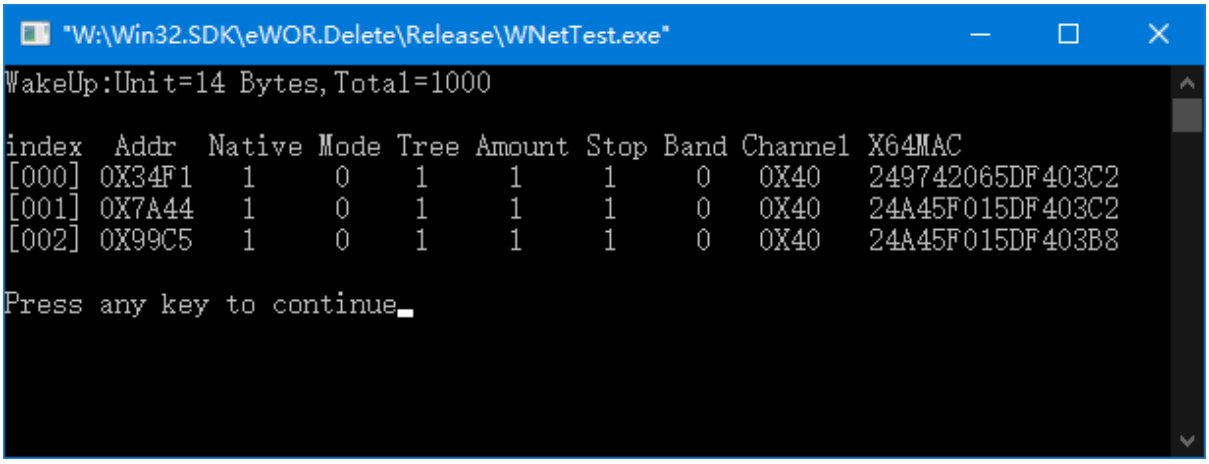
8.4 测试例程：休眠节点的删除

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

int main( int argc, char* argv[] )
{
    char iRetVal;
    FILE * pFile;
    unsigned char   iCounter;
    unsigned char   iUnit;
    unsigned short index;
    unsigned short iSize;
    unsigned short iObject;
    WiMinet_WakeUp nWakeUp;

    // COM port interface
    iRetVal = OpenWiMinetShell( "COM6",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;
    }

    // Open one file to save the wakeup items
    pFile = fopen( "D:\\Wakeup.Bin", "w+b" );
```

```
    // Validate the file pointer
    if ( !pFile )
    {
        printf( "Failed to open file D:\\Wakeup.Bin" );
        return 0X00;
    }

    // Get the total wakeup size
    iRetVal = GetTotalWakeSize( 0X00, 0X00, &iUnit, &iSize );

    // Validate the operation status
    if ( !iRetVal )
    {
        printf( "Error:GetTotalWakeSize\r\n" );
        return 0X00;
    }

    // The wakeup item status
    printf( "WakeUp:Unit=%d Bytes,Total=%d\r\n\r\n", iUnit, iSize );

    // The header table
    printf( "index    Addr    Native Mode Tree Amount Stop Band Channel X64MAC\r\n" );

    // Read out all the wakeup items
    for ( index = 0X00; index < iSize; index++ )
    {
        // The item to read from the device
        iCounter = 0X01;

        // Read out the item
        iRetVal = GetOneWakeMember( 0X00, 0X00, index, &iCounter, ( char * )&nWakeUp,
sizeof( nWakeUp ) );

        // Validate the operation results
        if ( !iRetVal )
        {
            printf( "Error:GetOneWakeMember\r\n" );
            break;
        }

        // Validate the item address
        if ( !nWakeUp.m_iTxAddr )
        {
            continue;
        }

        // Save the item in the disk file
        iCounter = fwrite( &nWakeUp, 0X01, sizeof( nWakeUp ), pFile );

        // The wakeup item status
        printf( "[%03d]",       index );
        printf( " 0X%04X",      nWakeUp.m_iTxAddr );
        printf( "      %d ",    nWakeUp.m_iNative );
        printf( "       %d",    nWakeUp.m_iTxMode );
        printf( "       %d",    nWakeUp.m_iTxTree );
        printf( "         %d",  nWakeUp.m_iAmount );
        printf( "         %d",  nWakeUp.m_iTxStop );
        printf( "         %d",  nWakeUp.m_Profile.m_WORadio.m_iBand );
        printf( "      0X%02X", nWakeUp.m_Profile.m_WORadio.m_iChannel );
```

```
        // The X64MAC address
        printf( "      " );
        for ( iCounter = 0X00; iCounter < 0X08; iCounter++ )
        {
            printf( "%02X", ( unsigned char )nWakeUp.m_pX64MAC[iCounter] );
        }
        printf( "\r\n" );
    }

    // The end of the limiter
    printf( "\r\n" );

    // Close this file
    fclose( pFile );


    // The wakeup item to be deleted
    iObject = 0X8341;

    // Delete one item
    iRetVal = DeleteWakeMember( 0X00, 0X00, ( char * )&iObject, sizeof( iObject ) );

    // Validate the operation status
    if ( !iRetVal )
    {
        printf( "Error:DeleteWakeMember\r\n" );
        return 0X00;
    }

    // Stop the shell
    StopWiMinetShell( 0X00 );

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

8.5 测试说明：休眠节点的删除



该程序首先以二进制写入方式打开一个磁盘文件，用于保存读取的休眠节点信息，然后将读取到的休眠节点信息写入到磁盘文件中。

8.6 插入休眠节点信息

函数名	**char InsertWakeMember** (char iShell , short iObject , WiMinet_WakeUp * pWakeUp)	
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	需要控制的目标基站的网络地址
参数三	WiMinet_WakeUp * pWakeUp	指向多字节的内存指针，用于存放需要写入休眠节点的信息
返回值	0X01=操作成功，0X00=操作失败	

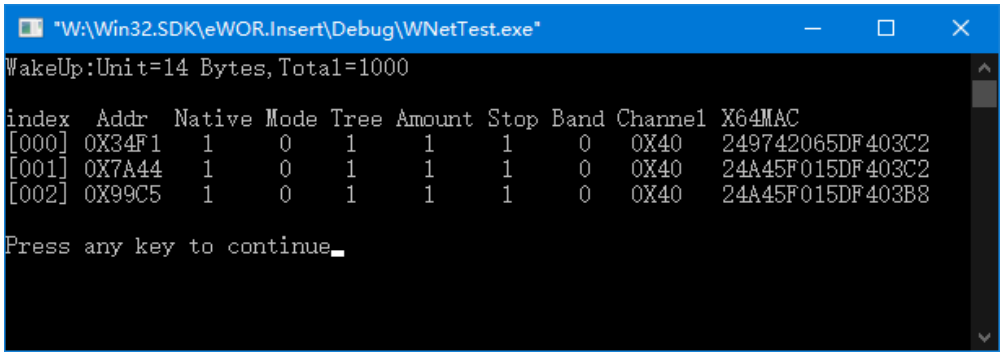
8.7 测试例程：休眠节点的插入

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

int main( int argc, char* argv[] )
{
    char iRetVal;
    FILE * pFile;
    unsigned char   iCounter;
    unsigned char   iUnit;
    unsigned long   dwCount;
    unsigned short index;
    unsigned short iSize;
    WiMinet_WakeUp nWakeUp;

    // COM port interface
    //iRetVal = OpenWiMinetShell( "COM6",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;
    }

    // Open one file to save the wakeup items
    pFile = fopen( "D:\\Wakeup.Bin", "rb" );

    // Validate the file pointer
    if ( !pFile )
    {
        printf( "Failed to open file D:\\Wakeup.Bin" );
        return 0X00;
    }
```

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

    // Get the file size
    dwCount = ftell( pFile );

    // Seek to the end of this file
    fseek( pFile, 0X00, SEEK_SET );

    // The counter of the item in this file
    iSize = ( unsigned short )( dwCount / sizeof( WiMinet_WakeUp ) );

    // Insert all the items in to this device
    for ( index = 0X00; index < iSize; index++ )
    {
        // Read out the data from the file
        fread( &nWakeUp, 0X01, sizeof( WiMinet_WakeUp ), pFile );

        // Insert to the device
        InsertWakeMember( 0X00, 0X00, &nWakeUp );
    }

    // Close this file
    fclose( pFile );

    // Reset the device for reload for working(0X00=hot reset, 0XFF=cool reset)
    ResetWiMinetHost( 0X00, 0X00, 0X00 );

    // Get the total wakeup size
    iRetVal = GetTotalWakeSize( 0X00, 0X00, &iUnit, &iSize );

    // Validate the operation status
    if ( !iRetVal )
    {
        printf( "Error:GetTotalWakeSize\r\n" );
        return 0X00;
    }

    // The wakeup item status
    printf( "WakeUp:Unit=%d Bytes,Total=%d\r\n\r\n", iUnit, iSize );

    // The header table
    printf( "index    Addr    Native Mode Tree Amount Stop Band Channel X64MAC\r\n" );

    // Read out all the wakeup items
    for ( index = 0X00; index < iSize; index++ )
    {
        // The item to read from the device
        iCounter = 0X01;

        // Read out the item
        iRetVal = GetOneWakeMember( 0X00, 0X00, index, &iCounter, ( char * )&nWakeUp,
sizeof( nWakeUp ) );

        // Validate the operation results
        if ( !iRetVal )
        {
            printf( "Error:GetOneWakeMember\r\n" );
            break;
```

```
        }

        // Validate the item address
        if ( !nWakeUp.m_iTxAddr )
        {
            continue;
        }

        // The wakeup item status
        printf( "[%03d]",       index );
        printf( " 0X%04X",      nWakeUp.m_iTxAddr );
        printf( "     %d ",     nWakeUp.m_iNative );
        printf( "      %d",     nWakeUp.m_iTxMode );
        printf( "      %d",     nWakeUp.m_iTxTree );
        printf( "       %d",    nWakeUp.m_iAmount );
        printf( "       %d",    nWakeUp.m_iTxStop );
        printf( "       %d",    nWakeUp.m_Profile.m_WORadio.m_iBand );
        printf( "     0X%02X", nWakeUp.m_Profile.m_WORadio.m_iChannel );

        // The X64MAC address
        printf( "     " );
        for ( iCounter = 0X00; iCounter < 0X08; iCounter++ )
        {
            printf( "%02X", ( unsigned char )nWakeUp.m_pX64MAC[iCounter] );
        }
        printf( "\r\n" );
    }

    // The end of the limiter
    printf( "\r\n" );

    // Stop the shell
    StopWiMinetShell( 0X00 );

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

8.8 测试说明：休眠节点的插入



该程序首先以二进制只读方式打开一个磁盘文件，用于读取休眠节点的信息，然后将读取到的休眠节点信息写入到设备中。写入完成之后，为了将设备热复位以将配置参数起作用。

特别说明：休眠节点参数信息写入之后，有下述注意事项

（1） **不要立刻关闭设备的电源**，因为写入的休眠节点信息需可靠地同步写入到 Flash 中去，需要几秒钟的的时间

（2） **不要对系统进行冷复位**，因为冷复位会丢失刚才写入的所有的信息，热复位不会丢失

（3） 写入了休眠节点的信息之后，这些数据需要在重启之后才会加到到运行内存中去，因此需要一个复位操作，冷复位需要的时间较长，热复位很快，可以立刻执行

9. 基站接口管理

9.1 复位系统

函数名	char ResetWiMinetHost （char iShell， short iObject， char iReboot）	
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	需要控制的目标基站的网络地址
参数三	char iReboot	复位的模式，0XFF 是冷复位，其余数值=热复位
返回值	0X01=操作成功，0X00=操作失败	

9.2 获取基站的 IP 地址：64 位整数标识

函数名	long GetHostViaX64INT （unsigned __int64 qwINT64， char * pBuffer， 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	unsigned __int64 qwINT64	基站的全球唯一标识号码，64 位的整数值
参数二	char * pBuffer	对应的基站的 IP 地址，至少为 16 字节的长度
参数三	unsigned char iSize	参数二的 pBuffer 的内存长度
返回值	32 位的网络地址，以网络字节顺序存储	

9.3 获取基站的 IP 地址：字符串指针标识

函数名	long GetHostViaX64STR （char * pX64STR , char * pBuffer， 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 * pX64STR	基站的全球唯一标识符号的字符串
参数二	char * pBuffer	对应的基站的 IP 地址，至少为 16 字节的长度
参数三	unsigned char iSize	参数二的 pBuffer 的内存长度
返回值	32 位的网络地址，以网络字节顺序存储	

9.4 获取基站的 IP 地址：8 字节数组标识

函数名	long GetHostViaX64MAC （char * pX64MAC , char * pBuffer , 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 * pX64STR	基站的全球唯一标识符号的 8 字节数组标识
参数二	char * pBuffer	对应的基站的 IP 地址，至少为 16 字节的长度
参数三	unsigned char iSize	参数二的 pBuffer 的内存长度
返回值	32 位的网络地址，以网络字节顺序存储	

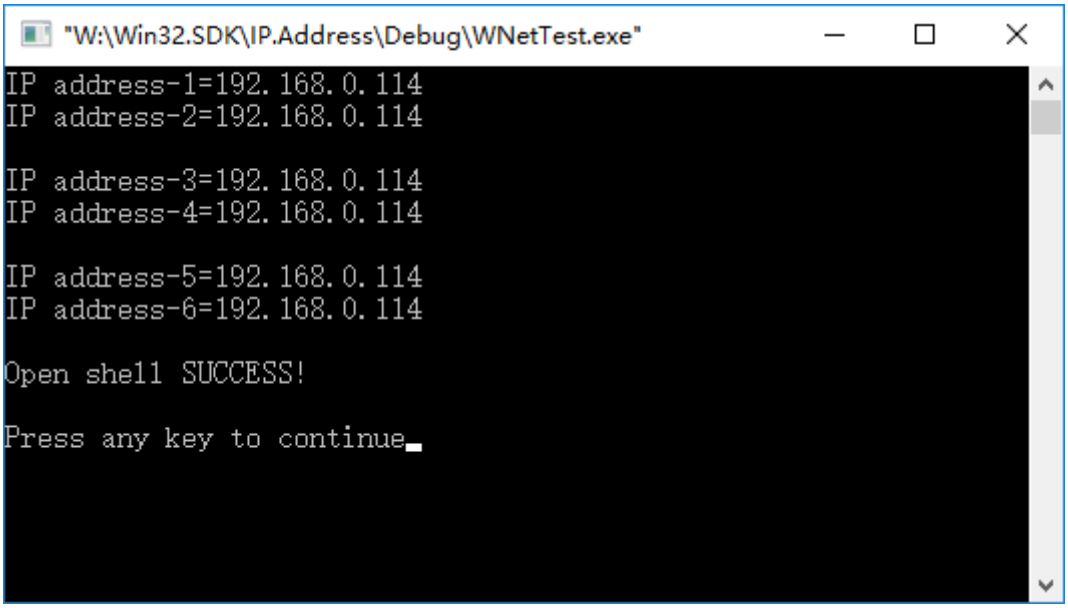
9.5 测试例程：基站接口管理

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

int main( int argc, char* argv[] )
{
    char iRetVal;
    char pX64MAC[0X08];
    char buffer[0X10];
    char * pAddr;
    struct in_addr addr;
    unsigned __int64 qwX64INT;

    // ==================== GetHostViaX64INT ====================
    //
    // The int64 value
    //
    // (1) X64INT = 0X24E4C303556B5917
    // (2) X64INT = 0X24E4C303556B5917UL
    // (3) X64INT = 0X24E4C303556B591ul
    // (4) X64INT = 0X24E4C303556B5917i64
    //
    qwX64INT = 0X24E4C303556B5917;

    // Get the address value
    addr.S_un.S_addr = GetHostViaX64INT( qwX64INT, buffer, sizeof( buffer ) );

    // Translate to a.b.c.d format IP address
    pAddr = inet_ntoa( addr );

    // The IP address-1 information
    printf( "IP address-1=%s\r\n", pAddr );

    // The IP address information
    printf( "IP address-2=%s\r\n\r\n", buffer );
```

```
// ==================== GetHostViaX64STR ====================
//
// The X64String value
//
// (1) X64STR="24E4C303556B5917"
// (2) X64STR="0X24E4C303556B5917"
// (3) X64XTR="0x24E4C303556B5917"
//
// Get the address value
addr.S_un.S_addr = GetHostViaX64STR( "24E4C303556B5917", buffer, sizeof( buffer ) );

// Translate to a.b.c.d format IP address
pAddr = inet_ntoa( addr );

// The IP address-1 information
printf( "IP address-3=%s\r\n", pAddr );

// The IP address information
printf( "IP address-4=%s\r\n\r\n", buffer );


// ==================== GetHostViaX64MAC ====================
//
// X64MAC:HEX for decimal value
//
// The X64MAC address
pX64MAC[0X00] = ( char )0X24;
pX64MAC[0X01] = ( char )0XE4;
pX64MAC[0X02] = ( char )0XC3;
pX64MAC[0X03] = ( char )0X03;
pX64MAC[0X04] = ( char )0X55;
pX64MAC[0X05] = ( char )0X6B;
pX64MAC[0X06] = ( char )0X59;
pX64MAC[0X07] = ( char )0X17;

// Get the address value
addr.S_un.S_addr = GetHostViaX64MAC( pX64MAC, buffer, sizeof( buffer ) );

// Translate to a.b.c.d format IP address
pAddr = inet_ntoa( addr );

// The IP address-1 information
printf( "IP address-5=%s\r\n", pAddr );

// The IP address information
printf( "IP address-6=%s\r\n\r\n", buffer );


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

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

// The open shell status
printf( "Open shell SUCCESS!\r\n\r\n" );
```

```
    // Stop the shell
    StopWiMinetShell( 0X00 );

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

9.6 测试说明：基站接口管理



该程序用三种形式，分别是 GetHostViaX64INT，GetHostViaX64STR，GetHostViaX64MAC 分别获取基站的 IP 地址，然后用该 IP 地址打开目标基站的连接。

注意三个函数的第一个参数的类型和输入方式是不一样，其参数输出和函数返回数值是一样的。

10. 休眠节点的 LED 控制

10.1 控制休眠设备的 LED

函数名	char Set_eWakeUp_Ctrl (char iShell , short iObject , char iMask , char iStatus , short iTimer)	
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	需要控制的目标基站的网络地址
参数三	char iMask	目标设备的通道掩码，每一个比特代表一个通道,数值=1 代表需要控制该通道，数值=0 代表不控制该通道 Bit0=Channel0 Bit1=Channel1 Bit2=Channel2 Bit3=Channel3 Bit4=Channel4 Bit5=Channel5 Bit6=Channel6 Bit7=Channel7
参数四	char iStatus	目标设备的状态掩码，每一个比特代表对应一个通道的输出状态,数值=1 代表需要开启该通道，数值=0 代表关闭该通道 Bit0：通道 0 开关状态 Bit1=通道 1 开关状态 Bit2=通道 2 开关状态 Bit3=通道 3 开关状态 Bit4=通道 4 开关状态 Bit5=通道 5 开关状态 Bit6=通道 6 开关状态 Bit7=通道 7 开关状态
参数五	short iTimer	通道开关延时，如果是开启通道，该通道在延时该时间之后自动关闭，如果是关闭通道，该通道会立刻关闭。
返回值	0X01=操作成功，0X00=操作失败	

10.2 开启休眠设备的 LED 通道

函数名	char Set_eWakeUp_Open (char iShell , short iObject , char iChannel , short iTimer)	
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	需要控制的目标基站的网络地址
参数三	char iChannel	目标设备的通道号码，取值范围 0-7
参数四	short iTimer	通道开启延时，该通道在延时该时间之后自动关闭
返回值	0X01=操作成功，0X00=操作失败	

10.3 关闭休眠设备的 LED 通道

函数名	char Set_eWakeUp_Stop （char iShell， short iObject， char iChannel）	
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	需要控制的目标基站的网络地址
参数三	char iChannel	目标设备的通道号码，取值范围 0-7
返回值	0X01=操作成功，0X00=操作失败	

10.4 测试例程：休眠 LED 控制

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

#define WIMINET_EWOR_CMD0()
#define WIMINET_EWOR_CMD1()
#define WIMINET_EWOR_CMD2()
#define WIMINET_EWOR_CMD3()
#define WIMINET_EWOR_CMD4()
#define WIMINET_EWOR_CMD5()
#define WIMINET_EWOR_CMD6()
#define WIMINET_EWOR_CMD7()

int main( int argc, char* argv[] )
{
    char iRetVal;
    unsigned char iChar;
    unsigned short iTimer;
    unsigned short iObject;

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

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

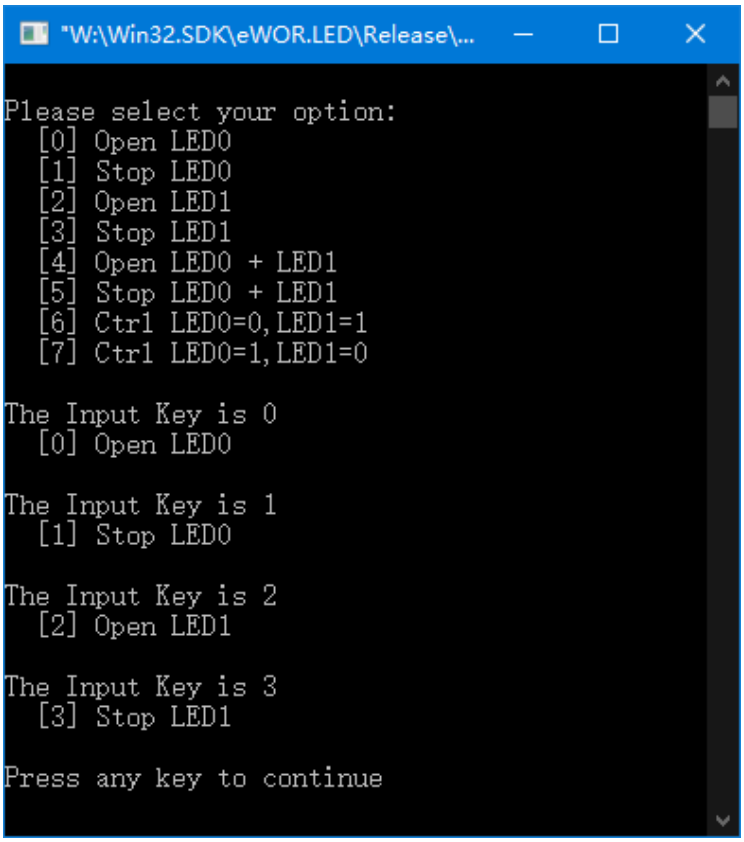
```
printf( "   [0] Open LED0\r\n" )
printf( "   [1] Stop LED0\r\n" )
printf( "   [2] Open LED1\r\n" )
printf( "   [3] Stop LED1\r\n" )
printf( "   [4] Open LED0 + LED1\r\n" )
printf( "   [5] Stop LED0 + LED1\r\n" )
printf( "   [6] Ctrl LED0=0,LED1=1\r\n" )
printf( "   [7] Ctrl LED0=1,LED1=0\r\n" )
```

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

// The notice header
printf( "\r\nPlease select your option:\r\n" );

// The command menu
WIMINET_EWOR_CMD0();
WIMINET_EWOR_CMD1();
WIMINET_EWOR_CMD2();
WIMINET_EWOR_CMD3();
WIMINET_EWOR_CMD4();
WIMINET_EWOR_CMD5();
WIMINET_EWOR_CMD6();
WIMINET_EWOR_CMD7();

// The unit is mini-second(ms)
iTimer = 5000;

// The object address
iObject = 0X805D;

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

    // Get the input character
    iChar = _getch();

    // The exit input
    if ( ( iChar == 'q' ) || ( iChar == 'Q' ) )
    {
        printf( "\r\n" );
        break;
    }

    // The input key value
    printf( "\r\nThe Input Key is %c\r\n", iChar );

    // The command processor
    switch ( iChar )
    {
    case '0':
        {
            // The command notice message
            WIMINET_EWOR_CMD0();

            // Channel-0 = Open
            Set_eWakeUp_Open( 0X00, iObject, 0X00, iTimer );
        }
        break;

    case '1':
        {
```

```
            // The command notice message
            WIMINET_EWOR_CMD1();

            // Channel-0 = Stop
            Set_eWakeUp_Stop( 0X00, iObject, 0X00 );
        }
        break;

    case '2':
        {
            // The command notice message
            WIMINET_EWOR_CMD2();

            // Channel-1 = Open
            Set_eWakeUp_Open( 0X00, iObject, 0X01, iTimer );
        }
        break;

    case '3':
        {
            // The command notice message
            WIMINET_EWOR_CMD3();

            // Channel-1 = Stop
            Set_eWakeUp_Stop( 0X00, iObject, 0X01 );
        }
        break;

    case '4':
        {
            // The command notice message
            WIMINET_EWOR_CMD4();

            // Channel-0/1 = Open
            Set_eWakeUp_Ctrl( 0X00, iObject, 0X03, 0X03, iTimer );
        }
        break;

    case '5':
        {
            // The command notice message
            WIMINET_EWOR_CMD5();

            // Channel-0/1 = Stop
            Set_eWakeUp_Ctrl( 0X00, iObject, 0X03, 0X00, iTimer );
        }
        break;

    case '6':
        {
            // The command notice message
            WIMINET_EWOR_CMD6();

            // Channel-0 = Stop
            // Channel-1 = Open
            Set_eWakeUp_Ctrl( 0X00, iObject, 0X03, 0X02, iTimer );
        }
        break;

    case '7':
```

```
            {
                // The command notice message
                WIMINET_EWOR_CMD7();

                // Channel-0 = Open
                // Channel-1 = Stop
                Set_eWakeUp_Ctrl( 0X00, iObject, 0X03, 0X01, iTimer );
            }
            break;

        default:
            {
                printf( "Invalid command!\r\n" );
            }
            break;
        }
    }

    // Stop the shell
    StopWiMinetShell( 0X00 );

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

10.5 测试说明：休眠 LED 控制



该程序首先打印提示信息，演示了控制单个通道的开关，以及多个通道的开关控制策略。

11. 全局消息钩子

11.1 使能全局消息钩子

函数名	**void EnableGlobalHook** （unsigned long dwEnable）	
头文件	**API-WiMinet.h**	
静态库	**WiMinet.lib**	
动态库	**WiMinet.dll**	
	形式	说明
参数一	unsigned long dwEnable	同一条消息的最大输出次数。 (1) 0X00 代表关闭钩子的功能 (2) 关闭全局消息钩子之后，消息将会以普通数据输出。
返回值	无	

11.2 读取全局消息钩子数据

函数名	**char GetHookRxMessage**（unsigned short * pSource， 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	unsigned short * pSource	指向源节点地址的 16 位数据指针
参数二	char * pBuffer	指向源节点上报的全局消息的数据块指针
参数三	long dwSize	参数二的数据块的长度
返回值	全局消息的数据块的大小	
备注一	全局消息通常是开关量上报的信号，默认以 WiMinet_DIReport_SDK 结构体的形式封装	
备注二	WiMinet_DIReport_SDK 结构体成员变量的定义： typedef struct _WiMinet_DIReport_SDK_ { WiMinet_DIReport_EXT m_nPacket; unsigned long m_dwShell; char m_pDevice[0X20]; } WiMinet_DIReport_SDK; 其中各成员变量定义如下： (1) m_nPacket 是基站上报的原始数据结构，具体见 WiMinet_DIReport_EXT 定义 (2) m_dwShell 是当前连接的 Shell 的编号 (3) m_pDevice 是当前连接的设备名称	

备注三	WiMinet_DIReport_EXT 结构体成员变量的定义： typedef struct _WiMinet_DIReport_EXT_ { WiMinet_DIReport m_nReport; unsigned char m_pX64MAC[0X08]; } WiMinet_DIReport_EXT; 其中各成员变量定义如下： (1) m_nReport 是从站上报的原始数据结构，具体见 WiMinet_DIReport 定义 (2) m_pX64MAC 是基站的全球唯一的 64 位 UUID 硬件编码
备注四	typedef struct _WiMinet_DIReport_ { unsigned short m_iBATVol; unsigned char m_iChannel:0X04; unsigned char m_iUserDI:0X01; unsigned char m_iQEvent:0X03; unsigned char m_iStatus; unsigned long m_dwSerial; unsigned short m_iX16NET; unsigned char m_iRFChip; signed char m_iRxRSSI; signed char m_iTxRSSI; } WiMinet_DIReport; WiMinet_DIReport 结构体成员变量的定义： (1) m_iBATVol：16bit，电池的电压 * 1000 的整数值 (2) m_iChannel：4bit，数字量的通道号码 (3) m_iUserDI：1bit，数字量上报消息，固定为数值 1 (4) m_iQEvent：3bit，事件类型代码 (5) m_iStatus：8bit，数字量通道的状态值 (6) m_dwSerial：32bit，报文的序列号码，按照递增的顺序确保唯一性 (7) m_iX16NET：16bit，基站的 16 位网络地址 (8) m_iRFChip：8bit，接收到该报文的射频模组的编号 (9) m_iRxRSSI：8bit，从节点接收到的基站的信号强度，单位是 dBm (10) m_iTxRSSI：8bit 基站的射频模组接收到从节点的信号强度，单位是 dBm

11.3 测试例程：全局消息钩子

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

int main( int argc, char* argv[] )
{
    char buffer[0X80];
    unsigned char index;
    unsigned char iSize;
    unsigned char iShell[0X10];
    unsigned long dwTotal;
    unsigned short iSource;
    WiMinet_DIReport_SDK * pReport;

    // Clear all the shell
    memset( iShell, 0X00, sizeof( iShell ) );

    // The 1st shell interface
    iShell[0X00] = OpenWiMinetShell( "COM4",115200UL, 0X01 );
```

```
// The 1st shell interface
//iShell[0X01] = OpenWiMinetShell( "COM10",115200UL, 0X01 );

// The 1st shell interface
//iShell[0X00] = OpenWiMinetShell( "192.168.0.240",12580, 0X01 );

// The 2nd shell interface
iShell[0X01] = OpenWiMinetShell( "192.168.0.241",12580, 0X01 );

// Enable the hook service
EnableGlobalHook( 0XFF );

// Validate the shell open interface
if ( !iShell[0X00] && !iShell[0X01] )
{
    printf( "Open shell failed!\r\n" );
    return 0X00;
}

// The total size
dwTotal = 0X00UL;

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

    // Get the hooked message
    iSize = GetHookRxMessage( &iSource, buffer, sizeof( buffer ) );

    // Validate the packet size
    if ( !iSize )
    {
        continue;
    }

    // Update the total size
    dwTotal += iSize;

    // The source address
    printf( "\r\nSource=0X%04X\r\n", iSource );

    // Report the message: contents
    printf( "Content[%d/%lu]=", iSize, dwTotal );

    // The byte contents of the input buffer
    for ( index = 0X00; index < iSize; index++ )
    {
        printf( "%02X ", (unsigned char)buffer[index] );
    }

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

    // The pointer to the object
    pReport = ( WiMinet_DIReport_SDK * )buffer;

    // The member contents
    printf( "Members:\r\n" );
    printf( "    [0] Battery=%.3f(V)\r\n", pReport->m_nPacket.m_nReport.m_iBATVol * 0.001f );
    printf( "    [1] Channel=%d\r\n", pReport->m_nPacket.m_nReport.m_iChannel );
    printf( "    [2] IReport=%d\r\n", pReport->m_nPacket.m_nReport.m_iUserDI );
```

```
        printf( "     [3] EventID=0X%02X\r\n", pReport->m_nPacket.m_nReport.m_iQEvent );
        printf( "     [4] DIState=0X%02X\r\n", pReport->m_nPacket.m_nReport.m_iStatus );
        printf( "     [5] Counter=0X%lX\r\n", pReport->m_nPacket.m_nReport.m_dwSerial );
        printf( "     [6] NETAddr=0X%04X\r\n", pReport->m_nPacket.m_nReport.m_iX16NET );
        printf( "     [7] Chipset=%lX\r\n", pReport->m_nPacket.m_nReport.m_iRFChip );
        printf( "     [8] Rx-RSSI=%d(dBm)\r\n", pReport->m_nPacket.m_nReport.m_iRxRSSI );
        printf( "     [9] Tx-RSSI=%d(dBm)\r\n", pReport->m_nPacket.m_nReport.m_iTxRSSI );
        printf( "     [9] Tx-RSSI=%d(dBm)\r\n", pReport->m_nPacket.m_nReport.m_iTxRSSI );

        // The X64MAC address of the server
        printf( "     [A] MACAddr=0X" );

        // The size of the X64MAC address
        iSize = sizeof( pReport->m_nPacket.m_pX64MAC );

        // Get the item counter
        iSize /= sizeof( pReport->m_nPacket.m_pX64MAC[0X00] );

        // The X64MAC address
        for ( index = 0X00; index < iSize; index++ )
        {
            printf( "%02X", pReport->m_nPacket.m_pX64MAC[index] );
        }
        printf( "\r\n" );

        // The shell of the server
        printf( "     [B] IOShell=%lu\r\n", pReport->m_dwShell );

        // The device name of the server
        printf( "     [C] DevName=%s\r\n", pReport->m_pDevice );
    }

    // Stop the shell
    StopWiMinetShell( 0X00 );

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

11.4 测试说明：全局消息钩子

该程序首先连接 2 个基站，接收全局消息，然后通过函数 **EnableGlobalHook** 设置全局消息的输出数量，如果该数量设置为 0XFF 或者 更大的数值，代表所有的消息都输出，由用户自己做唯一性处理

如果设置数量为 0X01，则 SDK 会做过滤，确保只输出一个消息

如果设置数量为 0X00，则关闭全局消息输出，所有的全局消息都会被当做普通的数据输出，可以通过函数 **ReadInputMessage** 读取数据。

关于全局消息的说明如下：

(1) 所有的双模基站都配置有 2 个射频模组，其中一个工作在专有信道，另外一个工作在公共信道

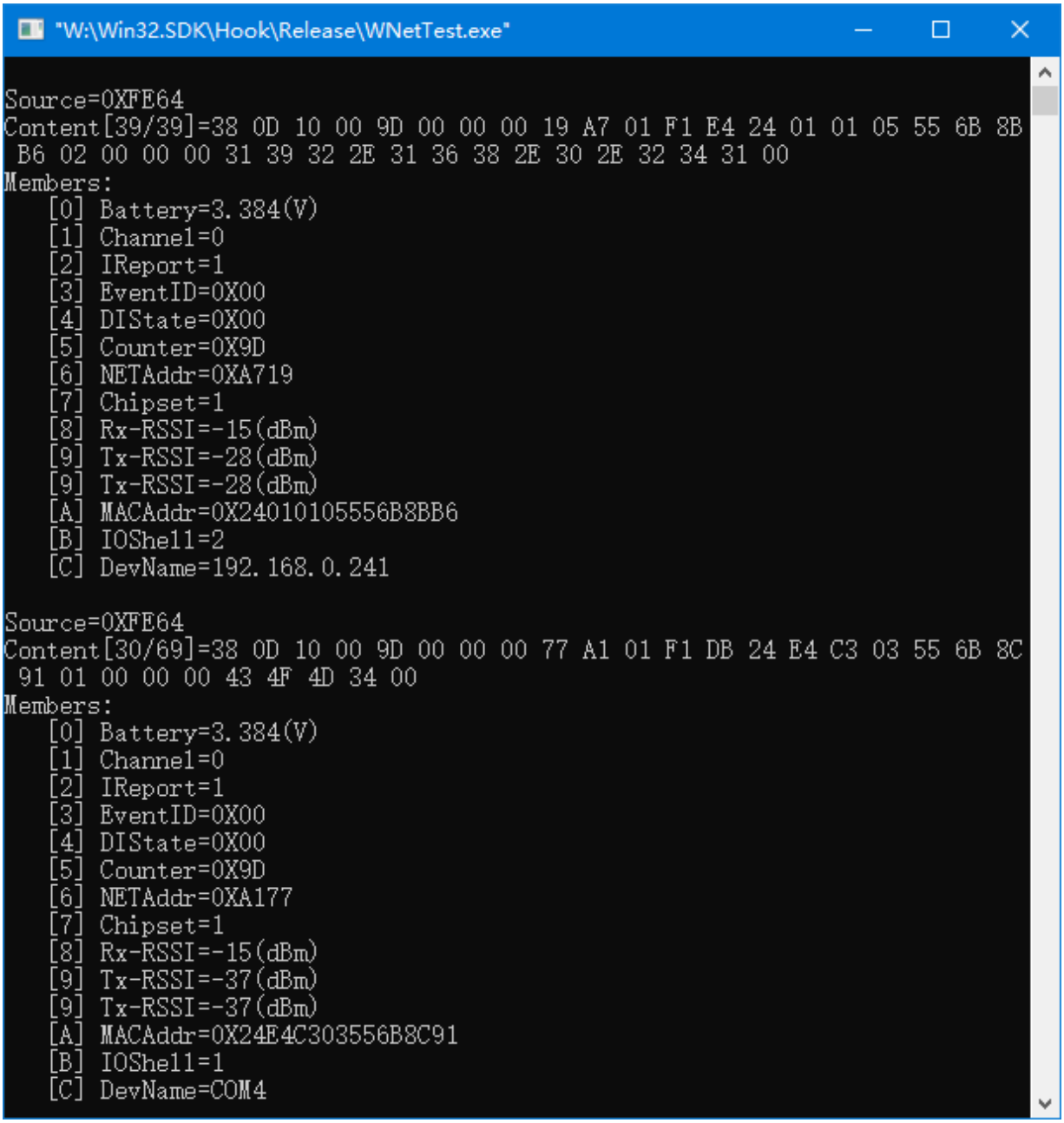
(2) 从节点发送的全局消息，所有的双模基站的公共信道都可以收到一份数据

(3) 从节点发送的局域消息，只有其中某一个基站的自组网模组可以接收到数据，公共模组不会接收到数据

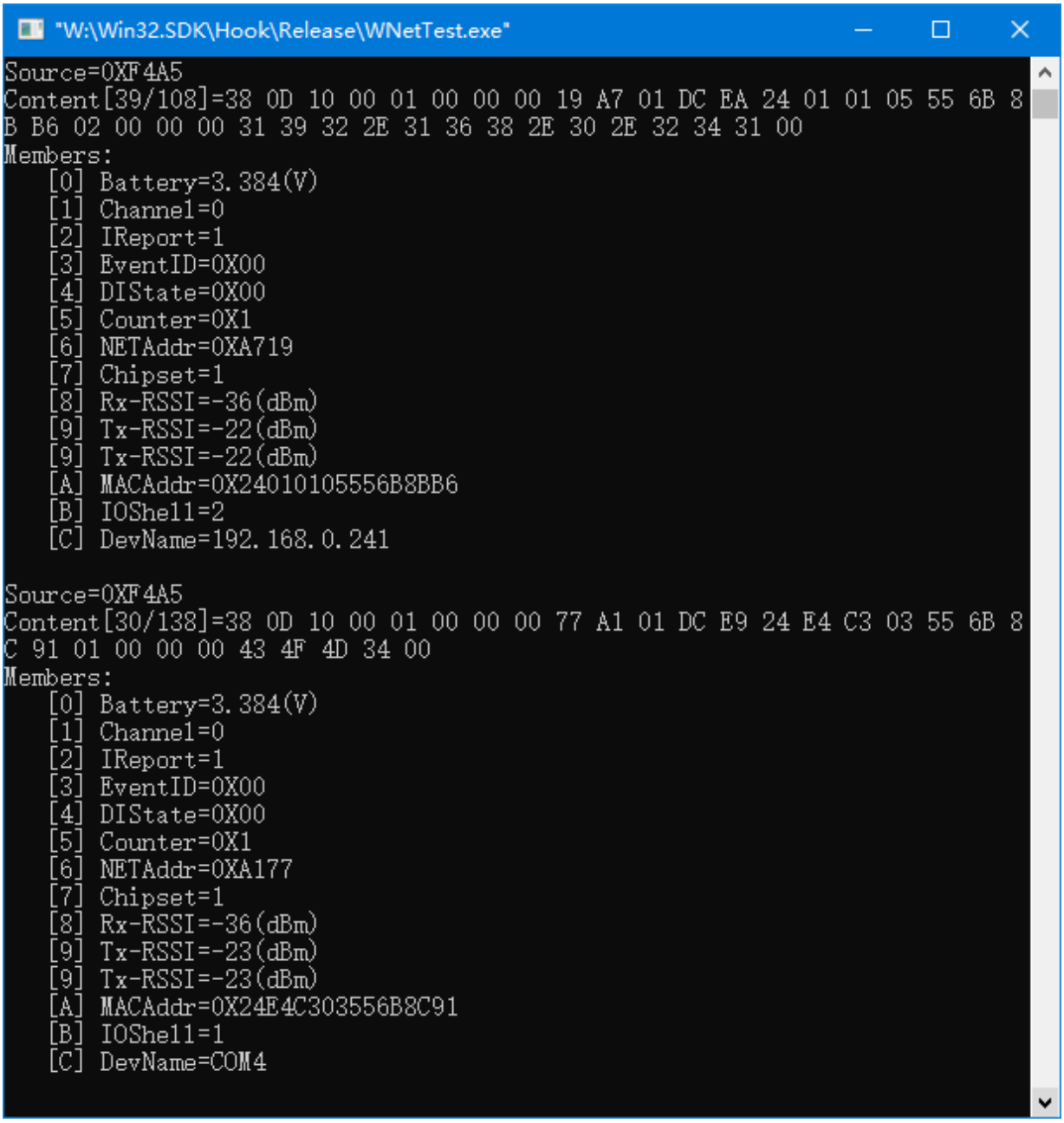
需要说明的是，全局消息是所有的双模基站都会收到并输出的，因此函数 GetHookRxMessage 可以接收到所有的双模基站输出的同一个同一份数据，如果需要 SDK 协助过滤，请将 **EnableGlobalHook** 的参数设置为 0X01。

程序先输出二进制格式的数据内容，然后按照 WiMinet_DIReport 的成员变量的形式输出每一个变量的数值。

如果用户不关心数据的内容，仅仅关心是哪一个节点触发了按键事件，可以直接将函数 **GetHookRxMessage** 的第 2,3 参数设置为 0X00。



低功耗休眠节点（0XFE64）按键事件、电池电压上报事件、开关灯确认事件、漫游节点信号等信息



低功耗休眠节点（0XF4A5）按键事件、电池电压上报事件、开关灯确认事件、漫游节点信号等信息

12. 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