

ESP-iSYS 实时数据库软件

API 接口规范手册






声 明

- 严禁转载本手册的部分或全部内容。
- 在不经预告和联系的情况下，本手册的内容有可能发生变更，请谅解。
- 本手册所记载的内容，不排除有误记或遗漏的可能性。如对本手册内容有疑问，请与我公司联系，联系邮箱：SMS@supcon.com。

商 标

中控、SUPCON、SPlant、Webfield、ESP-iSYS、MultiF、InScan、SupField 以上商标或标识均是浙江中控技术股份有限公司已经注册或已经申请注册或正在使用的商标和标识，拥有以上商标的所有权，未经浙江中控技术股份有限公司的书面授权，任何个人及企业不得擅自使用上述商标，对于非法使用我司商标的行为，我司将保留依法追究行为人及企业的法律责任的权利。

文档标志符定义

	<p>警告：标示有可能导致人身伤亡或设备损坏的信息。</p> <p>WARNING: Indicates information that a potentially hazardous situation which, if not avoided, could result in serious injury or death.</p>
	<p>电击危险：标示有可能产生电击危险的信息。</p> <p>RISK OF ELECTRICAL SHOCK: Indicates information that Potential shock hazard where HAZARDOUS LIVE voltages greater than 30V RMS, 42.4V peak, or 60V DC may be accessible.</p>
	<p>防止静电：标示防止静电损坏设备的信息。</p> <p>ESD HAZARD: Indicates information that Danger of an electro-static discharge to which equipment may be sensitive. Observe precautions for handling electrostatic sensitive devices</p>
	<p>注意：提醒需要特别注意的信息。</p> <p>ATTENTION: Identifies information that requires special consideration.</p>
	<p>提示：标记对用户的建议或提示。</p> <p>TIP: Identifies advice or hints for the user.</p>

目 录

API 接口规范手册	5
1 前言	5
1.1 手册简介	5
1.2 客户端需求	5
2 软件概述	6
2.1 API 模块概念	6
2.2 API 模块功能概述	6
3 接口定义	7
3.1 辅助函数模块	7
3.1.1 GetLastException	7
3.1.2 Connect	7
3.1.3 GetVersionNumber	8
3.1.4 GetRTDBTime	9
3.1.5 IsConnected	9
3.1.6 Disconnect	10
3.1.7 GetErrorMgs	10
3.1.8 GetWaitTime	11
3.1.9 SetWaitTime	12
3.1.10 ReqUpdateMessage	12
3.2 用户管理模块	13
3.2.1 LogIn	13
3.2.2 LogOut	14
3.2.3 SetClientName	14
3.2.4 AddUser	14
3.2.5 DelUser	15
3.2.6 GetUserInfo	16
3.2.7 SetUserDef	17
3.2.8 GetGroupsInUser	18
3.2.9 EnumUser	18
3.2.10 AddUserGroup	19
3.2.11 DelUserGroup	19
3.2.12 GetUserGroupDef	20
3.2.13 SetUserGroupDef	21
3.2.14 AddUserToUserGroup	21
3.2.15 RemoveUserFromUserGroup	22
3.2.16 EnumUserGroup	22
3.2.17 EnumUserACL	23

3.2.18 EnumUserGroupACL.....	24
3.2.19 Authenticate.....	24
3.2.20 AdjustPrivilege.....	25
3.2.21 GetAllTypeRight	26
3.2.22 OnlineSave	27
3.3 接口软件模块.....	28
3.3.1 AddTransfer.....	28
3.3.2 DelTransfer.....	28
3.3.3 EnumTransfer	29
3.3.4 GetTransferDef.....	29
3.3.5 SetTransferDef	30
3.3.6 GetAllRegisteredTransers	31
3.3.7 QueryTransferAvailableProperties	31
3.3.8 GetPropertiesVal.....	32
3.3.9 SetPropertiesVal	33
3.4 位号模块.....	34
3.4.1 AddVirtualTag	34
3.4.2 DelVirtualTag	35
3.4.3 EnumVirtualTag	35
3.4.4 GetVirtualTagInfo.....	36
3.4.5 SetVirtualTagInfo	37
3.4.6 GetVirtualTagScript.....	38
3.4.7 SetVirtualTagScript	38
3.4.8 GetTagIDByName.....	39
3.4.9 GetTagNameByID.....	39
3.4.10 GetTagIDByPath	40
3.4.11 GetTagPathByID	40
3.4.12 MoveTag.....	41
3.4.13 AddRealTag.....	41
3.4.14 DelRealTag.....	42
3.4.15 EnumRealTag	43
3.4.16 EnumRealTagByTransfer	44
3.4.17 GetRealTagInfo	44
3.4.18 SetRealTagInfo.....	45
3.5 位号数据读写模块.....	46
3.5.1 ReadTagsValue	46
3.5.2 WriteTagsValue	47
3.5.3 WriteTagsValueEx.....	48

3.5.4	WriteTagBulkHisData	49
3.5.5	ReadTagAllMemHis	50
3.5.6	ReadTagsMemHisAtTime	50
3.5.7	ReadTagMemHisInTime	52
3.5.8	ReadTagMemHisSampling.....	53
3.5.9	ReadTagsDiskHisAtTime	54
3.5.10	ReadTagDiskHisInTime	55
3.5.11	ReadTagDiskHisSampling	56
3.5.12	ReadTagsHisAtTime	57
3.5.13	ReadTagHisInTime.....	59
3.5.14	ReadTagHisSampling	60
3.5.15	ReadMaxDiskHisInTime	61
3.5.16	ReadMinDiskHisInTime	62
3.5.17	ReadSumDiskHisInTime	63
3.5.18	ReadAvgDiskHisInTime	64
3.6	区域模块.....	64
3.6.1	AddRegion	64
3.6.2	DelRegion.....	65
3.6.3	GetRegionInfoByPath	65
3.6.4	SetRegionInfo.....	66
3.6.5	MoveRegion	67
3.6.6	EnumChildRegion	68
3.6.7	EnumTagsInRegion	68
3.6.8	Snapshot	70
4	使用指南.....	71
4.1	数据结构.....	71
4.1.1	BASETAGDEF	71
4.1.2	REALTAGDEF	72
4.1.3	VIRTUALTAGDEF.....	73
4.1.4	REGION_DEF	74
4.1.5	REGION_ATTRIBUTE	74
4.1.6	REGION_TREE.....	75
4.1.7	TAGVALUESTATE	76
4.1.8	USER_DEF	76
4.1.9	USERGROUP_DEF.....	77
4.1.10	ACEHEADER.....	78
4.1.11	RIGHT_DEF	78
4.1.12	TYPE_INFO.....	79

4.1.13	TYPE_DEF	79
4.1.14	TRANSFER_DEF	80
4.1.15	TRANSFER_INFO	81
4.1.16	TRANSFER_PROPERTYDEF	81
4.1.17	TRANSFER_PROPERTYVALUE	83
4.2	数据读取策略详解	83
4.2.1	时间段读取策略	83
4.2.2	时间点读取策略	85
4.3	注意事项	87
4.3.1	内存管理	87
4.3.2	有关 FILETIME 时间类型	87
4.3.3	有关错误信息	87
4.3.4	回调函数	88
4.4	应用实例	88
4.4.1	动态加载 DLL	88
4.4.2	静态加载 DLL	89
4.4.3	接口调用举例	90
4.4.4	单线程接口调用举例	92
4.4.5	多线程接口调用举例	93
5	资料版本说明	96

API 接口规范手册

1 前言

1.1 手册简介

本手册是面向用户详细介绍本软件功能以及操作方法，共分为四个部分。其中第一部分为手册简介，简要介绍本手册的内容及使用软件时需要的基本环境；第二部分介绍了 ESP-iSYS 实时数据库网络服务 API 接口的概念及功能；第三部分对该软件的接口函数定义做了详细介绍；第四部分介绍了软件的使用方法及注意事项。

1.2 客户端需求

最低系统配置：

- PentiumIII 866MHz CPU
- 256M 内存
- 1GB 以上可用硬盘空间

推荐系统配置：

- Pentium 4 2.4GHz CPU
- 1GB 内存
- 1GB 以上可用硬盘空间

软件资源要求(操作系统)：

- IE5.0 或以上
- Microsoft Windows 2000+SP4 以上操作系统
- Microsoft.VisualStudio 6.0+SP5 或者以上版本

2 软件概述

2.1 API 模块概念

ESP-iSYS 实时数据库 API 模块由浙江中控软件技术有限公司为二次开发所提供的访问接口，它为二次开发人员提供了方便的接口与实时数据库服务器进行通讯，从而获取所需的信息。API 模块以动态链接库的形式提供给用户，用户使用时需加载相关 DLL，当用户调用其接口时，它自动将用户请求发送给实时数据库服务器。API 模块支持多线程的访问，允许多个连接，但一个线程只有一个用户可以登录。

API 模块与用户、实时数据库服务器的总体关系如下图所示：

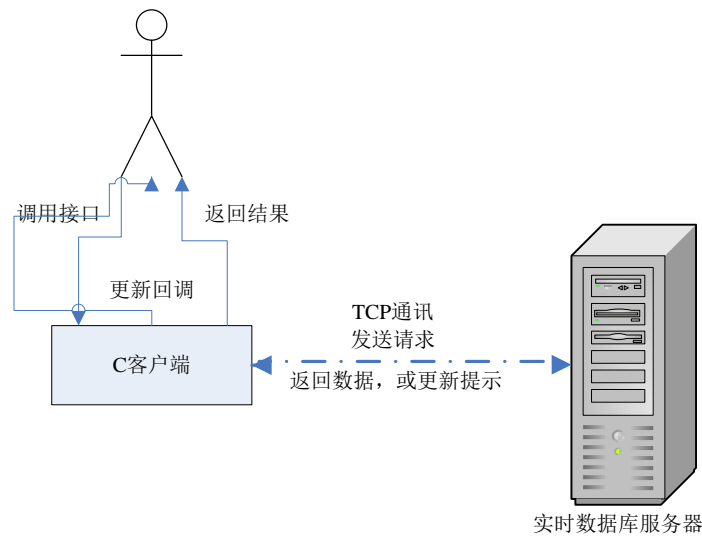


图 2-1 API 模块关系示意图

2.2 API 模块功能概述

API 模块负责与实时数据库服务器之间的通讯，它提供了获取其他客户端软件（如：组态软件）需要的所有信息的接口函数。功能主要包含六个模块：用户管理模块、接口软件模块、位号模块、位号数据读写模块、区域模块、辅助函数模块。

1. 每个模块都有一组对应的 API 来完成用户要求的操作：
2. 用户管理模块：管理所有用户\用户组的权限
3. 接口软件模块：负责对接口软件的相关操作
4. 位号模块：管理所有位号，添加、删除、获取属性\数据值等
5. 位号数据读写模块：管理位号的读写数据操作。
6. 区域模块：提供了对区域的有关操作，更有效的管理利用位号
7. 辅助函数模块：为更好的完成以上操作而需要的基础函数，如连接数据库，回调更新提示函数等

在下面的章节，我们将依次对各接口函数功能做详细介绍。

3 接口定义

3.1 辅助函数模块

3.1.1 GetLastErrorException

描述: 获取最近一次异常错误值。

```
HRESULT GetLastErrorException();
```

参数说明:无。

返回值

返回值	描述
HRESULT 值	即为错误值

注意

如果程序中尚未出现任何异常，函数返回 S_OK。

3.1.2 Connect

描述:连接服务器，若连接成功，返回值即为连接句柄，调用其他接口需要此连接句柄参数。

```
HANDLE Connect( LPWSTR lpServerIPAddr, USHORT  
usServerPort );
```

参数说明

参数	描述
lpServerIPAddr	服务器 IP 地址
usServerPort	服务器端口号

返回值

返回值	描述
NULL	连接失败
非 NULL	连接成功，数值为连接句柄

例子

```
char chIP[MAX_LEN] = "127.0.0.1";
USHORT usPort = 5150;
USES_CONVERSION;
HANDLE handle = Connect( T2OLE(chIP), usPort );
if( NULL == handle )
{
    HRESULT hr = GetLastError();
    PRINTF("连接失败, 错误值为: 0x%08X \r\n", hr);
}
else
{
    PRINTF(" 连接成功 , 连接句柄为 : 0x%08X \r\n",
(DWORD)handle );
}
```

3.1.3 GetVersionNumber

描述: 获取服务器版本号。

```
DWORD GetVersionNumber( HANDLE handle );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。

返回值

返回值	描述
0	获取失败
>0	获取成功, 值即为版本号

例子

```
char chIP[MAX_LEN] = "127.0.0.1";
USHORT usPort = 5150;
USES_CONVERSION;
HANDLE handle = Connect( T2OLE(chIP), usPort );
if( NULL == handle )
{
    HRESULT hr = GetLastError();
    PRINTF("连接失败, 错误值为: 0x%08X \r\n", hr);
}
```

```

    }
    else
    {
        PRINTF("连接成功，连接句柄为: 0x%08X \r\n",
        (DWORD)handle );
        DWORD dw = GetVersionNumber(handle);
        if( 0 == dw )
        {
            HRESULT hr = GetLastError();
            PRINTF("获取失败，错误值为: 0x%08X \r\n", hr);
        }
        else
            PRINTF("获取成功，版本号值为: 0x%08X \r\n",
            dw );
    }
}

```

3.1.4 GetRTDBTime

描述: 获取实时数据库服务器的系统时间。

```
HREUSLT GetRTDBTime( HANDLE handle, FILETIME& ftTime );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
ftTime	返回实时数据库服务器的系统时间

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，此时可能为网络问题

3.1.5 IsConnected

描述: 判断与服务器的连接是否正常。

```
HRESULT IsConnected( HANDLE handle, bool& bConnected );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值
bConnected	连接是否正常的表示，true 为正常，false 为连接异常

返回值

返回值	描述
S_OK	调用成功
其他返回值	调用失败，常见原因是传入的连接句柄无效

3.1.6 Disconnect

描述: 断开与服务器的连接。

```
BOOL Disconnect( HANDLE handle );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。

返回值

返回值	描述
TRUE	断开成功
FALSE	断开失败

3.1.7 GetErrorMgs

描述: 获取错误值的具体解释。

```
HRESULT GetErrorMgs(HANDLE handle, HRESULT hr, LPWSTR* pch );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hr	要查找的错误值
pch	返回的错误信息

返回值

返回值	描述
-----	----

返回值	描述
S_OK	获取成功
其他值	获取失败，此时可能为网络问题

注意

若函数返回 S_OK，参数 pch 返回错误值的具体解释。参数 pch 的内存由 DLL 分配，用户必须调用 CoTaskMemFree 释放。

例子

```
// 假设g_hConnect为连接句柄
void TGetErrorMessage(HRESULT hr)
{
    LPWSTR pchMsg = NULL;
    HRESULT bRes = GetErrorMgs( g_hConnect, hr, &pchMsg );
    if( bRes != S_OK )
        PRINTF("获取失败 \r\n" );
    else
    {
        PRINTF("错误值 0x%08X: %S \r\n", hr, pchMsg );
        ::CoTaskMemFree( pchMsg );
    }
}
```

3.1.8 GetWaitTime

描述：获得当前设置的网络通讯超时时间。

```
DWORD GetWaitTime();
```

参数说明：无。

返回值

返回值	描述
DWORD 值	当前设置的超时时间

注意

网络通讯超时时间默认为 10 秒，即当用户调用接口函数时，接口函数向 ESP-iSYS 端发送请求命令后，若 10 秒内还没有收到回复，则认为网络连接已断开，接口函数直接返回。用户可以通过 SetWaitTime 函数重新设置超时时间。

3.1.9 SetWaitTime

描述: 设置网络通讯超时时间。

```
void SetWaitTime( DWORD dw );
```

参数说明

参数	描述
dw	用户需要设置的超时时间

返回值: 无。

3.1.10 ReqUpdateMessage

描述: 请求组态更新提示。

```
HRESULT ReqUpdateMessage(  
    HANDLE handle,  
    CallbackMessageFunc fpMsgFun,  
    BOOL bFlag  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
fpMsgFun	回调函数指针，定义详见第 4 章 使用指南中的注意事项
bFlag	TRUE 表示请求更新提示，FALSE 表示不请求

返回值

返回值	描述
S_OK	请求成功
其他值	请求失败，返回值即为错误代码。

注意

默认情况下 ESP-iSYS 实时数据库端是不发送更新提示的。请求更新提示是为了当多个客户端对 ESP-iSYS 实时数据库进行组态时，当其它客户端更新了组态信息，本客户端能及时获得通知，从而更新本地的缓存。

例子

```
class COtherMgt
```

```
{
    public:
        static HRESULT TReqUpdateMessage( LPCTSTR
lpCmd )
        { // 假设g_hConnect为连接句柄
            BOOL bFlag = TRUE;
            HRESULT hr = ReqUpdateMessage( g_hConnect,
COtherMgt::Callback, bFlag );
            if( FAILED(hr) )
                PRINTF("请求失败, 错误值为: 0x%08X
\r\n", hr);
            else
                PRINTF("请求成功~\r\n" );
            return S_OK;
        }
        //回调函数
        static void Callback( HANDLE handle, euUpdateType
eu, FILETIME ft )
        {
            PRINTF("系统成功执行回调函数一次~ \r\n");
        }
    };
};
```

3.2 用户管理模块

3.2.1 LogIn

描述: 用户登录。

```
HRESULT LogIn( HANDLE handle, LPWSTR lpUserName,
LPWSTR lpPassword );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserName	用户名称
lpPassword	用户密码

返回值

返回值	描述
S_OK	登录成功
其他值	登录失败, 返回值即为错误代码。

例子

```
// 假设g_hConnect为连接句柄
HRESULT hr = LogIn( g_hConnect, L"Administrator", L"supcon");
```

3.2.2 LogOut

描述: 用户注销。

```
HRESULT LogOut( HANDLE handle );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。

返回值

返回值	描述
S_OK	注销成功
其他值	注销失败，返回值即为错误代码。

3.2.3 SetClientName

描述: 设置指定句柄所对应的连接的客户端名称。

```
HRESULT SetClientName(
HANDLE handle,
LPCWSTR wszClientName
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
wszClientName	客户端名称

返回值

返回值	描述
S_OK	登录成功
其他值	登录失败，返回值即为错误代码。

3.2.4 AddUser

描述: 添加一个用户。


```
HRESULT AddUser(HANDLE handle, USER_DEF* pUser);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
pUser	用户信息结构，其具体定义请详见 USER_DEF

返回值

返回值	描述
S_OK	添加成功
其他值	添加失败，返回值即为错误代码。

例子

```
USER_DEF def;
def.wszUserName = L"user1"
def.wszUserPassword = L"1234";
def.wszUserDesc = L"用户1";
::VariantInit( &def.vCustomData );
def.vCustomData.vt = VT_BSTR;
def.vCustomData.bstrVal = ::SysAllocString(L"自定义数据");

HRESULT hr = AddUser( g_hConnect, &def ); //假设
g_hConnect为连接句柄
SysFreeString( def.vCustomData.bstrVal );
if(FAILED(hr))
    PRINTF("AddUser失败, 错误值: 0x%08X\r\n",hr);
else
    PRINTF("添加用户 %s 成功~\r\n", lpName);
```

3.2.5 DelUser

描述: 删除一个用户。

```
HRESULT DelUser(HANDLE handle, LPWSTR lpUserName);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserName	要删除的用户名称

返回值

返回值	描述
S_OK	删除成功
其他值	删除失败，返回值即为错误代码。

3.2.6 GetUserInfo

描述：获取一个用户信息。

```
HRESULT GetUserInfo(HANDLE handle, LPWSTR lpUserName,  
USER_DEF** ppUser);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserName	要获取的用户名称
ppUser	用于返回用户信息，其具体定义请详见 USER_DEF

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，返回值即为错误代码。此时参数*ppUser 值为 NULL

注意

如果函数返回 S_OK，参数 ppUser 将返回用户信息。指针* ppUser 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 USER_DEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

例子

```
class CUserMgt  
{  
public:  
    //获取一个用户信息  
    static void TGetUserInfo()  
    {  
        USER_DEF* pUserInfo = NULL;  
        HRESULT hr = GetUserInfo( g_hConnect, L"user1",  
&pUserInfo );  
        if( FAILED(hr) )
```

```
        PRINTF("GetUserInfo 失败 , 错误值 :  
0x%08X\r\n", hr);  
        else{  
            PRINTF("GetUserInfo 成功");  
            Clean( pUserInfo );  
        }  
    }  
    static void Clean( USER_DEF& attr )  
    {  
        ::CoTaskMemFree( attr.wszUserName );           //用户  
名称  
        ::CoTaskMemFree ( attr.wszUserPassword ); //用户密  
码  
        ::CoTaskMemFree ( attr.wszUserDesc );          //用户  
描述  
        ::VariantClear (&(attr.vCustomData) );        //用户的自  
定义数据  
    }  
};
```

3.2.7 SetUserDef

描述: 设置一个用户信息。

```
HRESULT SetUserDef(HANDLE handle, USER_DEF* pUser);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
ppUser	要设置的用户定义结构，其具体定义请详见 USER_DEF

返回值

返回值	描述
S_OK	设置成功
其他值	设置失败，返回值即为错误代码。

注意

能设置的信息包括密码、描述、自定义属性。

3.2.8 GetGroupsInUser

描述: 获取用户隶属的用户组。

```
HRESULT GetGroupsInUser(  
    HANDLE handle,  
    LPWSTR lpUserName,  
    DWORD& dwCount,  
    USERGROUP_DEF** lpUserGroups  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserName	用户名称
dwCount	用于返回用户组个数
lpUserGroups	用于返回指定用户所隶属的用户组结构数组，该结构具体定义请参见 USERGROUP_DEF

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，返回值即为错误代码。此时用户组个数为 0，*lpUserGroups 值为 NULL

注意

如果函数返回 S_OK，参数 lpUserGroups 将返回指定用户所隶属的用户组的结构数组。指针*lpUserGroups 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 USERGROUP_DEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.2.9 EnumUser

描述: 枚举用户。

```
HRESULT EnumUser(HANDLE handle, DWORD&  
dwCount, USER_DEF** ppUser);
```

参数说明

参数	描述
----	----

handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	用于返回用户个数
ppUser	用于返回用户数组，用户属性结构的具体定义请参见 USER_DEF

返回值

返回值	描述
S_OK	枚举成功
其他值	枚举失败，返回值即为错误代码。此时用户个数 dwCount 为 0，*ppUser 值为 NULL

注意

如果函数返回 S_OK，参数 ppUser 将返回所有用户属性的数组。指针* ppUser 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 USER_DEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.2.10 AddUserGroup

描述：增加一个用户组。

```
HRESULT AddUserGroup(HANDLE handle,
USERGROUP_DEF* pUserGroup);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
pUserGroup	需要增加的用户组属性，该结构的具体定义请参见 USERGROUP_DEF

返回值

返回值	描述
S_OK	添加用户组成功
其他值	添加失败，返回值即为错误代码。

3.2.11 DelUserGroup

描述：删除一个用户组。

```
HRESULT DelUserGroup(HANDLE handle, LPWSTR
```

```
lpUserGroup);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserGroup	用户组名称

返回值

返回值	描述
S_OK	删除用户组成功
其他值	删除失败，返回值即为错误代码。

3.2.12 GetUserGroupDef

描述：获取一个用户组信息。

```
HRESULT GetUserGroupDef(  
    HANDLE handle,  
    LPWSTR lpUserGroup,  
    USERGROUP_DEF** ppUserGroup  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserGroup	用户组名称
ppUserGroup	用于返回用户组信息，该结构定义具体请参见 USERGROUP_DEF

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，返回值即为错误代码。

注意

如果函数返回 S_OK，参数 ppUserGroup 将返回用户组信息。指针* ppUserGroup 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 USERGROUP_DEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.2.13 SetUserGroupDef

描述: 获取一个用户组信息。

```
HRESULT SetUserGroupDef(HANDLE handle,
USERGROUP_DEF* pUserGroup);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
pUserGroup	用户组属性结构，该结构定义具体请参见 USERGROUP_DEF

返回值

返回值	描述
S_OK	设置成功
其他值	设置失败，返回值即为错误代码。

注意

能设置的信息仅为用户组描述、自定义信息。

3.2.14 AddUserToUserGroup

描述: 将指定用户添加到一个用户组。

```
HRESULT AddUserToUserGroup(
HANDLE handle,
LPWSTR lpUserGroup,
LPWSTR lpUserName
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserGroup	用户组名称
lpUserName	用户名称

返回值

返回值	描述
S_OK	将用户加入到用户组成功
其他值	将用户加入到用户组失败，返回值即为错误代码。

3.2.15 RemoveUserFromUserGroup

描述：从一个用户组删除一个用户。

```
HRESULT RemoveUserFromUserGroup (  
    HANDLE handle,  
    LPWSTR lpUserGroup,  
    LPWSTR lpUserName  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserGroup	用户组名称
lpUserName	用户名称

返回值

返回值	描述
S_OK	将用户从用户组移除成功
其他值	将用户从用户组移除失败，返回值即为错误代码。

3.2.16 EnumUserGroup

描述：枚举用户组。

```
HRESULT EnumUserGroup(  
    HANDLE handle,  
    DWORD& dwCount,  
    USERGROUP_DEF** ppUserGroup  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	用于返回用户组个数
ppUserGroup	用于返回所有用户组的属性数组，该结构的具体定义请参见 USERGROUP_DEF

返回值

返回值	描述
S_OK	枚举用户组成功，此时返回所有用户组
其他值	枚举用户组失败，返回值即为错误代码。此时 dwCount 值为 0, *ppUserGroup 值为 NULL

注意

如果函数返回 S_OK，参数 ppUserGroup 将返回所有用户组的属性数组。指针* ppUserGroup 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 USERGROUP_DEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.2.17 EnumUserACL

描述：枚举一个指定用户的权限列表。

```
HRESULT EnumUserACL(
    HANDLE handle,
    LPWSTR lpUserName,
    DWORD& dwCount,
    ACEHEADER** ppACL
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserName	用户名称
dwCount	用于返回权限个数
ppACL	用于返回权限结构数组，该结构的具体定义请参见 ACEHEADER

返回值

返回值	描述
S_OK	枚举用户权限成功
其他值	枚举用户权限失败，返回值即为错误代码。此时 dwCount 值为 0, *ppACL 值为 NULL

注意

如果函数返回 S_OK，参数 ppACL 将返回用户权限结构数组。指针* ppACL 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 ACEHEADER 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用

CoTaskMemFree 函数释放。

3.2.18 EnumUserGroupACL

描述：枚举一个指定用户组的权限列表。

```
HRESULT EnumUserGroupACL(  
    HANDLE handle,  
    LPWSTR lpUserGroup,  
    DWORD& dwCount,  
    ACEHEADER** ppACL  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpUserName	用户组名称
dwCount	用于返回权限个数
ppACL	用于返回权限结构数组，该结构的具体定义请参见 ACEHEADER

返回值

返回值	描述
S_OK	枚举用户组权限成功
其他值	枚举用户组权限失败，返回值即为错误代码。此时 dwCount 值为 0，*ppACL 值为 NULL

注意

如果函数返回 S_OK，参数 ppACL 将返回用户组权限结构数组。指针* ppACL 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 ACEHEADER 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.2.19 Authenticate

描述：验证一个用户/用户组的操作权限。

```
HRESULT Authenticate(  
    HANDLE handle,  
    LPWSTR lpName,  
    LPWSTR wszPath,
```

```
DWORD dwCount,  
LPWSTR* pszRight,  
HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpName	用户\用户组名称
wszPath	要验证的对象的路径
dwCount	要验证的权限个数
pszRight	要验证的权限名称列表
ppRes	用户返回各权限的验证结果数组，该数组大小为 dwCount，其各个位置上的值是 pszRight 数组对应位置上的权限的验证结果。

返回值

返回值	描述
S_OK	验证操作执行成功，ppRes 参数返回各权限的验证结果。仅当验证结果为 S_OK 时，表示用户对指定对象拥有 pszRight 数组对应位置上的权限。
其他值	验证操作执行失败，返回值即为错误代码。此时*ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回各权限验证结果的数组。指针* ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.2.20 AdjustPrivilege

描述：调整一个用户/用户组的操作权限。

```
HRESULT AdjustPrivilege(  
HANDLE handle,  
LPWSTR lpName,  
LPWSTR wszPath,  
BOOL bAdd,  
BOOL bGrand,  
DWORD dwCount,
```

```
LPWSTR* pszRight,  
HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpName	用户\用户组名称
wszPath	要调整的对象的路径
bAdd	指定要增加权限还是删除权限。 指定该参数为 TRUE，表明需要增加权限。 指定该参数为 FALSE，表明需要删除权限。
bGrand	指定要调整允许权限还是拒绝权限。 指定该参数为 TRUE，表明需要调整允许权限。 指定该参数为 FALSE，表明需要调整拒绝权限。
dwCount	要调整的权限个数
pszRight	要调整的权限名称列表
ppRes	用户返回调整结果，大小为 dwCount，其各个位置上的值是 pszRight 数组对应位置上的权限的调整结果。

返回值

返回值	描述
S_OK	调整操作执行成功，ppRes 参数返回各权限的调整结果。仅当调整结果为 S_OK 时，表示调整 pszRight 数组对应位置上的权限成功。
其他值	调整操作执行失败，返回值即为错误代码。此时*ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回各权限调整结果的数组。指针* ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.2.21 GetAllTypeRight

描述：获取所有类型以及隶属于各类型的权限。

```
HRESULT GetAllTypeRight(  
HANDLE handle,  
DWORD &dwCount,  
TYPE_INFO** ppTypeInfo
```

```
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	用于返回类型个数
ppTypeInfo	用于返回所有类型属性结构数组，该结构中包括了隶属于该类型的权限。该结构的具体定义请参见 TYPE_INFO

返回值

返回值	描述
S_OK	获取所有类型成功
其他值	获取失败，返回值即为错误代码。此时 dwCount 值为 0，*ppTypeInfo 值为 NULL

注意

如果函数返回 S_OK，参数 ppTypeInfo 将返回所有类型属性结构数组。指针* ppTypeInfo 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TYPE_INFO 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

TYPE_INFO 结构中包括了隶属于类型的权限列表 RIGHT_DEF* pRightDef。若隶属于类型的权限个数不为 0，用户需要调用 CoTaskMemFree 函数释放指针 pRightDef 的内存，同时，用户还应该释放 RIGHT_DEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.2.22 OnlineSave

描述：在线保存所有组态信息。

```
HRESULT OnlineSave(HANDLE handle);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。

返回值

返回值	描述
S_OK	保存组态信息成功
其他值	保存组态信息失败，返回值即为错误

返回值	描述
	代码。

3.3 接口软件模块

3.3.1 AddTransfer

描述: 增加一个接口。

```
HRESULT AddTransfer(  
    HANDLE handle,  
    TRANSFER_DEF* pTransDef,  
    DWORD& hTransferID  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
pTransDef	接口属性结构。该结构的具体定义请参见 TRANSFER_DEF
hTransferID	用于返回接口 ID。接口 ID 是接口的唯一标识

返回值

返回值	描述
S_OK	添加接口成功
其他值	添加接口失败，返回值即为错误代码。

3.3.2 DelTransfer

描述: 删除一个接口。

```
HRESULT DelTransfer(HANDLE handle, LPWSTR  
lpTransName);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpTransName	需要删除的接口的名称

返回值

返回值	描述
-----	----

返回值	描述
S_OK	删除接口成功
其他值	删除接口失败，返回值即为错误代码。

3.3.3 EnumTransfer

描述：枚举接口。

```
HRESULT EnumTransfer(  
HANDLE handle,  
DWORD& dwCount,  
TRANSFER_DEF** ppTransDef  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	用于返回接口个数
ppTransDef	用于返回所有接口的属性，该结构的具体定义请参见 TRANSFER_DEF

返回值

返回值	描述
S_OK	枚举接口成功，ppTransDef 返回所有接口的属性
其他值	枚举接口失败，返回值即为错误代码。此时 dwCount 值为 0，*ppTransDef 值为 NULL

注意

如果函数返回 S_OK，参数 ppTransDef 将返回所有接口属性的数组。指针* ppTransDef 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TRANSFER_DEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.3.4 GetTransferDef

描述：获取一个指定接口的组态信息。

```
HRESULT GetTransferDef(  
HANDLE handle,  
LPWSTR lpTransName,
```

```
TRANSFER_DEF** ppTransDef
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpTransName	接口名称
ppTransDef	用于返回接口属性，该结构的具体定义请参见 TRANSFER_DEF

返回值

返回值	描述
S_OK	获取接口信息成功，ppTransDef 返回接口的属性
其他值	获取接口信息失败，返回值即为错误代码。此时*ppTransDef 值为 NULL

注意

如果函数返回 S_OK，参数 ppTransDef 将返回接口属性。指针* ppTransDef 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TRANSFER_DEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.3.5 SetTransferDef

描述：设置一个指定接口的组态信息。

```
HRESULT SetTransferDef (HANDLE handle, TRANSFER_DEF*
pTransDef);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
ppTransDef	要设置的接口属性，该结构的具体定义请参见 TRANSFER_DEF

返回值

返回值	描述
S_OK	设置接口信息成功
其他值	设置接口信息失败，返回值即为错误代码。

注意

能设置的仅为接口描述。

3.3.6 GetAllRegisteredTransers

描述: 获取所有已经注册的接口软件的列表。

```
HRESULT GetAllRegisteredTransers(  
    HANDLE handle,  
    DWORD& dwCount,  
    TRANSFER_INFO** ppTransRegInfo  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	用于返回接口个数
ppTransRegInfo	用于返回已注册的接口信息，该结构的具体定义请参见 TRANSFER_INFO

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，返回值即为错误代码。此时 dwCount 值为 0，*ppTransRegInfo 值为 NULL

注意

如果函数返回 S_OK，参数 ppTransRegInfo 将返回所有已注册的接口软件列表。指针 *ppTransRegInfo 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TRANSFER_INFO 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.3.7 QueryTransferAvailableProperties

描述: 获取指定接口的自定义属性列表。

```
HRESULT QueryTransferAvailableProperties(  
    HANDLE handle,  
    LPWSTR lpTransName,  
    DWORD& dwCount,  
    TRANSFER_PROPERTYDEF** ppProperDef
```

```
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpTransName	接口名称
dwCount	用于返回接口自定义属性个数
ppTransRegInfo	用于返回自定义属性数组，该结构的具体定义请参见 TRANSFER_PROPERTYDEF

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，返回值即为错误代码。此时 dwCount 值为 0，* ppProperDef 值为 NULL

注意

如果函数返回 S_OK，参数 ppProperDef 将返回指定接口的自定义属性列表。指针* ppProperDef 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TRANSFER_PROPERTYDEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.3.8 GetPropertiesVal

描述：获取指定接口的自定义属性值。

```
HRESULT GetPropertiesVal(
    HANDLE handle,
    LPWSTR lpTransName,
    DWORD& dwCount,
    TRANSFER_PROPERTYVALUE** ppProperty
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpTransName	接口名称
dwCount	用于返回接口自定义属性值个数
pProperty	用于返回自定义属性值数组，该结构的具体定义请参见 TRANSFER_PROPERTYVALUE

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败, 返回值即为错误代码。此时 dwCount 值为 0, ppProperty 值为 NULL

注意

如果函数返回 S_OK, 参数 ppProperty 将返回指定接口的自定义属性值列表。指针* ppProperty 的内存由 DLL 分配, 用户使用完后必须调用 CoTaskMemFree 进行释放。同时, 用户还应该释放 TRANSFER_PROPERTYVALUE 结构内部成员的内存, 其中 VARIANT 类型成员调用 VariantClear 函数释放, 字符串类型成员调用 CoTaskMemFree 函数释放。

3.3.9 SetPropertyVal

描述: 设置指定接口的自定义属性值。

```
HRESULT _stdcall SetPropertyVal(  
    HANDLE handle,  
    LPWSTR lpTransName,  
    DWORD dwCount,  
    TRANSFER_PROPERTYVALUE* pProperty,  
    HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpTransName	接口名称
dwCount	接口自定义属性值个数
pProperty	自定义属性值数组, 该结构的具体定义请参见 TRANSFER_PROPERTYVALUE
ppRes	用于返回设置结构数组。

返回值

返回值	描述
S_OK	设置成功。ppRes 返回各个属性值的设置结果。仅仅当设置结果为 S_OK 时, 表明 pProperty 对应位置上的属性值设置成功。
其他值	设置失败, 返回值即为错误代码。此*ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回设置结果数组。指针* ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.4 位号模块

3.4.1 AddVirtualTag

描述：增加一组虚位号。

```
HRESULT AddVirtualTag(  
    HANDLE handle,  
    DWORD dwCount,  
    VIRTUALTAGDEF* pVirDef,  
    HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	虚位号个数
pVirDef	虚位号属性结构，该结构的具体定义请参见 VIRTUALTAGDEF
ppRes	返回结果数组，大小等于 dwCount。当值为 S_OK 时，表明 pVirDef 数组对应位置上的虚位号增加成功。

返回值

返回值	描述
S_OK	添加执行成功，返回 HRESULT 数组对应每个虚位号的具体添加结果，仅当值为 S_OK 时，表示对应虚位号添加成功
其他值	添加失败，返回值即为错误代码。此时*ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回操作结果数组。指针* ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.4.2 DelVirtualTag

描述: 删除一组虚位号。

```
HRESULT DelVirtualTag(  
    HANDLE handle,  
    DWORD dwCount,  
    HTAG* hTags,  
    HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	虚位号个数
hTags	要删除的虚位号句柄数组
ppRes	返回结果数组，大小等于 dwCount。当返回结果为 S_OK 时，表明 hTags 数组对应位置上的虚位号删除成功。

返回值

返回值	描述
S_OK	删除执行成功，返回 HRESULT 数组对应每个虚位号的具体删除，仅当值为 S_OK 时，表示对应虚位号删除成功
其他值	删除失败，返回值即为错误代码。此时*ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回操作结果数组。指针* ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.4.3 EnumVirtualTag

描述: 枚举虚位号。

```
HRESULT EnumVirtualTag(  
    HANDLE handle,  
    DWORD& dwCount,  
    VIRTUALTAGDEF** ppVirDef  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	虚位号个数
ppVirDef	返回虚位号属性结构数组，该结构的具体定义请参见 VIRTUALTAGDEF

返回值

返回值	描述
S_OK	枚举执行成功
其他值	枚举失败，返回值即为错误代码。此时 *ppVirDef 值为 NULL

注意

如果函数返回 S_OK, 参数 ppVirDef 将返回所有虚位号属性列表。指针* ppVirDef 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 VIRTUALTAGDEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.4.4 GetVirtualTagInfo

描述：获取一组虚位号属性。

```

HRESULT GetVirtualTagInfo(
    HANDLE handle,
    DWORD dwCount,
    HTAG* hTags,
    HRESULT** ppRes,
    VIRTUALTAGDEF** ppVirDef
);

```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	虚位号个数
hTags	虚位号句柄数组
ppRes	返回获取结果数组，大小等于 dwCount。当值为 S_OK 时，表明数组 hTags 对应位置上的位号其属性获取成功，属性信息在数组 ppVirDef 的对应位置上。

ppVirDef	返回虚位号属性结构数组，该结构的具体定义请参见 VIRTUALTAGDEF
----------	---

返回值

返回值	描述
S_OK	获取操作执行成功，返回 HRESULT 数组对应每个虚位号属性的获取结果，仅当值为 S_OK 时，表示对应虚位号属性获取成功
其他值	获取操作执行失败，返回值即为错误代码。此时 *ppRes 和 *ppVirDef 值为 NULL

注意

如果函数返回 S_OK，参数 ppVirDef 将返回虚位号属性列表，参数 ppRes 将返回各个位号的获取结果。指针 *ppVirDef 和 *ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 VIRTUALTAGDEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.4.5 SetVirtualTagInfo

描述：设置一组虚位号的组态信息。

```
HRESULT SetVirtualTagInfo(
    HANDLE handle,
    DWORD dwCount,
    VIRTUALTAGDEF* pVirDef,
    HRESULT** ppRes
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	虚位号个数
pVirDef	要设置的虚位号定义结构数组，该结构的具体定义请参见 VIRTUALTAGDEF
ppRes	返回设置结果数组，大小等于 dwCount。结果数组中的值为 S_OK 时，表明数组 pVirDef 对应位置上的位号其属性设置成功。

返回值

返回值	描述
S_OK	设置操作执行成功，返回 HRESULT 数组对应

返回值	描述
	每个虚位号属性的设置结果，仅当值为 S_OK 时，表示对应虚位号属性设置成功
其他值	设置操作执行失败，返回值即为错误代码。此时 ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回各个位号的操作结果。指针*ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.4.6 GetVirtualTagScript

描述：获取一个虚位号的脚本内容。

```
HRESULT GetVirtualTagScript(HANDLE handle, HTAG hTag,
LPWSTR* pszScript );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	虚位号句柄
pszScript	存返回结果，虚位号脚本内容

返回值

返回值	描述
S_OK	获取成功， pszScript 存脚本内容
其他值	获取失败，返回值即为错误代码。此时 pszScript 值为 NULL

注意

如果函数返回 S_OK，参数 pszScript 将返回脚本内容。指针 pszScript 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.4.7 SetVirtualTagScript

描述：设置一个虚位号的脚本内容。

```
HRESULT SetVirtualTagScript( HANDLE handle, HTAG hTag,
LPWSTR lpScript );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	虚位号句柄
lpScript	要设置的虚位号脚本内容

返回值

返回值	描述
S_OK	设置成功
其他值	设置失败，返回值即为错误代码。

3.4.8 GetTagIDByName

描述：根据一个位号名字获取 ID。

```
HRESULT GetTagIDByName( HANDLE handle, LPWSTR
lpName, HTAG& hTag);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpName	位号名称
hTag	用于返回位号句柄

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，返回值即为错误代码。

3.4.9 GetTagNameByID

描述：根据一个位号 ID 获取其名字。

```
HRESULT GetTagNameByID( HANDLE handle, HTAG hTag,
LPWSTR* pszName );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
pszName	存返回的位号名称

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，返回值即为错误代码。

注意

如果函数返回 S_OK，参数 pszName 将返回位号名称。指针 pszName 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.4.10 GetTagIDByPath

描述：根据一个位号路径获得 ID。

```
HRESULT GetTagIDByPath( HANDLE handle, LPWSTR  
lpPath, HTAG& hTag );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpPath	位号路径
hTag	存返回的位号句柄

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，返回值即为错误代码。

3.4.11 GetTagPathByID

描述：根据一个位号 ID 获取路径。

```
HRESULT GetTagPathByID( HANDLE handle, HTAG hTag,  
LPWSTR* pszPath );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
pszPath	存返回的位号路径

返回值

返回值	描述
S_OK	获取成功
其他值	获取失败，返回值即为错误代码。

注意

如果函数返回 S_OK，参数 pszPath 将返回位号路径。指针 pszPath 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.4.12 MoveTag

描述：移动一个位号在对象树上的位置。

```
HRESULT MoveTag( HANDLE handle, HTAG hTag, LPWSTR  
wszFatherPath );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
wszFatherPath	位号移动到目的对象的路径

返回值

返回值	描述
S_OK	移动成功
其他值	移动失败，返回值即为错误代码。

3.4.13 AddRealTag

描述：增加一组实位号。

```
HRESULT AddRealTag(  
HANDLE handle,  
DWORD dwCount,  
REALTAGDEF* pRealDef,  
HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。

dwCount	实位号个数
pRealDef	要添加的实位号属性结构数组，该结构的具体定义请参见 REALTAGDEF
ppRes	返回结果数组，大小等于 dwCount。当值 S_OK 时，表明 pRealDef 数组对应位置上的位号增加成功。

返回值

返回值	描述
S_OK	添加执行成功，返回 HRESULT 数组对应每个实位号的具体添加结果，仅当值为 S_OK 时，表示对应实位号添加成功
其他值	添加失败，返回值即为错误代码。此时*ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回操作结果数组。指针* ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.4.14 DelRealTag

描述：删除一组实位号。

```
HRESULT DelRealTag(  
    HANDLE handle,  
    DWORD dwCount,  
    HTAG* hTags,  
    HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	实位号个数
hTags	要删除的实位号句柄数组
ppRes	返回结果数组，大小等于 dwCount。当返回结果为 S_OK 时，表明 hTags 数组对应位置上的实位号删除成功。

返回值

返回值	描述
-----	----

返回值	描述
S_OK	删除执行成功，返回 HRESULT 数组对应每个实位号的具体删除，仅当值为 S_OK 时，表示对应实位号删除成功
其他值	删除失败，返回值即为错误代码。此时*ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回操作结果数组。指针* ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.4.15 EnumRealTag

描述：枚举实位号。

```
HRESULT EnumRealTag(  
    HANDLE handle,  
    DWORD& dwCount,  
    REALTAGDEF** ppRealDef  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	用于返回实位号个数
ppRealDef	返回实位号属性结构数组，该结构的具体定义请参见 REALTAGDEF

返回值

返回值	描述
S_OK	枚举执行成功
其他值	枚举失败，返回值即为错误代码。此时* ppRealDef 值为 NULL

注意

如果函数返回 S_OK，参数 ppRealDef 将返回所有实位号属性列表。指针* ppRealDef 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 REALTAGDEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.4.16 EnumRealTagByTransfer

描述: 枚举指定接口下的实位号。

```
HRESULT EnumRealTagByTransfer(  
    HANDLE handle,  
    LPWSTR lpInterfaceName,  
    DWORD& dwCount,  
    REALTAGDEF** ppRealDef  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
lpInterfaceName	接口软件名称
dwCount	用于返回实位号个数
ppRealDef	返回实位号属性结构数组，该结构的具体定义请参见 REALTAGDEF

返回值

返回值	描述
S_OK	枚举执行成功
其他值	枚举失败，返回值即为错误代码。此时* ppRealDef 值为 NULL

注意

如果函数返回 S_OK，参数 ppRealDef 将返回指定接口下的实位号属性列表。指针* ppRealDef 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 REALTAGDEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.4.17 GetRealTagInfo

描述: 获取一组实位号属性。

```
HRESULT GetRealTagInfo(  
    HANDLE handle,  
    DWORD dwCount,  
    HTAG* hTags,  
    HRESULT** ppRes,  
    REALTAGDEF** ppRealDef
```

```
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	实位号个数
hTags	实位号句柄数组
ppRes	返回获取结果数组，大小等于 dwCount。当值为 S_OK 时，表明数组 hTags 对应位置上的位号其属性获取成功，属性信息在数组 ppRealDef 的对应位置上。
ppRealDef	返回实位号属性结构数组，该结构的具体定义请参见 REALTAGDEF

返回值

返回值	描述
S_OK	获取操作执行成功，返回 HRESULT 数组对应每个实位号属性的获取结果，仅当值为 S_OK 时，表示对应实位号属性获取成功
其他值	获取操作执行失败，返回值即为错误代码。此时 *ppRes 和 *ppRealDef 值为 NULL

注意

如果函数返回 S_OK，参数 ppRealDef 将返回实位号属性列表，参数 ppRes 将返回各个位号的获取结果。指针 *ppRealDef 和 *ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 REALTAGDEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.4.18 SetRealTagInfo

描述：设置一组实位号的组态信息。

```
HRESULT SetRealTagInfo(  
    HANDLE handle,  
    DWORD dwCount,  
    REALTAGDEF* pRealDef,  
    HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	实位号个数
pRealDef	要设置的实位号定义结构数组，该结构的具体定义请参见 REALTAGDEF
ppRes	返回设置结果数组，大小等于 dwCount。结果数组中的值为 S_OK 时，表明数组 pRealDef 对应位置上的位号其属性设置成功。

返回值

返回值	描述
S_OK	设置操作执行成功，返回 HRESULT 数组对应每个实位号属性的设置结果，仅当值为 S_OK 时，表示对应实位号属性设置成功
其他值	设置操作执行失败，返回值即为错误代码。此时 ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回各个位号的设置结果。指针*ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.5 位号数据读写模块

3.5.1 ReadTagsValue

描述：读取一批位号的实时数据。

```
HRESULT ReadTagsValue(  
    HANDLE handle,  
    DWORD dwCount,  
    HTAG *hTags,  
    HRESULT** ppRes,  
    TAGVALSTATE** ppValue  
);
```

参数说明

参数	描述
----	----

handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	位号个数
hTags	位号句柄数组
ppRes	返回各个位号实时值获取结果
ppValue	返回各个位号的实时值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
S_OK	获取操作执行成功，每一个 HRESULT 值对应一个位号的获取实时值结果，仅当值为 S_OK 值，表示对应实位号获取实时值成功
其他值	获取操作执行失败，返回值即为错误代码。此时 ppRes 和 ppValue 值为 NULL

注意

如果函数返回 S_OK，参数 ppValue 将返回各个位号的实时值，参数 ppRes 将返回各个位号的读取结果。指针* ppValue 和*ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.5.2 WriteTagsValue

描述：写入一批位号的实时数据。

```
HRESULT WriteTagsValue(  
    HANDLE handle,  
    DWORD dwCount,  
    HTAG *hTags,  
    VARIANT *pvtValue,  
    HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	位号个数
hTags	位号句柄数组

pvtValue	要写入的数据值结构数组
ppRes	返回各个位号写入结果

返回值

返回值	描述
S_OK	写入操作执行成功，每一个 HRESULT 值对应一个位号的写入实时值结果，仅当值为 S_OK 值，表示对应实位号写入实时值成功
其他值	写入操作执行失败，返回值即为错误代码。此时 ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回操作结果数组。指针* ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.5.3 WriteTagsValueEx

描述：写入一批位号的实时数据。

```
HRESULT WriteTagsValueEx(  
    HANDLE handle,  
    DWORD dwCount,  
    HTAG *hTags,  
    VARIANT *pvtValue,  
    FILETIME *pftTime,  
    WORD *pwQuality,  
    HRESULT** ppRes  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	位号个数
hTags	位号句柄数组
pvtValue	要写入的数据值结构数组
pftTime	要写入的时间戳数组
pwQuality	要写入的质量码数组
ppRes	返回各个位号写入结果

返回值

返回值	描述
S_OK	写入操作执行成功，每一个 HRESULT 值对应一个位号的写入实时值结果，仅当值为 S_OK 值，表示对应实位号写入实时值成功
其他值	写入操作执行失败，返回值即为错误代码。此时 ppRes 值为 NULL

注意

如果函数返回 S_OK，参数 ppRes 将返回操作结果数组。指针* ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。

3.5.4 WriteTagBulkHisData

描述：写入一个位号的一批过时数据。

```
HRESULT WriteTagBulkHisData(  
    HANDLE handle,  
    HTAG hTag,  
    DWORD dwCount,  
    VARIANT *pvtValue,  
    FILETIME *pftTime,  
    WORD *pwQuality  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	需要写入的位号的句柄
dwCount	位号个数
pvtValue	要写入的数据值结构数组
pftTime	要写入的时间戳数组
pwQuality	要写入的质量码数组

返回值

返回值	描述
S_OK	写入操作执行成功
其他值	写入操作执行失败，返回值即为错误代码。

注意

调用该函数写入过时数据时候，有两个限制：

1. 所写入的数据必须是过时数据，也就是说已经有比这些数据更新的数据已经上传到了磁盘历史组件；
2. 所写入的数据的时间范围最大不能超过当前时间 1 个月，否则将会返回时间戳错误标记。

3.5.5 ReadTagAllMemHis

描述：读取单个位号所有的内存历史数据。

```
HRESULT ReadTagAllMemHis(  
    HANDLE handle,  
    HTAG hTag,  
    DWORD &dwCount,  
    TAGVALSTATE** ppValue  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
dwCount	返回位号的内存历史数据个数
ppValue	返回位号的所有内存历史数据值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。此时 dwCount 值为 0，ppValue 值为 NULL

注意

如果函数返回 S_OK，参数 ppValue 将返回位号的所有内存历史数据值。指针* ppValue 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.5.6 ReadTagsMemHisAtTime

描述：读取一批位号在某个时刻的内存历史值。

```
HRESULT ReadTagsMemHisAtTime(  
    HANDLE handle,  
    DWORD dwCount,  
    HTAG* hTags,  
    FILETIME ftTime,  
    DWORD wFlag,  
    DWORD dwTimeInterval,  
    HRESULT** ppRes,  
    TAGVALSTATE** ppValue  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
dwCount	需要读取的位号个数
hTags	需要读取的位号句柄数组
ftTime	指定时间点
wFlag	读取策略 0x0000: SAMPLE_BEFORE(前值) 0x0001: SAMPLE_AFTER(后值) 0x0002: SAMPLE_INTERPLOATE (插值)
dwTimeInterval	采样点和实际值之间的允许的时间差，毫秒为单位，填 0 则忽略该参数。
ppRes	返回每个位号的读取结果
ppValue	返回各个位号指定时间点上的内存历史数据值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。此时 ppRes 值为 NULL，ppValue 值为 NULL

注意

如果函数返回 S_OK, 参数 ppValue 将返回各个实位号在指定时间点上的内存历史值, 参数 ppRes 将返回各个位号的读取结果。指针* ppValue 和*ppRes 的内存由 DLL 分配, 用户使用完后必须调用 CoTaskMemFree 进行释放。同时, 用户还应该释放 TAGVALSTATE 结构内部成员的内存, 其中 VARIANT 类型成员调用 VariantClear 函数释放, 字符串类型成员调用 CoTaskMemFree 函数释放。

如果指定时间点 `ftTime` 有值，实时数据库端将直接返回该值。如果指定时间点 `ftTime` 上没有值，则利用指定的读取策略 `wFlag` 进行取值。若传入参数 `dwTimeInterval` 不为 0，那么还要计算最终取到的值的时间点与指定时间点 `ftTime` 的偏差，要是偏差在 `dwTimeInterval` 范围内，返回该值；否则返回指定时间点没有值的错误。

3.5.7 ReadTagMemHisInTime

描述：读取单个位号在某个时刻的内存历史值。

```
HRESULT ReadTagMemHisInTime(  
    HANDLE handle,  
    HTAG hTag,  
    FILETIME ftBegin,  
    FILETIME ftEnd,  
    DWORD wFlag,  
    DWORD &dwCount,  
    TAGVALSTATE** ppValue  
);
```

参数说明

参数	描述
<code>handle</code>	连接句柄。 <code>Connect</code> 函数连接成功后的返回值。
<code>hTag</code>	位号句柄
<code>ftBegin</code>	开始时间点
<code>ftEnd</code>	结束时间点
<code>wFlag</code>	读取策略 0x0000:BOUND_NONE(不含左右边界) 0x0001:BOUND_LEFT(含左边界) 0x0002:BOUND_RIGHT(含右边界) 0x0003:BOUND_BOTH(含左右边界)
<code>dwCount</code>	返回的数据个数
<code>ppValue</code>	返回位号在指定时间段的内存历史数据值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
<code>S_OK</code>	读取执行成功
其他值	读取执行失败，返回值即为错误代码。此时 <code>dwCount</code> 值为 0， <code>ppValue</code> 值为 NULL

注意

如果函数返回 S_OK，参数 ppValue 将返回位号在指定时间段的内存历史数据值。指针* ppValue 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.5.8 ReadTagMemHisSampling

描述：采样读取一个位号内存历史值。

```
HRESULT ReadTagMemHisSampling(  
    HANDLE handle,  
    HTAG hTag,  
    FILETIME ftBegin,  
    FILETIME ftEnd,  
    DWORD dwInterval,  
    DWORD wFlag,  
    DWORD dwTimeDiff,  
    DWORD &dwCount,  
    TAGVALSTATE** ppValue  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
ftBegin	开始时间点
ftEnd	结束时间点
dwInterval	采样间隔
wFlag	读取策略 0x0000:SAMPLE_BEFORE(前值) 0x0001:SAMPLE_AFTER(后值) 0x0002:SAMPLE_INTERPLOATE (插值)
dwTimeDiff	采样点和实际值之间的允许的时间差，毫秒为单位，填 0 则忽略该参数。
dwCount	返回的数据个数
ppValue	返回位号在各采样时间点上的内存历史数据值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。此时 dwCount 值为 0，ppValue 值为 NULL

注意

如果函数返回 S_OK，参数 ppValue 将返回位号在各采样时间点上的内存历史数据值。指针*ppValue 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

实时数据库端首先会根据采样间隔 dwInterval 对给定时间段进行分割，得到等间隔的时间点链表。然后读取各个时间点上的值，组成采样值数组返回。在读取每个时间点上的值时，如果指定时间点有值，将直接取该值。如果指定时间点上没有值，则利用指定的采样策略 wFlag 进行取值。若传入参数 dwTimeDiff 不为 0，那么还要计算最终取到的值的时间点与所要读取的时间点的偏差，要是偏差在 dwTimeDiff 范围内，取该值；否则忽略该值。

3.5.9 ReadTagsDiskHisAtTime

描述：读取一批位号在某个时刻的磁盘历史值。

```
HRESULT ReadTagsDiskHisAtTime (  
    HANDLE handle,  
    DWORD dwCount,  
    HTAG* hTags,  
    FILETIME ftTime,  
    DWORD wFlag,  
    DWORD dwTimeInterval,  
    HRESULT** ppRes,  
    TAGVALSTATE** ppValue  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
dwCount	需要读取的位号个数
hTags	需要读取的位号句柄数组
ftTime	指定时间点

wFlag	读取策略 0x0000:SAMPLE_BEFORE(前值) 0x0001:SAMPLE_AFTER(后值) 0x0002:SAMPLE_INTERPLOATE (插值)
dwTimeInterval	采样点和实际值之间的允许的时间差，毫秒为单位，填 0 则忽略该参数。
ppRes	返回每个位号的读取结果
ppValue	返回各个位号指定时间点上的磁盘历史数据值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。此时 ppRes 值为 NULL，ppValue 值为 NULL

注意

如果函数返回 S_OK, 参数 ppValue 将返回各个实位号在指定时间点上的磁盘历史值, 参数 ppRes 将返回各个位号的读取结果。指针* ppValue 和*ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

如果指定时间点 ftTime 有值，实时数据库端将直接返回该值。如果指定时间点 ftTime 上没有值，则利用指定的读取策略 wFlag 进行取值。若传入参数 dwTimeInterval 不为 0，那么还要计算最终取到的值的时间点与指定时间点 ftTime 的偏差，要是偏差在 dwTimeInterval 范围内，返回该值；否则返回指定时间点没有值的错误。

3.5.10 ReadTagDiskHisInTime

描述：读取单个位号在某个时刻的磁盘历史值。

```

HRESULT ReadTagDiskHisInTime (
    HANDLE handle,
    HTAG hTag,
    FILETIME ftBegin,
    FILETIME ftEnd,
    DWORD wFlag,
    DWORD &dwCount,
    TAGVALSTATE** ppValue
);

```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
ftBegin	开始时间点
ftEnd	结束时间点
wFlag	读取策略 0x0000:BOUND_NONE(不含左右边界) 0x0001:BOUND_LEFT (含左边界) 0x0002:BOUND_RIGHT(含右边界) 0x0003:BOUND_BOTH (含左右边界)
dwCount	返回的数据个数
ppValue	返回位号在指定时间段的磁盘历史数据值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。此时 dwCount 值为 0，ppValue 值为 NULL

注意

如果函数返回 S_OK，参数 ppValue 将返回位号在指定时间段的磁盘历史数据值。指针* ppValue 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.5.11 ReadTagDiskHisSampling

描述：采样读取一个位号磁盘历史值。

```
HRESULT ReadTagDiskHisSampling (  
    HANDLE handle,  
    HTAG hTag,  
    FILETIME ftBegin,  
    FILETIME ftEnd,  
    DWORD dwInterval,  
    DWORD wFlag,  
    DWORD dwTimeDiff,  
    DWORD &dwCount,  
    TAGVALSTATE** ppValue
```

);

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
ftBegin	开始时间点
ftEnd	结束时间点
dwInterval	采样间隔
wFlag	读取策略 0x0000: SAMPLE_BEFORE(前值) 0x0001: SAMPLE_AFTER(后值) 0x0002: SAMPLE_INTERPLOATE (插值)
dwTimeDiff	采样点和实际值之间的允许的时间差，毫秒为单位，填 0 则忽略该参数。
dwCount	返回的数据个数
ppValue	返回位号在各采样时间点上的磁盘历史数据值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。此时 dwCount 值为 0，ppValue 值为 NULL

注意

如果函数返回 S_OK，参数 ppValue 将返回位号在各采样时间点上的磁盘历史数据值。指针*ppValue 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

实时数据库端首先会根据采样间隔 dwInterval 对给定时间段进行分割，得到等间隔的时间点链表。然后读取各个时间点上的值，组成采样值数组返回。在读取每个时间点上的值时，如果指定时间点有值，将直接取该值。如果指定时间点上没有值，则利用指定的采样策略 wFlag 进行取值。若传入参数 dwTimeDiff 不为 0，那么还要计算最终取到的值的时间点与所要读取的时间点的偏差，要是偏差在 dwTimeDiff 范围内，取该值；否则忽略该值。

3.5.12 ReadTagsHisAtTime

描述: 读取一批位号在某个时刻的历史值，如果位号有内存历史值则读取内存历史值，否则读取磁

盘历史值。

```
HRESULT ReadTagsHisAtTime (  
    HANDLE handle,  
    DWORD dwCount,  
    HTAG* hTags,  
    FILETIME ftTime,  
    DWORD wFlag,  
    DWORD dwTimeInterval,  
    HRESULT** ppRes,  
    TAGVALSTATE** ppValue  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
dwCount	需要读取的位号个数
hTags	需要读取的位号句柄数组
ftTime	指定时间点
wFlag	读取策略 0x0000: SAMPLE_BEFORE(前值) 0x0001: SAMPLE_AFTER(后值) 0x0002: SAMPLE_INTERPLOATE (插值)
dwTimeInterval	采样点和实际值之间的允许的时间差，毫秒为单位，填 0 则忽略该参数。
ppRes	返回每个位号的读取结果
ppValue	返回各个位号指定时间点上的历史数据值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。 此时 ppRes 值为 NULL，ppValue 值为 NULL

注意

如果函数返回 S_OK，参数 ppValue 将返回各个实位号在指定时间点上的历史值，参数 ppRes 将返回各个位号的读取结果。指针* ppValue 和*ppRes 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中

VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

如果指定时间点 ftTime 有值，实时数据库端将直接返回该值。如果指定时间点 ftTime 上没有值，则利用指定的读取策略 wFlag 进行取值。若传入参数 dwTimeInterval 不为 0，那么还要计算最终取到的值的时间点与指定时间点 ftTime 的偏差，要是偏差在 dwTimeInterval 范围内，返回该值；否则返回指定时间点没有值的错误。

3.5.13 ReadTagHisInTime

描述：读取单个位号在某个时间段内的历史值，如果位号有内存历史值则读取内存历史，否则读取磁盘历史值。

```
HRESULT ReadTagHisInTime (  
    HANDLE handle,  
    HTAG hTag,  
    FILETIME ftBegin,  
    FILETIME ftEnd,  
    DWORD wFlag,  
    DWORD &dwCount,  
    TAGVALSTATE** ppValue  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
ftBegin	开始时间点
ftEnd	结束时间点
wFlag	读取策略 0x0000:BOUND_NONE(不含左右边界) 0x0001:BOUND_LEFT(含左边界) 0x0002:BOUND_RIGHT(含右边界) 0x0003:BOUND_BOTH(含左右边界)
dwCount	返回的数据个数
ppValue	返回位号在指定时间段的历史数据值，该结构的具体定义请参见 TAGVALSTATE

返回值

返回值	描述
S_OK	读取执行成功

返回值	描述
其他值	读取执行失败，返回值即为错误代码。 此时 dwCount 值为 0，ppValue 值为 NULL

注意

如果函数返回 S_OK，参数 ppValue 将返回位号在指定时间段的历史数据值。指针* ppValue 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.5.14 ReadTagHisSampling

描述：采样读取一个位号历史值。

```

HRESULT ReadTagHisSampling (
    HANDLE handle,
    HTAG hTag,
    FILETIME ftBegin,
    FILETIME ftEnd,
    DWORD dwInterval,
    DWORD wFlag,
    DWORD dwTimeDiff,
    DWORD &dwCount,
    TAGVALSTATE** ppValue
);

```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
hTag	位号句柄
ftBegin	开始时间点
ftEnd	结束时间点
dwInterval	采样间隔
wFlag	读取策略 0x0000: SAMPLE_BEFORE(前值) 0x0001: SAMPLE_AFTER(后值) 0x0002: SAMPLE_INTERPLOATE (插值)
dwTimeDiff	采样点和实际值之间的允许的时间差，毫秒为单位，填 0 则忽略该参数。
dwCount	返回的数据个数

ppValue	返回位号在各采样时间点上的历史数据值，该结构的具体定义请参见 TAGVALSTATE
---------	--

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。此时 dwCount 值为 0，ppValue 值为 NULL

注意

如果函数返回 S_OK，参数 ppValue 将返回位号在各采样时间点上的历史数据值。指针* ppValue 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 TAGVALSTATE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

实时数据库端首先会根据采样间隔 dwInterval 对给定时间段进行分割，得到等间隔的时间点链表。然后读取各个时间点上的值，组成采样值数组返回。在读取每个时间点上的值时，如果指定时间点有值，将直接取该值。如果指定时间点上没有值，则利用指定的采样策略 wFlag 进行取值。若传入参数 dwTimeDiff 不为 0，那么还要计算最终取到的值的时间点与所要读取的时间点的偏差，要是偏差在 dwTimeDiff 范围内，取该值；否则忽略该值。

3.5.15 ReadMaxDiskHisInTime

描述：读取单个位号在某个时间段内的磁盘历史值的最大值。

```

HRESULT ReadMaxDiskHis\InTime (
HANDLE handle,
HTAG hTag,
FILETIME ftBegin,
FILETIME ftEnd,
DWORD wFlag,
VARIANT* pMaxValue
);

```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值
hTag	位号句柄
ftBegin	开始时间点
ftEnd	结束时间点

wFlag	读取策略 0x0000:SAMPLE_BEFORE(前值) 0x0001:SAMPLE_AFTER(后值) 0x0002:SAMPLE_INTERPLOATE (插值)
pMaxValue	返回位号在时间段内的磁盘历史值的最大值

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。

注意

如果函数返 S_OK，参数 pMaxValue 将返回位号在时间段内的磁盘历史值的最大值。指针* pMaxValue 的内存由 DLL 分配。其中 VARIANT 类型成员调用 VariantClear 函数释放。

3.5.16 ReadMinDiskHisInTime

描述：读取单个位号在某个时间段内的磁盘历史值的最小值。

```

HRESULT ReadMinDiskHisInTime (
HANDLE handle,
HTAG hTag,
FILETIME ftBegin,
FILETIME ftEnd,
DWORD wFlag,
VARIANT* pMinValue
);

```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值
hTag	位号句柄
ftBegin	开始时间点
ftEnd	结束时间点
wFlag	读取策略 0x0000:SAMPLE_BEFORE(前值) 0x0001:SAMPLE_AFTER(后值) 0x0002:SAMPLE_INTERPLOATE (插值)
pMinValue	返回位号在时间段内的磁盘历史值的最小值

返回值

返回值	描述
-----	----

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。

注意

如果函数返回 S_OK，参数 pMinValue 将返回位号在时间段内的磁盘历史值的最小值。指针* pMinValue 的内存由 DLL 分配。其中 VARIANT 类型成员调用 VariantClear 函数释放。

3.5.17 ReadSumDiskHisInTime

描述：读取单个位号在某个时间段内的磁盘历史值的和值。

```

HRESULT ReadSumDiskHisInTime (
HANDLE handle,
HTAG hTag,
FILETIME ftBegin,
FILETIME ftEnd,
DWORD wFlag,
VARIANT* pSumValue
);

```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值
hTag	位号句柄
ftBegin	开始时间点
ftEnd	结束时间点
wFlag	读取策略 0x0000:SAMPLE_BEFORE(前值) 0x0001:SAMPLE_AFTER(后值) 0x0002:SAMPLE_INTERPLOATE (插值)
pSumValue	返回位号在时间段内的磁盘历史值的和值

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。

注意

如果函数返回 S_OK，参数 pSumValue 将返回位号在时间段内的磁盘历史值的和值。指针* pSumValue 的内存由 DLL 分配。其中 VARIANT 类型成员调用 VariantClear 函数释放。

3.5.18 ReadAvgDiskHisInTime

描述: 读取单个位号在某个时间段内的磁盘历史值的平均值。

```
HRESULT ReadAvgDiskHisInTime (
HANDLE handle,
HTAG hTag,
FILETIME ftBegin,
FILETIME ftEnd,
DWORD wFlag,
VARIANT* pAvgValue
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值
hTag	位号句柄
ftBegin	开始时间点
ftEnd	结束时间点
wFlag	读取策略 0x0000: SAMPLE_BEFORE(前值) 0x0001: SAMPLE_AFTER(后值) 0x0002: SAMPLE_INTERPLOATE (插值)
pAvgValue	返回位号在时间段内的磁盘历史值的平均值

返回值

返回值	描述
S_OK	读取执行成功
其他值	读取执行失败，返回值即为错误代码。

注意

如果函数返回 S_OK，参数 pAvgValue 将返回位号在时间段内的磁盘历史值的平均值。指针* pAvgValue 的内存由 DLL 分配。其中 VARIANT 类型成员调用 VariantClear 函数释放。

3.6 区域模块

3.6.1 AddRegion

描述: 增加一个区域。

```
HRESULT AddRegion( HANDLE handle, REGION_DEF*
pRegionDef );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
pRegionDef	接口属性结构。该结构的具体定义请参见 REGION_DEF

返回值

返回值	描述
S_OK	添加区域成功
其他值	添加区域失败，返回值即为错误代码。

3.6.2 DelRegion

描述：删除一个区域。

```
HRESULT DelRegion( HANDLE handle, LPWSTR  
wszRegionPath );
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
wszRegionPath	要删除的区域路径

返回值

返回值	描述
S_OK	删除区域成功
其他值	删除区域失败，返回值即为错误代码。

3.6.3 GetRegionInfoByPath

描述：根据一组区域路径获取区域组态信息。

```
HRESULT GetRegionInfoByPath(  
HANDLE handle,  
DWORD dwCount,  
LPWSTR* pszPath,  
HRESULT** ppRes,  
REGION_ATTRIBUTE** ppRegion  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。

dwCount	区域路径个数
pszPath	区域路径数组
ppRes	用于返回各区域获取结果,大小等于 dwCount,若获取结果为 S_OK,表明 pszPath 数组对应位置上的区域其属性获取成功, ppRegion 数组对应位置上即为该区域的属性
ppRegion	用于返回的获取到的区域属性列表,该结构的具体定义请参见 REGION_ATTRIBUTE

返回值

返回值	描述
S_OK	获取操作执行成功, ppRes 参数返回各区域的获取结果, 仅当获取结果为 S_OK 时, 表示获取该区域属性成功。
其他值	获取操作执行失败, 返回值即为错误代码。此时*ppRes 和*ppRegion 的值为 NULL

注意

如果函数返回 S_OK, 参数 ppRes 返回各区域获取结果, 参数 ppRegion 返回区域属性列表。指针*ppRes 和* ppRegion 的内存由 DLL 分配, 用户使用完后必须调用 CoTaskMemFree 进行释放。同时, 用户还应该释放 REGION_ATTRIBUTE 结构内部成员的内存, 其中 VARIANT 类型成员调用 VariantClear 函数释放, 字符串类型成员调用 CoTaskMemFree 函数释放。

3.6.4 SetRegionInfo

描述: 设置一组区域组态信息。

```

HRESULT SetRegionInfo(
    HANDLE handle,
    DWORD dwCount,
    REGION_ATTRIBUTE* pRegions,
    HRESULT** ppRes
);

```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
dwCount	需要设置的区域个数
pRegions	需要设置的区域属性列表, 该结构的具体定义请参见 REGION_ATTRIBUTE

ppRes	用于返回各区域设置结果,大小等于 dwCount,若设置结果为 S_OK,表明 pRegions 数组对应位置上的区域其属性设置成功
-------	--

返回值

返回值	描述
S_OK	设置操作执行成功, ppRes 参数返回各区域的设置结果, 仅当获取结果为 S_OK 时, 表示设置该区域属性成功。
其他值	设置操作执行失败, 返回值即为错误代码。此时*ppRes 值为 NULL

注意

如果函数返回 S_OK, 参数 ppRes 返回各区域设置结果。指针*ppRes 的内存由 DLL 分配, 用户使用完后必须调用 CoTaskMemFree 进行释放。同时, 用户还应该释放 REGION_ATTRIBUTE 结构内部成员的内存, 其中 VARIANT 类型成员调用 VariantClear 函数释放, 字符串类型成员调用 CoTaskMemFree 函数释放。

3.6.5 MoveRegion

描述: 移动一个区域, 该方法兼做重命名。

```
HRESULT MoveRegion(  
    HANDLE handle,  
    LPWSTR wszRegionPath,  
    LPWSTR lpDesPath  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
wszRegionPath	要移动的区域的路径
lpDesPath	区域移动后的路径。当区域移动后的路径中的区域名称与原名称不同, 则视作区域需要重命名。

返回值

返回值	描述
S_OK	移动区域成功
其他值	移动区域失败, 返回值即为错误代码。

3.6.6 EnumChildRegion

描述：枚举指定区域下的子区域。

```
HRESULT EnumChildRegion(  
    HANDLE handle,  
    LPWSTR wszRegionPath,  
    DWORD &dwCount,  
    REGION_ATTRIBUTE** ppRegion,  
    WORD wdRecursive  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
wszRegionPath	要枚举的区域的完整路径
dwCount	用于返回子区域个数
ppRegion	返回子区域定义结构数组，该结构的具体定义请参见 REGION_ATTRIBUTE
wdRecursive	是否递归标志。 该参数指定为 1，表明递归枚举指定区域下的所有层子区域。 该参数指定为 0，表明只需枚举指定区域下的第一层子区域。

返回值

返回值	描述
S_OK	枚举成功
其他值	枚举失败，返回值即为错误代码。此时 dwCount 值为 0，*ppRegion 值为 NULL

注意

如果函数返回 S_OK，参数 ppRegion 返回枚举出的子区域列表。指针*ppRegion 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 REGION_ATTRIBUTE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.6.7 EnumTagsInRegion

描述：枚举一个指定区域下的位号。

```

HRESULT EnumTagsInRegion(
    HANDLE handle,
    LPWSTR wszRegionPath,
    DWORD& dwRealCount,
    REALTAGDEF** ppReal,
    DWORD& dwVirCount,
    VIRTUALTAGDEF** ppVirtual,
    WORD wdRecursive
);

```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
wszRegionPath	要举的区域的路径
dwRealCount	用于返回实位号的个数
ppReal	用于返回实位号结构数组，该结构的具体定义请参见 REALTAGDEF
dwVirCount	用于返回虚位号个数
ppVirtual	用于返回虚位号结构数组，该结构的具体定义请参见 VIRTUALTAGDEF
wdRecursive	是否递归标志。 该参数指定为 1，表明递归枚举指定区域下的位号，以及该区域下所有层子区域的位号。 该参数指定为 0，表明只需枚举指定区域下的位号。

返回值

返回值	描述
S_OK	枚举成功
其他值	枚举失败，返回值即为错误代码。此时实虚位号个数为 0，ppReal 和 ppVirtual 值为 NULL

注意

如果函数返回 S_OK，参数 ppReal 和 ppVirtual 分别返回枚举出的实虚位号列表。指针*ppReal 和*ppVirtual 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 REALTAGDEF 和 VIRTUALTAGDEF 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

3.6.8 Snapshot

描述：以一组对象类型作为过滤器，获取以本区域为根节点的一棵区域树。

```
HRESULT _stdcall Snapshot(  
    HANDLE handle,  
    LPWSTR wszRegionPath,  
    DWORD dwTypeCount,  
    HELEMENTTYPE* pElementType,  
    REGION_TREE** ppRegionTree  
);
```

参数说明

参数	描述
handle	连接句柄。Connect 函数连接成功后的返回值。
wszRegionPath	要枚举区域的路径
dwTypeCount	对象类型个数，仅有位号和区域两个对象
pElementType	所要进行过滤的对象类型的数组。数组元素值可以包含{2,3}，其中， 2 表示枚举出该区域下的子区域 3 表示枚举出该区域下的子位号 由于位号不可能单独被枚举，枚举子位号时必须同时也枚举出子区域。
ppRegionTree	用于返回对象树信息结构，该结构的具体定义请参见 REGION_TREE

返回值

返回值	描述
S_OK	枚举成功
其他值	枚举失败，返回值即为错误代码。此时 ppRegionTree 值为 NULL

注意

如果函数返回 S_OK，参数 ppRegionTree 返回对象树信息结构。指针* ppRegionTree 的内存由 DLL 分配，用户使用完后必须调用 CoTaskMemFree 进行释放。同时，用户还应该释放 REGION_TREE 结构内部成员的内存，其中 VARIANT 类型成员调用 VariantClear 函数释放，字符串类型成员调用 CoTaskMemFree 函数释放。

4 使用指南

4.1 数据结构

4.1.1 BASETAGDEF

位号的公共属性。

```
typedef struct
{
    LPWSTR      szTagName;
    LPWSTR      szTagDescr;
    LPWSTR      szPath;
    LPWSTR      szTagUnit;
    HTAG        hTag;
    VARTYPE      vtDataType;
    WORD         bReadOnly;
    WORD         bStore;
    DWORD        dwPoolLenth;
    double       dbHihi_limit;
    double       dbHi_limit;
    double       dbLolo_limit;
    double       dbLo_limit;
    WORD         bAlarmFlag;
    WORD         bArchive;
    WORD         bCompress;
    WORD         wCompMax;
    WORD         wCompMin;
    double       dbCompDev;
    DWORD        dwBufferCount;
    WORD         wFloatPrecision;
} BASETAGDEF;
```

成员变量:

成员名称	含义
szTagName	位号名
szTagDescr	位号描述，可以为空

szPath	位号在区域树中的路径
szTagUnit	位号的工程单位，可以为空
hTag	位号的客户端句柄
vtDataType	位号的请求数据类型
bReadOnly	位号只读标志
fEngHigh	位号的工程上限值
fEngLow	位号的工程下限值
bStore	是否保存内存历史数据
dwPoolLenth	内存历史数据的长度，最大值 65535
dbHihi_limit	报警高高限
dbHi_limit	报警高限
dbLolo_limit	报警低低限
dbLo_limit	报警低限
bAlarmFlag	是否启用报警
bArchive	是否保存磁盘历史数据
bCompress	是否对于磁盘历史数据进行压缩
wCompMax	最大压缩间隔（秒），如果超过该间隔则无论数据是否有显著变化都会保存最后一个数值
wCompMin	最小压缩间隔（秒），在该范围内无论数据是否有显著变化均不保存，主要目的是排除信号干扰
dbCompDev	压缩偏移量
dwBufferCount	磁盘历史位号缓存个数
wFloatPrecision	保存磁盘历史数据的小数点位数，仅对于浮点类型（单精度和双精度）类型有效

说明：

在增加位号时，用户不需要指定位号的客户端句柄 hTag，该属性在将位号增加到实时数据库时自动生成并赋值到用户传入的位号结构 hTag 属性中。

4.1.2 REALTAGDEF

实位号属性。

```
typedef struct
{
    BASETAGDEF    baseDef;
    LPWSTR        szTransferName;
    LPWSTR        szItemID;
    WORD          bScan;
    DWORD         dwScanSec;
    DWORD         dwInterfaceFlag;
} REALTAGDEF;
```

成员变量:

成员名称	含义
baseDef	公共部分
szTransferName	位号所在接口软件的名称
szItemID	位号在底层的名称
bScan	接口层是否扫描， 1: 扫描
dwScanSec	接口层扫描周期
dwInterfaceFlag	接口层标志位

说明:

该数据结构在增加/枚举实位号，设置/获取实位号属性时使用。

4.1.3 VIRTUALTAGDEF

虚位号属性。

```
typedef struct
{
    BASETAGDEF    baseDef;
    DWORD         dwTriggerFlag;
    DWORD         dwInterval;
    LPWSTR        szTimerScript;
    LPWSTR        szReadScript;
    LPWSTR        szWriteScript;
} VIRTUALTAGDEF;
```

成员变量:

成员名称	含义
baseDef	公共部分
dwTriggerFlag	虚位号的触发条件 TRIGGER_NONE : 0x0 (无触发) TRIGGER_TIMER : 0x1 (定时触发) TRIGGER_READ : 0x2 (读触发) TRIGGER_WRITE : 0x4 (写触发)
dwInterval	如果是定时触发, 则该参数代表触发周期
szTimerScript	虚位号对应的定时执行脚本名称
szReadScript	虚位号对应的读取执行脚本名称
szWriteScript	虚位号对应的写入执行脚本名称

说明:

该数据结构在增加/枚举虚位号, 设置/获取虚位号属性时使用。

4.1.4 REGION_DEF

区域定义。

```
typedef struct tagREGION_DEF
{
    LPWSTR    wszParentPath;
    LPWSTR    wszName;
    LPWSTR    wszDesc;
    VARIANT   vCustomData;
} REGION_DEF;
```

成员变量:

成员名称	含义
wszParentPath	父区域对象的路径
wszName	区域对象名
wszDesc	区域对象描述, 可以为空
vCustomData	区域对象的自定义信息, 若无自定义信息, 指定该属性为 VT_EMPTY

说明:

该数据结构在增加一个区域时使用。

4.1.5 REGION_ATTRIBUTE

区域属性。

```
typedef struct tagREGION_ATTRIBUTE
{
    LPWSTR    wszPath;
    LPWSTR    wszDesc;
    VARIANT   vCustomData;
} REGION_ATTRIBUTE;
```

成员变量:

成员名称	含义
wszPath	对象路径, 只读
wszDesc	对象描述, 可以为空
vCustomData	对象的自定义信息, 若无自定义信息, 指定该属性为 VT_EMPTY

说明:

该数据结构在获取/设置区域属性时使用。

4.1.6 REGION_TREE

区域树信息。

```
typedef struct tagREGION_TREE
{
    REGION_ATTRIBUTE   RegionAttr;
    DWORD              dwRealTagCount;
    REALTAGDEF*        pRealTags;
    DWORD              dwVirtualTagCount;
    VIRTUALTAGDEF*      pVirtualTags;
    DWORD              dwChildRegionCount;
    REGION_TREE*        pChildRegions;
} REGION_TREE;
```

成员变量:

成员名称	含义
RegionAttr	区域属性
dwRealTagCount	该区域第一层下实位号的个数
pRealTags	该区域第一层下实位号属性数组
dwVirtualTagCount	该区域第一层下虚位号的个数

pVirtualTags	该区域第一层下虚位号属性数组
dwChildRegionCount	该区域第一层下的子区域个数
pChildRegions	该区域第一层下的子区域信息

说明:

该数据结构在调用 Snapshot 时使用, 返回指定区域下的对象树。

4.1.7 TAGVALUESTATE

位号数据值定义。

```
typedef struct
{
    HTAG      hTag;
    FILETIME  ftTimeStamp;
    WORD      wQuality;
    WORD      wAlarmState;
    VARIANT   vEng_value;
}TAGVALSTATE;
```

成员变量:

成员名称	含义
hTag	位号全局句柄
ftTimeStamp	时间戳
wQuality	质量码
wAlarmState	报警状态 ALARM_NONE: 0x00 无报警 ALARM_LO: 0x01 低限报警 ALARM_LOLO: 0x02 低低限报警 ALARM_HI: 0x03 高限报警 ALARM_HIHI: 0x04 高高限报警
vEng_value	数据值

说明:

该数据结构在获取各类位号数据值, 写入实时数据值时使用。

4.1.8 USER_DEF

用户定义。

```
typedef struct tagUSER_DEF
{
    LPWSTR wszUserName;
    LPWSTR wszUserPassword;
    LPWSTR wszUserDesc;
    VARIANT vCustomData;
} USER_DEF;
```

成员变量:

成员名称	含义
wszUserName	用户名，只读
wszUserPassword	用户密码
wszUserDesc	用户描述，可以为空
vCustomData	用户的自定义数据，若无自定义数据，指定该属性为 VT_EMPTY

说明:

该数据结构在增加一个用户、设置一个用户属性时使用。

4.1.9 USERGROUP_DEF

用户组属性。

```
typedef struct tagUSERGROUP_DEF
{
    LPWSTR wszUserGroupName;
    LPWSTR wszUserGroupDesc;
    VARIANT vCustomData;
} USERGROUP_DEF;
```

成员变量:

成员名称	含义
wszUserGroupName	用户组名称，只读
wszUserGroupDesc	用户组描述
vCustomData	用户组的自定义数据，若无自定义数据，指定该属性为 VT_EMPTY

说明:

该数据结构在增加用户组，设置/获取用户组属性时使用。

4.1.10 ACEHEADER

访问权限条目。

```
typedef struct tagACEHEADER
{
    LPWSTR wszElementPath;
    LPWSTR wszRightName;
    USHORT bGrant;
}ACEHEADER;
```

成员变量:

成员名称	含义
wszElementPath	对象路径
wszRightName	权限的名称
bGrant	授予/拒绝的权限

说明:

该数据结构在枚举用户/用户组权限时使用。

4.1.11 RIGHT_DEF

权限定义。

```
typedef DWORD HRIGHTTYPE;
typedef struct tagRIGHTDEF
{
    LPWSTR wszRightName;
    LPWSTR wszRightAlias;
    LPWSTR wszRightDescr;
    HRIGHTTYPE hRight;
}RIGHT_DEF;
```

成员变量:

成员名称	含义
wszRightName	权限的名称
wszRightAlias	权限的别名
wszRightDescr	权限的描述，该属性可以为空
hRight	权限的句柄

说明: 无。

4.1.12 TYPE_INFO

获取对象类型信息，以及隶属于该对象类型的权限列表。

```
typedef DWORD HELEMENTTYPE;
typedef struct tagTYPEINFO
{
    LPWSTR      wszTypeName;
    LPWSTR      wszTypeAlias;
    LPWSTR      wszTypeDescr;
    HELEMENTTYPE hElementType;
    DWORD       dwRightCount;
    RIGHT_DEF*  pRightDef;
}TYPE_INFO;
```

成员变量：

成员名称	含义
wszTypeName	对象类型的名称，可以是 Global, Region, Tag
wszTypeAlias	对象类型的别名
wszTypeDescr	对象类型的描述，该属性可以为空
hElementType	对象类型的句柄
dwRightCount	隶属于该对象的权限个数
pRightDef	隶属于该对象的权限数组

说明：

该数据结构在枚举所有类型以及隶属于类型的权限时使用。

4.1.13 TYPE_DEF

获取对象类型信息。

```
typedef DWORD HELEMENTTYPE;
typedef struct tagTYPEDEF
{
    LPWSTR      wszTypeName;
    LPWSTR      wszTypeAlias;
    LPWSTR      wszTypeDescr;
    HELEMENTTYPE hElementType;
}TYPE_DEF;
```

成员变量:

成员名称	含义
wszTypeName	对象类型的名称, 可以是 Global, Region, Tag
wszTypeAlias	对象类型的别名
wszTypeDescr	对象类型的描述, 该属性可以为空
hElementType	对象类型的句柄

说明:

该数据结构在枚举所有类型时使用。

4.1.14 TRANSFER_DEF

接口的基本信息定义。

```
typedef struct
{
    DWORD hTransfer;
    LPWSTR szTransferName;
    LPWSTR szTransferProgID;
    LPWSTR szTransferDescription;
}TRANSFER_DEF;
```

成员变量:

成员名称	含义
hTransfer	接口软件 ID

szTransferName	接口软件名称
szTransferProgID	接口软件的 ProgID
szTransferDescription	接口软件描述

说明：

该数据结构在增加一个接口，获取/设置接口属性时使用。增加一个接口时，用户可以不指定 hTransfer 属性。因为该属性是将接口加入到实时数据库时自动分配。

4.1.15 TRANSFER_INFO

接口的注册信息。

```
typedef struct tagTransferRegInfo
{
    LPWSTR wszTransferProgID;
    LPWSTR wszTransferDescription;
} TRANSFER_INFO;
```

成员变量：

成员名称	含义
wszTransferProgID	接口 ProgID
wszTransferDescription	接口描述

说明：

该数据结构在枚举已注册接口时使用。

4.1.16 TRANSFER_PROPERTYDEF

接口自定义属性定义。

```
typedef struct
{
    LPWSTR    wszName;
    LPWSTR    wszDescription;
    BOOL      bReadOnly;
    VARTYPE    vt;
    VARIANT    varDefault;
    WORD      wLimitType;
    double     dbUpLimit;
    double     dbLowLimit;
    LPWSTR    wszRegularExpression;
} TRANSFER_PROPERTYDEF;
```

成员变量:

成员名称	含义
wszName	属性的名称
wszDescription	属性的描述，可以为空
bReadOnly	是否只读。如果是只读属性，组态的时候不需要用户不调整，但是可以通过该属性反映服务器内部状态
vt	属性的数据类型，如果为 VT_EMPTY 说明该位号无缺省值
varDefault	缺省值，如果没有缺省值，该属性无效
wLimitType	标志采用什么方式对属性进行校验。 PROP_HI 0x01 校验上限 PROP_LO 0x02 校验下限 PROP_HILO 0x03 校验上下限 PROP_REG 0x04 通过正则表达式校验
dbUpLimit	属性上限,仅对数值类型有效
dbLowLimit	属性下限,仅对数值类型有效
wszRegularExpression	正则表达式，如果不是采用正则表达式进行校验，该字符串可以为 NULL

说明:

该数据结构在枚举接口自定义属性的定义时使用。接口的自定义属性由接口本身的类型决定，用户无法修改。

4.1.17 TRANSFER_PROPERTYVALUE

接口属性值。

```
typedef struct
{
    LPWSTR wszName; //属性的名称
    VARIANT varPropty; //属性的值
} TRANSFER_PROPERTYVALUE;
```

成员变量:

成员名称	含义
wszName	属性的名称
vtPropertyVal	属性的值

说明:

该数据结构在枚举接口自定义属性值时使用。

4.2 数据读取策略详解

4.2.1 时间段读取策略

时间段读取策略包括:

```
0x0000:BOUND_NONE(不含左右边界)
0x0001:BOUND_LEFT (含左边界)
0x0002:BOUND_RIGHT(含右边界)
0x0003:BOUND_BOTH (含左右边界)
```

该策略在读取位号一段时间的历史数据时使用。以下两张表格分别罗列了采用策略与实际采样时间段的关系。

表 4-1 实际采样起始时间与采用策略关系表

	给定起始时间 点上		实际采样起 始时间
	有 值	无 值	
Bound_None			st+1
Bound_Left/Bound_Bot h		√	st+1
	√		st

表 4-2 实际采样结束时间与采用策略关系表

	给定结束时间 点上	实际采样结 束时间
--	--------------	--------------

	有 值	无 值	
Bound_None			et-1
Bound_Right/Bound_Both		√	et-1
	√		et

注：

st：指定的起始时间点

st+1：大于指定起始时间点上第一个数据值的时间戳。

st-1：小于指定起始时间点上第一个数据值的时间戳。

et：指定的结束时间点

et+1：大于指定结束时间点上第一个数据值的时间戳。

et-1：小于指定结束时间点上第一个数据值的时间戳。

因此：

如果采用左边界 Bound_Left 策略，实际采样时间段的起始时间大于等于指定起始时间。

如果采用右边界 Bound_Right 策略，实际采样时间段的结束时间小于等于指定结束时间。

如果采用左右边界 Bound_Both 策略，则同时满足以上两条。

如果采用不包含边界 Bound_None 策略，实际采样时间段的起始时间大于指定起始时间，实际采样时间段的结束时间小于指定结束时间

接下来通过两个示例来说明各个策略的采样方式。

假设当前实时数据库在01:00到01:07时间段内每隔一分钟存在一个值。现在需要采样01:02:20到01:04:21时间段内的值，可见在起始和结束时间点上都没有值。不同的采样策略对应的实际采样时间段见以下4张图。



图 4-1 不包含边界策略



图 4-2 包含左边界策略



图 4-3 包含右边界策略



图 4-4 包含左右边界策略

假设当前实时数据库在 01:00 到 01:07 时间段内每隔一分钟存在一个值。现在需要采样 01:02:00 到 01:05:00 时间段内的值，可见在起始和结束时间点上都存在值。不同的采样策略对应的实际采样时间段见以下 4 张图。



图 4-5 不包含边界策略



图 4-6 包含左边界策略

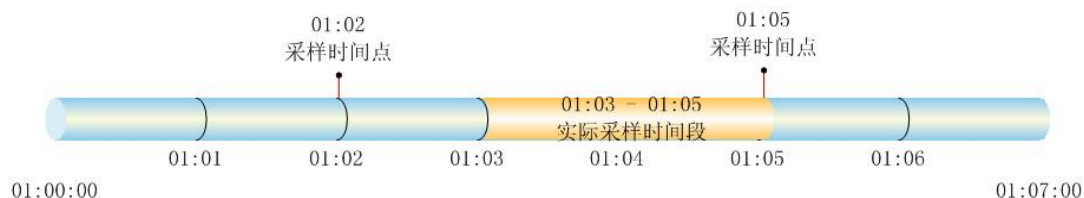


图 4-7 包含右边界策略



图 4-8 包含左右边界策略

4.2.2 时间点读取策略

时间点读取策略包括：

0x0000: SAMPLE_BEFORE(前值)
 0x0001: SAMPLE_AFTER(后值)
 0x0002: SAMPLE_INTERPLOATE (插值)

接下来通过一个示例来说明各个策略的采样方式。

假设当前实时数据库在01:00到01:07时间段内每隔一分钟存在一个值。如图4-9所示。

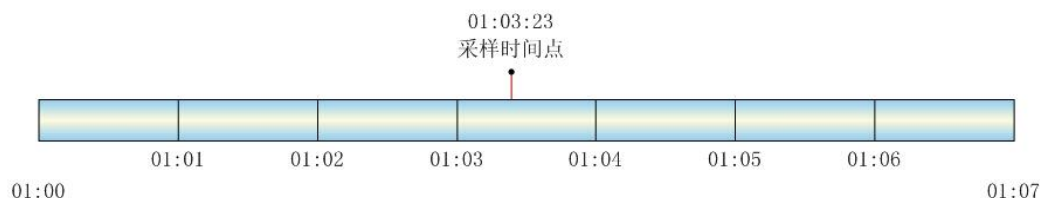


图4-9 历史值时间点

当指定采样时间为01:03:23时，由于该时间点上没有值，所以将根据采样策略取值。如果采样策略为Sample_Before，取01:03时间点上的值；如果采样策略为Sample_After，取01:04时间点上的值；如果采样策略为Sample_Interpolate则会通过01:03和01:04时间点上的值进行插值计算后取得，具体计算方法如下。

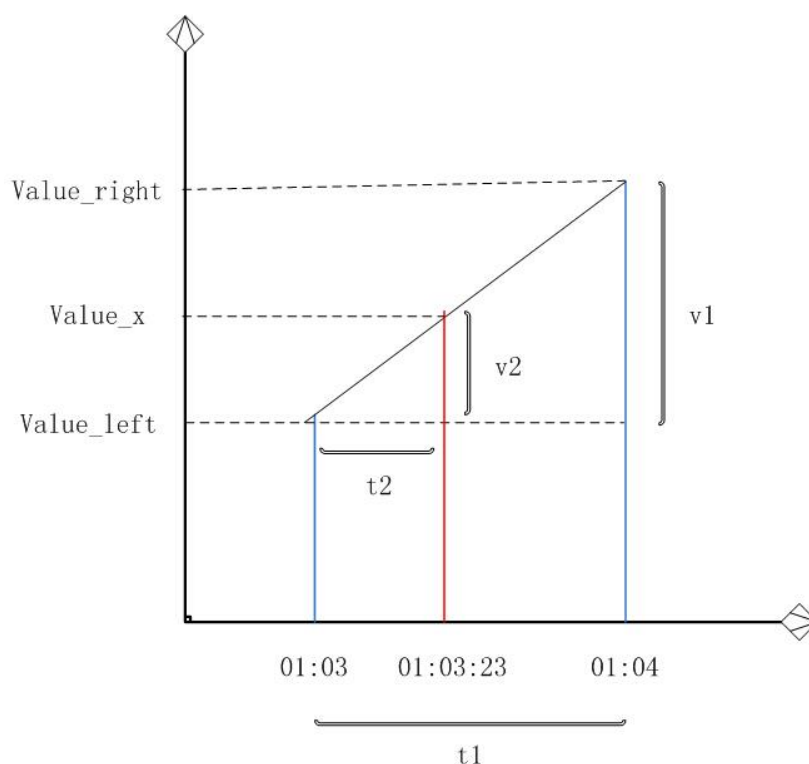


图4-10 插值计算坐标图

假设：

时间点01:03上的值为Value_left;

时间点01:04上的值为Value_right;

时间点01:03:23上的值为Value_x;

根据图4-10可得：

$v1 = \text{Value_right} - \text{Value_left};$

t1为01:04 与 01:03的时间差

t2为01:03:23与01:03的时间差;

因此可得 $v2 = (t2 * v1) / t1$;

由于 $Value_x = Value_left + v2$;

可得 $Value_x = Value_left + (t2 * (Value_right - Value_left)) / t1$.

4.3 注意事项

4.3.1 内存管理

API 模块 DLL 接口中对内存的管理有统一的原则：用户分配的由用户释放，DLL 分配的也由用户释放。

当接口中有参数是数据结构指针需要存返回的数据内容时，当接口调用成功，即返回值为 S_OK，DLL 会为此类结构指针分配内存并存入返回数据，分配内存函数使用 ::CoTaskMemAlloc()，调用者必须使用::CoTaskMemFree()将相应结构指针参数的内存释放，否则会导致内存泄露；若接口调用失败，即返回值不是 S_OK，此时不用考虑参数的内存释放，因为 DLL 没有分配内存。

4.3.2 有关 FILETIME 时间类型

在本接口中需传入的 FILETIME 时间参数均为 UTC 时间，若用户指定时间为本地时间，则传入参数前需做转换，如下：

```
//假设获取时间点 2009/3/1 3:14:25
//使用 COleDateTime 需加头文件:
// #include <ATLComTime.h>
COleDateTime t(2009, 3, 1, 3, 14, 25);
FILETIME Localft;
SYSTEMTIME sy;
t.GetAsSystemTime(sy);
//SystemTime转化为FileTime
SystemTimeToFileTime( &sy, &Localft);
//将本地时间转化为UTC时间
FILETIME ftTime;
LocalFileTimeToFileTime(&Localft, & ftTime);
//ftTime即为2009/3/1 3:14:25的UTC时间
```

4.3.3 有关错误信息

对于调用接口的返回值 SUCCEED () 失败的情况，根据错误值可以用 GetLastError 接口获取具体错误信息。错误信息分两部分，一部分是由服务器指定，一部分是本地 DLL 指定。本地错误大致为打包解包数据出错、内存分配失败、连接超时等等此类异常，错误值范围在 (E0090001 ~ E0090601) 之间。若为本地错误，调用 GetLastError 时的第一个参数 handle 连接句柄可以忽略，错误信息直接在本地找，不需要连接到实时数据库服务器，若非本地错误 handle 连接句柄必须填有效句柄，此时需要连接到服务器去获取。

4.3.4 回调函数

回调函数定义:

```
enum euUpdateType
{
    ELEMENT_TREE = 1, //对象树
    USER, //用户/用户组
    TRANSFER //接口
};
//回调函数
```

```
typedef void (*CallbackMessageFunc) (HANDLE handle, euUpdateType Type, FILETIME
ftUpdateTime);
```

handle: 连接句柄

euUpdateType: 指定更新的内容

ftUpdateTime: 服务器更新组态内容的时间

4.4 应用实例

4.4.1 动态加载 DLL

动态加载 API 模块 DLL 需要: iSYSNetAPI.dll 和 接口声明头文件 DllFunDef.h, 以及数据类型定义文件 DataStructure.h, 下面对连接服务器和用户登录两个接口的调用以动态加载 DLL 方式实现:

1. 工程中加入 DllFunDef.h 和 DataStructure.h 文件
2. 分别定义连接和登录接口的函数指针:

//登录

```
typedef HRESULT (_stdcall *faLogIn) ( HANDLE handle, LPWSTR lpUserName, LPWSTR
lpPassword );
```

//连接服务器

```
typedef HANDLE (_stdcall *faConnect)(
LPWSTR lpServerIPAddr, USHORT usServerPort );
```

将以上定义加入到调用接口 CPP 的对应头文件中 (若没有头文件, 直接加到 CPP 文件的头部)

3. 在 CPP 函数中实现:

//动态加载 DLL

```
HMODULE hModule = LoadLibrary( "ISYSNetAPI.dll" );
```

/*此时的 dll 放在当前工作目录下, 路径也可以用绝对路径, 但建议用相对路径*/

//获取 Connect 连接接口地址

```
faConnect faCon = (faConnect)GetProcAddress( hModule, "Connect");
```

//获取LogIn登录接口地址

```
faLogIn faLog = (faLogIn)GetProcAddress( hModule, "LogIn");
```

//初始化一些参数

```
char IP[] = "127.0.0.1";
```

```
char admin[] = "admin";
```

```

char pas[] = "123456";
//执行连接
HANDLE handle = faCon( CA2W( IP ), 5150 );

HRESULT hr = faLog( handle, CA2W(admin), CA2W(pas) );
/*若连接失败，handle是返回NULL的，此时可调用
GetLastException()接口来获取最后的异常错误值，再调用GetErrorMgs()获取具体的连接错误
信息*/

```

4.4.2 静态加载 DLL

静态加载 API 模块 DLL 需要：iSYSNetAPI.dll、iSYSNetAPI.lib 和接口声明头文件 DllFunDef.h，以及数据类型定义文件 DataStructure.h，下面对连接服务器和用户登录两个接口的调用以静态加载 DLL 方式实现：

1. 工程中加入 DllFunDef.h 和 DataStructure.h 文件

2. 加载 lib 库：

```
#pragma comment( lib, "ISYSNetAPI.lib" )
```

3. 在 CPP 函数中实现：

```

char IP[] = "192.168.0.154";
char admin[] = "administrator";
char pas[] = "supcon";
HANDLE handle = Connect( CA2W( IP ), 5150 );
if( NULL == handle )
{
    HRESULT hError = GetLastException ();
    LPWSTR pErrorWstr = NULL;
    HRESULT hTmp = GetErrorMgs( NULL, hError, pErrorWstr );
    if( S_OK == hTmp )
    {
        printf("hError: %S \r\n", pErrorWstr );
        ::CoTaskMemFree( pErrorWstr );
    }
    Else
    {
        printf("获取错误信息失败~ \r\n");
    }
    continue;
}

```

```
HRESULT hr = LogIn( handle, CA2W(admin),
```

```
CA2W(pas) );
```

/*静态加载不需要获取接口函数地址，可以直接调用，但要注意不能与本地CPP中的相关函数名相冲突*/

4.4.3 接口调用举例

1、枚举实位号

```

REALTAGDEF* pDefs = NULL;
DWORD dwCount = 0;
HRESULT hr = EnumRealTag( g_hConnect, dwCount, &pDefs );
if( FAILED(hr) )
    printf("EnumRealTag 失败, 错误值: 0x%08X\r\n",hr);
else
{
    for( DWORD i = 0; i < dwCount; i++ )
    {
        printf(" -hTag %u -szTagName %s -vt %hd -szItemID %s\r\n", pDefs[i].baseDef.hTag, (const char
        *) (COLE2T(pDefs[i].baseDef.szTagName)), pDefs[i].baseDef.vtDataType, (const char
        *) (COLE2T(pDefs[i].szItemID)));
        ::CoTaskMemFree( pDefs[i].baseDef.szPath );
        ::CoTaskMemFree( pDefs[i].baseDef.szTagName );
        ::CoTaskMemFree( pDefs[i].baseDef.szTagDescr );
        ::CoTaskMemFree( pDefs[i].szItemID );
        ::CoTaskMemFree( pDefs[i].szTransferName );
    }
    ::CoTaskMemFree( pDefs );
}

printf("总共获得 %u 个实位号 \r\n", dwCount);

```

2、读取一批位号在某个时刻的内存历史值

```

//设定时间点
COleDateTime ttt(2009, 3, 1, 3, 14, 25);
FILETIME Localft;
SYSTEMTIME sy;
t.GetAsSystemTime(sy);
//SystemTime 转化为 FileTime
SystemTimeToFileTime( &sy, &Localft);
//将本地时间转化为 UTC 时间
FILETIME ftTime;
LocalFileTimeToFileTime(&Localft, &ftTime);
//准备位号句柄
HTAG hTags[2];
hTags[0] = 1;
hTags[1] = 2147483651;
//调用 DLL 接口执行
HRESULT* pRes = NULL;
TAGVALSTATE* pValues = NULL;
HTAG* pTags = &(tagVec[0]);
HRESULT hr = ReadTagsMemHisAtTime( g_hConnect, iSize, pTags, ft, wOption, wInterval,

```

```

&pRes, &pValues );
    if( FAILED(hr) )
printf ("TReadTagsMemHisAtTime 失败, 错误值: 0x%08X \r\n", hr);
else
{
    for( int i = 0; i < iSize; i++){
        if( S_OK != pRes[i] )
            PRINTF("位号 %u 读取实时数据失败, 错误值: 0x%08X \r\n", tagVec[i], pRes[i] );
        else{
            //输出个位号实时值
            CComVariant comVar( pValues[i].vEng_value );
            if( FAILED(comVar.ChangeType( VT_BSTR)) ) {
                printf ("位号句柄: %u 数据值转换失败! \r\n", tagVec[i] );
            }
            else{
                CTime t( pValues[i].ftTimeStamp );
                printf ("位号句柄: %u, 时间: %d-%d-%d %d:%d:%d, 质量码: %hd, 报警状态: %hd, 位号值: %s\r\n",
                    pValues[i].hTag, t.GetYear(), t.GetMonth(), t.GetDay(), t.GetHour(),
                    t.GetMinute(), t.GetSecond(),
                    pValues[i].wQuality, pValues[i].wAlarmState, COLE2T(comVar.bstrVal) );
            }
            ::VariantClear( &pValues[i].vEng_value);
        }
    }
    ::CoTaskMemFree(pRes);
    ::CoTaskMemFree(pValues);
}

```

3、回调函数使用

//回调函数

void Callback(HANDLE handle, euUpdateType eu, FILETIME ft)

```

{
    printf("系统成功执行回调函数一次~ \r\n");
}

```

//请求组态更新提示

HRESULT TReqUpdateMessage()

```

{
    //调用DLL接口执行, g_hConnect为全局连接句柄
    HRESULT hr = ReqUpdateMessage( g_hConnect, Callback, 1 );
    if( FAILED(hr) )
        printf ("请求失败, 错误值为: 0x%08X \r\n", hr);
    else
        printf("请求成功~\r\n" );
}

```

```

    return S_OK;
}

```

/*执行 TReqUpdateMessage 函数之后，当服务器上的组态信息修改时，服务器会主动发更新提示到本地连接对象中，自动调用 Callback 函数 */

4.4.4 单线程接口调用举例

例子中，通过创建一个线程来调用 DLL 的读取一批位号在某个时刻的历史值接口，其中，位号量为 10000。

4、创建线程

//开启线程

```

DWORD dwThreadID = 0;
HANDLE hThread = (HANDLE)_beginthreadex( NULL, 0, DefaultThreadProc, this,
0, (unsigned*)&dwThreadID );
if ( NULL == hThread )
{
    printf("创建读取一批位号在某个时刻的历史值线程失败...\n");
    return;
}

```

5、工作线程回调函数

```

unsigned CALLBACK CThreadWrap::DefaultThreadProc( void * pParam )
{
    CThreadWrap * pThread = (CThreadWrap*)pParam;
    ::CoInitialize(NULL);
    try
    {
        //调用执行读取一批位号在某个时刻的历史值的方法
        HRESULT hr = pThread->DealTasks();
    }
    catch(...)
    {
    }
    ::CoUninitialize();
    return S_OK;
}

```

6、读取一批位号在某个时刻的历史值

//设定时间点

```
COleDateTime ttt(2012, 5, 8, 3, 14, 25);
```

```
FILETIME Localft;
```

```
SYSTEMTIME sy;
```

```
t.GetAsSystemTime(sy);
```

//SystemTime 转化为 FileTime

```
SystemTimeToFileTime( &sy, &Localft);
```

//将本地时间转化为 UTC 时间

```
FILETIME ftTime;
```

```
LocalFileTimeToFileTime(&Localft, &ftTime);
```

//准备位号句柄

```
DWORD dwCount = 10000;
```

```
HTAG * hTags = new(std::nothrow) HTAG[dwCount];
```

```

    if ( NULL == hTags )
    {
        return ;
    }
    CAutoVectorPtr<HTAG> auto_hTags( hTags );
    int iIndex = 0;
    for( DWORD i = 1; i <= dwCount; i++)
    {
        *(hTags+iIndex) = i;
        iIndex++;
    }

    //调用 DLL 接口执行
    HRESULT* pRes = NULL;
    TAGVALSTATE* pValues = NULL;
    HTAG* pTags = &(tagVec[0]);
    HRESULT hr = ReadTagsHisAtTime( g_hConnect, dwCount, hTags, ft, wOption, wInterval, &pRes,
    &pValues );
    if( FAILED(hr) )
        printf("TReadTagsMemHisAtTime 失败, 错误值: 0x%08X \r\n", hr);
    else
    {
        ::CoTaskMemFree(pRes);
        ::CoTaskMemFree(pValues);
    }
}

```

4.4.5 多线程接口调用举例

例子中，通过创建 5 个线程来调用 DLL 的读取一批位号在某个时刻的历史值接口。位号量为 10000，每个线程分别处理 2000 个位号，。

7、创建多线程

//创建 5 个线程

```

DWORD dwCount = 5;
for( DWORD dwIndex = 0; dwIndex < dwCount; dwIndex++ )
{
    DWORD dwThreadID = 0;
    HANDLE hThread = (HANDLE)_beginthreadex( NULL, 0, DefaultThreadProc, this,
    0, (unsigned*)&dwThreadID );
    if ( NULL == hThread )
    {
        printf("创建读取一批位号在某个时刻的历史值线程失败...\n");
        return;
    }
}
]

```

8、工作线程回调函数

```

unsigned CALLBACK CThreadWrap::DefaultThreadProc( void * pParam )
{
    CThreadWrap * pThread = (CThreadWrap*)pParam;
    ::CoInitialize(NULL);
}

```

```

try
{
    //调用执行读取一批位号在某个时刻的历史值的方法
    HRESULT hr = pThread->DealTasks();
}
catch(...)
{
}
::CoUninitialize();
return S_OK;
}

```

9、读取一批位号在某个时刻的历史值

```

//获取每个线程的位号信息
// m_pParam表示每个线程对应信息的对象，保存开始位号、读取策略等信息
//开始位号
DWORD dwBeginTag = m_pParam->dwTag;
//位号个数
DWORD dwTagNum = 2000;
//设定时间点
COleDateTime ttt(2012, 5, 8, 3, 14, 25);
FILETIME Localft;
SYSTEMTIME sy;
t.GetAsSystemTime(sy);
//SystemTime 转化为 FileTime
SystemTimeToFileTime( &sy, &Localft);
//将本地时间转化为 UTC 时间
FILETIME ftTime;
LocalFileTimeToFileTime(&Localft, &ftTime);
//准备位号句柄
HTAG * pTags = new(std::nothrow) HTAG[dwTagNum];
if ( NULL == pTags)
{
    return ;
}
CAutoVectorPtr<HTAG> auto_htags(pTags);
int iIndex = 0;
for( DWORD i = dwBeginTag; i < dwBeginTag + dwTagNum ; i++)
{
    *(pTags +iIndex) = i;
    iIndex++;
}
//调用DLL接口执行
HRESULT* pRes = NULL;
TAGVALSTATE* pValues = NULL;
HRESULT hr = ReadTagsMemHisAtTime( g_hConnect, dwTagNum, pTags, ft, wOption, wInterval,
&pRes, &pValues );
if( FAILED(hr) )
{
    printf("TReadTagsMemHisAtTime 失败, 错误值: 0x%08X \r\n", hr);
}

```



```
    }  
else  
{  
    ::CoTaskMemFree(pRes);  
    ::CoTaskMemFree(pValues);  
}
```

5 资料版本说明

表 5-1 版本升级更改一览表

资料版本号	适用软件版本	更改说明
V1.0（20200210）	ESP-iSYS 5.3	