User Manual

APAX MODBUS Library Manual

1. MODBUS common functions

1.1. MOD_Initialize

LONG ADS_API MOD_Initialize();

Purpose:

Initialize the MODBUS library resource.

Parameters:

None.

Return:

ERR_SUCCESS, initialized MODBUS library succeeded.

ERR_WSASTART_FAILED, failed to initialize resource.

1.2. MOD_Terminate

LONG ADS_API MOD_Terminate();

Purpose:

Terminate the MODBUS library and release resource.

Before calling this function, all MODBUS clients and servers have to be stopped.

Parameters:

None.

Return:

ERR_SUCCESS, terminated MODBUS library succeeded.

ERR_WSACLEAN_FAILED, failed to terminate resource.

1.3. MOD_GetVersion

LONG ADS_API MOD_GetVersion();

Purpose:

Get the version number of the MODBUS library.

Parameters:

None.

Return:

The version number. In hex decimal format, for example 0x0101, it means the version is 1.01.

- **MODBUS common functions sample code**

```
#include <stdio.h>
#include <windows.h>
#include "ADSMOD.h"

int main(int argc, char* argv[])
{
	int iSlot;
	char szIp[32] = "172.18.3.15";
	int ildleSecs = 10; // client idle timeout 10 seconds

	if (ERR_SUCCESS == MOD_Initialize())
	{
		printf("ADSMOD library version = %d\n", MOD_GetVersion());
		MOD_Terminate();
	}
	else
		printf("MOD_Initialize failed!\n");
	return 0;
}
```

2. MODBUS-TCP server functions

2.1. MOD_StartTcpServer

LONG ADS_API MOD_StartTcpServer();

Purpose:

Start the MODBUS-TCP server.

Parameters:

None.

Return:

ERR_SUCCESS, started MODBUS-TCP server succeeded.

ERR_CREATESVR_FAILED, failed to start MODBUS-TCP server. Normally, this caused by the server port (502) has been used.

2.2. MOD_StopTcpServer

LONG ADS_API MOD_StopTcpServer();

Purpose:

Stop the MODBUS-TCP server.

Parameters:

None.

Return:

ERR_SUCCESS, stopped MODBUS-TCP server succeeded.

2.3. MOD_SetTcpServerPriority

LONG ADS_API MOD_SetTcpServerPriority(int i_iPriority);

Purpose:

Set the priority of the MODBUS-TCP server running thread.

Before calling this function, the MODBUS-TCP server has to be successfully started.

Parameters:

i_iPriority = Specifies the priority value for the server thread.

This parameter can be one of the following values

THREAD_PRIORITY_TIME_CRITICAL indicates 3 points above normal priority.

THREAD_PRIORITY_HIGHEST indicates 2 points above normal priority.

THREAD_PRIORITY_ABOVE_NORMAL indicates 1 point above normal priority.

THREAD_PRIORITY_NORMAL indicates normal priority.

THREAD_PRIORITY_BELOW_NORMAL indicates 1 point below normal priority.

THREAD_PRIORITY_LOWEST indicates 2 points below normal priority.

THREAD_PRIORITY_ABOVE_IDLE indicates 3 points below normal priority.

THREAD_PRIORITY_IDLE indicates 4 points below normal priority.

Return:

ERR_SUCCESS, set MODBUS-TCP server thread priority succeeded.

ERR_SETPRIO_FAILED, failed to set the thread priority.

2.4. MOD_SetTcpServerClientIpRestrict

LONG ADS_API MOD_SetTcpServerClientIpRestrict(bool i_bRestrict);

Purpose:

Enable/disable the restriction of remote client connection. If this function sets the restriction to true, then only those clients set by MOD_SetTcpServerClientIp will be acceptable by the server.

Parameters:

i_bRestrict = The Boolean value indicates whether the IP restriction is applied.

Return:

ERR_SUCCESS, set the restriction succeeded.

2.5. MOD_GetTcpServerClientIp

LONG ADS_API MOD_GetTcpServerClientIp(int i_iIndex, char *o_szIp);

Purpose:

Get the acceptable client IP address with indicated index.

Parameters:

i_iIndex = The index of the acceptable client. This value is ranged from 0 to 7.

o_szIp = The IP address of the client in the indicated index.

Return:

ERR_SUCCESS, get the client IP succeeded.

ERR_PARAMETER_INVALID, indicates the i_iIndex is out of range.

2.6. MOD_SetTcpServerClientIp

LONG ADS_API MOD_SetTcpServerClientIp(int i_iIndex, char *i_szIp);

Purpose:

Set the acceptable client IP address with indicated index.

Parameters:

i_iIndex = The index of the acceptable client. This value is ranged from 0 to 7.

i_szIp = The IP address of the client.

Return:

ERR_SUCCESS, set the client IP succeeded.

ERR_PARAMETER_INVALID, indicates the i_iIndex is out of range or the IP is invalid.

2.7. MOD_GetTcpServerClientIdle

LONG ADS_API MOD_GetTcpServerClientIdle(int *o_iIdleSec);

Purpose:

Get the client transaction idle timeout.

Parameters:

o_iIdleSec = The transaction idle timeout.

Return:

ERR_SUCCESS, get the transaction idle timeout succeeded.

2.8. MOD_SetTcpServerClientIdle

LONG ADS_API MOD_SetTcpServerClientIdle(int i_iIdleSec);

Purpose:

Set the client transaction idle timeout. If a connected client has been idled for the setting time, the server will disconnect the client from the server.

Parameters:

i_iIdleSec = The transaction idle timeout. The value is ranged from 5 to 600 (seconds).

Return:

ERR_SUCCESS, set the transaction idle timeout succeeded.

ERR_PARAMETER_INVALID, indicates the i_iIdleSec is out of range.

● MODBUS-TCP server sample code

```
#include <stdio.h>
#include <windows.h>
#include <conio.h>
#include "ADSMOD.h"

// client connect to / disconnect from server event call back functions
void RemoteTcpClientConnectEventHandler(char *i_szIp);
void RemoteTcpClientDisconnectEventHandler(char *i_szIp);
// client write to server event call back functions
void ClientWriteCoilHandler(int iStart, int iLen);
void ClientWriteHoldRegHandler(int iStart, int iLen);

int main(int argc, char* argv[])
{
	int iSlot;
	char szIp[32] = "10.0.0.1";
	int iIdleSecs = 10; // client idle timeout 10 seconds

	if (ERR_SUCCESS == MOD_Initialize())
	{
		// ===================================================================
		// setup configuration before starting the server
		// ===================================================================
		// enable the MODBUS-TCP client IP restriction
		MOD_SetTcpServerClientIpRestrict(true);
		// assign the client IP that is legel to access the server (set to index 0)
		MOD_SetTcpServerClientIp(0, szIp);
		// set the client idle time limit, if the client become silent more than the time limit
		// the connection will be closed
		MOD_SetTcpServerClientIdle(iIdleSecs);
		// ===================================================================
		// setup callback functions
		// ===================================================================
		MOD_SetTcpServerClientConnectEventHandler(&RemoteTcpClientConnectEventHandler);
		MOD_SetTcpServerClientDisconnectEventHandler(&RemoteTcpClientDisconnectEventHandler);
		MOD_SetServerCoilChangedEventHandler(&ClientWriteCoilHandler);
		MOD_SetServerHoldRegChangedEventHandler(&ClientWriteHoldRegHandler);
```

```
        // ====================================================================
        // start the MODBUS-TCP server
        // ====================================================================
        if (ERR_SUCCESS == MOD_StartTcpServer())
        {
            printf("MODBUS-TCP server started...\n");
            while (_kbhit() == 0)
            {
                Sleep(1);
            }
            MOD_StopTcpServer();
        }
        else
            printf("MOD_StartTcpServer failed!\n");
        MOD_Terminate();
    }
    else
        printf("MOD_Initialize failed!\n");
    return 0;
}

void RemoteTcpClientConnectEventHandler(char *i_szIp)
{
    printf("Client connects from '%s'\n", i_szIp);
}

void RemoteTcpClientDisconnectEventHandler(char *i_szIp)
{
    printf("Client from '%s' disconnected\n", i_szIp);
}

void ClientWriteCoilHandler(int iStart, int iLen)
{
    int iRetLen;
    unsigned char byData[256] = {0};

    if (ERR_SUCCESS == MOD_GetServerCoil(iStart, iLen, (unsigned char*)byData, &iRetLen))
    {
```

```
            printf("Coil Start = %d; Len = %d; Data = %02X\n", iStart, iLen, byData[0]);
      }
}

void ClientWriteHoldRegHandler(int iStart, int iLen)
{
      int iRetLen;
      unsigned char byData[256] = {0};

      if (ERR_SUCCESS == MOD_GetServerHoldReg(iStart, iLen, (unsigned char*)byData, &iRetLen))
      {
            printf("Reg Start = %d; Len = %d; Data = %02X\n", iStart, iLen, byData[0]);
      }
}
```

3. MODBUS-RTU server functions

3.1. MOD_StartRtuServer

LONG ADS_API MOD_StartRtuServer(int i_iServerIndex, int i_iComIndex, unsigned char i_bySlaveAddr);

Purpose:

Start the MODBUS-RTU server.

Parameters:

i_iServerIndex = The server index. This library supports two MODBUS-RTU servers can be created at a single process. The range of this value is from 0 to 1.

i_iComIndex = The index of the COM port. The range of this value is from 1 to 255.

i_bySlaveAddr = The slave address of the server. The range of this value is from 1 to 247.

Return:

ERR_SUCCESS, started MODBUS-RTU server succeeded.

ERR_CREATESVR_FAILED, failed to start MODBUS-RTU server. Normally, this caused by the COM port has been used.

ERR_PARAMETER_INVALID, indicates some of the parameters are out of range.

3.2. MOD_StopRtuServer

LONG ADS_API MOD_StopRtuServer(int i_iServerIndex);

Purpose:

Stop the MODBUS-RTU server.

Parameters:

i_iServerIndex = The server index. The range of this value is from 0 to 1.

Return:

ERR_SUCCESS, stopped MODBUS-RTU server succeeded.

ERR_PARAMETER_INVALID, indicates the i_iServerIndex is out of range.

3.3. MOD_SetRtuServerPriority

LONG ADS_API MOD_SetRtuServerPriority(int iServerIndex, int iPriority);

Purpose:

Set the priority of the MODBUS-RTU server running thread.

Before calling this function, the MODBUS-RTU server has to be successfully started.

Parameters:

i_iServerIndex = The server index. The range of this value is from 0 to 1.

i_iPriority = Specifies the priority value for the server thread.

This parameter can be one of the following values

THREAD_PRIORITY_TIME_CRITICAL indicates 3 points above normal priority.

THREAD_PRIORITY_HIGHEST indicates 2 points above normal priority.

THREAD_PRIORITY_ABOVE_NORMAL indicates 1 point above normal priority.

THREAD_PRIORITY_NORMAL indicates normal priority.

THREAD_PRIORITY_BELOW_NORMAL indicates 1 point below normal priority.

THREAD_PRIORITY_LOWEST indicates 2 points below normal priority.

THREAD_PRIORITY_ABOVE_IDLE indicates 3 points below normal priority.

THREAD_PRIORITY_IDLE indicates 4 points below normal priority.

Return:

ERR_SUCCESS, set MODBUS-RTU server thread priority succeeded.

ERR_PARAMETER_INVALID, indicates the i_iServerIndex is out of range.

ERR_SETPRIO_FAILED, failed to set the thread priority.

3.4. MOD_SetRtuServerComm

LONG ADS_API MOD_SetRtuServerComm(int i_iServerIndex, long i_lBaudrate, int i_iDataBits, int i_iParity, int i_iStop);

Purpose:

Set the MODBUS-RTU server communication parameters.

Parameters:

i_iServerIndex = The server index. The range of this value is from 0 to 1.

i_lBaudrate = Specifies the baud rate at which the MODBUS-RTU server operates.

This parameter can be one of the following values

CBR_110

CBR_300

CBR_600

CBR_1200

CBR_2400

CBR_4800

CBR_9600

CBR_14400

CBR_19200

CBR_38400

CBR_56000

CBR_57600

CBR_115200

i_iDataBits = Specifies the number of bits in the bytes transmitted and received. The range of this value is from 5 to 8.

i_iParity = Specifies the parity scheme to be used.

The following values are possible for this member.

EVENPARITY

MARKPARITY

NOPARITY

ODDPARITY

SPACEPARITY

i_iStop = Specifies the number of stop bits to be used.

The following values are possible for this member.

ONESTOPBIT

ONE5STOPBITS

TWOSTOPBITS

Return:

ERR_SUCCESS, set the MODBUS-RTU server communication parameters succeeded.

ERR_PARAMETER_INVALID, indicates the i_iServerIndex is out of range.

3.5. MOD_SetRtuServerTimeout

LONG ADS_API MOD_SetRtuServerTimeout(int i_iServerIndex,
int i_iReadIntervalTimeout,
int i_iReadTotalTimeoutConstant,
int i_iReadTotalTimeoutMultiplier,
int i_iWriteTotalTimeoutConstant,
int i_iWriteTotalTimeoutMultiplier);

Purpose:

Set the timeout parameters for all read and write operations on the MODBUS-RTU server.

Parameters:

i_iServerIndex = The server index. The range of this value is from 0 to 1.

i_iReadIntervalTimeout = This parameter sets the Specifies the maximum acceptable time, in milliseconds, to elapse between the arrival of two characters on the communication line.

i_iReadTotalTimeoutConstant = Specifies the multiplier, in milliseconds, used to calculate the total timeout period for read operations.

i_iReadTotalTimeoutMultiplier = Specifies the constant, in milliseconds, used to calculate the total timeout period for read operations.

i_iWriteTotalTimeoutConstant = Specifies the multiplier, in milliseconds, used to calculate the total timeout period for write operations.

i_iWriteTotalTimeoutMultiplier = Specifies the constant, in milliseconds, used to calculate the total timeout period for write operations.

Return:

ERR_SUCCESS, set the timeout parameters succeeded.

ERR_PARAMETER_INVALID, indicates the i_iServerIndex is out of range.

● MODBUS-RTU server sample code

```
#include <stdio.h>
#include <windows.h>
#include <conio.h>
#include "ADSMOD.h"

void ClientWriteCoilHandler(int iStart, int iLen);
void ClientWriteHoldRegHandler(int iStart, int iLen);

int main(int argc, char* argv[])
{
	int iServerIndex = 0;
	int iComPort = 1;
	int iSlaveAddr = 1;
	int iSlot;

	if (ERR_SUCCESS == MOD_Initialize())
	{
		// ========================================================================
		// setup configuration before starting the server
		// ========================================================================
		MOD_SetRtuServerComm(iServerIndex, CBR_9600, 8, NOPARITY, ONESTOPBIT);
		MOD_SetRtuServerTimeout(iServerIndex, 50, 50, 1, 50, 1);
		// ========================================================================
		// setup callback functions
		// ========================================================================
		MOD_SetServerCoilChangedEventHandler(&ClientWriteCoilHandler);
		MOD_SetServerHoldRegChangedEventHandler(&ClientWriteHoldRegHandler);
		// ========================================================================
		// start the MODBUS-RTU server
		// ========================================================================
		if (ERR_SUCCESS == MOD_StartRtuServer(iServerIndex, iComPort, iSlaveAddr))
		{
			printf("MODBUS-RTU server started...\n");
			while (_kbhit() == 0)
			{
				Sleep(1);
			}
```

```
            MOD_StopRtuServer(0);
        }
        else
            printf("MOD_StartRtuServer failed!\n");
        MOD_Terminate();
    }
    else
        printf("MOD_Initialize failed!\n");
    return 0;
}

void ClientWriteCoilHandler(int iStart, int iLen)
{
    int iRetLen;
    unsigned char byData[256] = {0};

    if (ERR_SUCCESS == MOD_GetServerCoil(iStart, iLen, (unsigned char*)byData, &iRetLen))
    {
        printf("Coil Start = %d; Len = %d; Data = %02X\n", iStart, iLen, byData[0]);
    }
}

void ClientWriteHoldRegHandler(int iStart, int iLen)
{
    int iRetLen;
    unsigned char byData[256] = {0};

    if (ERR_SUCCESS == MOD_GetServerHoldReg(iStart, iLen, (unsigned char*)byData, &iRetLen))
    {
        printf("Reg Start = %d; Len = %d; Data = %02X\n", iStart, iLen, byData[0]);
    }
}
```

4. MODBUS server internal data functions

4.1. MOD_GetServerCoil

LONG ADS_API MOD_GetServerCoil(int i_iStart, int i_iLen, unsigned char *o_byBuf, int *o_iRetLen);

Purpose:

Get the coil status data of the MODBUS server. In a process, the library has a single copy of coil status data that can be shared among threads.

Parameters:

i_iStart = The start index of the coil status data. The range of this value is from 1 to 65535.

i_iLen = The length of the coil status data to be read. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536.

o_byBuf = The buffer pointer where to store the read coil status data.

o_iRetLen = The length of the returned coil status data in bytes.

Return:

ERR_SUCCESS, get the coil status data succeeded.

ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the o_byBuf is NULL.

4.2. MOD_GetServerInput

LONG ADS_API MOD_GetServerInput(int i_iStart, int i_iLen, unsigned char *o_byBuf, int *o_iRetLen);

Purpose:

Get the input status data of the MODBUS server. In a process, the library has a single copy of input status data that can be shared among threads.

Parameters:

i_iStart = The start index of the input status data. The range of this value is from 1 to 65535.

i_iLen = The length of the input status data to be read. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536.

o_byBuf = The buffer pointer where to store the read input status data.

o_iRetLen = The length of the returned input status data in bytes.

Return:

ERR_SUCCESS, get the input status data succeeded.

ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the o_byBuf is NULL.

4.3. MOD_GetServerInputReg

LONG ADS_API MOD_GetServerInputReg(int i_iStart, int i_iLen, unsigned char *o_byBuf, int *o_iRetLen);

Purpose:

Get the input register data of the MODBUS server. In a process, the library has a single copy of input register data that can be shared among threads.

Parameters:

i_iStart = The start index of the input register data. The range of this value is from 1 to 65535.

i_iLen = The length of the input register data to be read. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536.

o_byBuf = The buffer pointer where to store the read input register data.

o_iRetLen = The length of the returned input register data in bytes.

Return:

ERR_SUCCESS, get the input register data succeeded.

ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the o_byBuf is NULL.

4.4. MOD_GetServerHoldReg

LONG ADS_API MOD_GetServerHoldReg(int i_iStart, int i_iLen, unsigned char *o_byBuf, int *o_iRetLen);

Purpose:

Get the holding register data of the MODBUS server. In a process, the library has a single copy of holding register data that can be shared among threads.

Parameters:

i_iStart = The start index of the holding register data. The range of this value is from 1 to 65535.

i_iLen = The length of the holding register data to be read. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536.

o_byBuf = The buffer pointer where to store the read holding register data.

o_iRetLen = The length of the returned holding register data in bytes.

Return:

ERR_SUCCESS, get the holding register data succeeded.

ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the o_byBuf is NULL.

4.5. MOD_SetServerCoil

LONG ADS_API MOD_SetServerCoil(int i_iStart, int i_iLen, unsigned char *i_byBuf);

Purpose:

Set the coil status data of the MODBUS server. In a process, the library has a single copy of coil status data that can be shared among threads.

Parameters:

i_iStart = The start index of the coil status data. The range of this value is from 1 to 65535.

i_iLen = The length of the coil status data to be set. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536.

i_byBuf = The buffer pointer where to hold the coil status data to be set. For example, if the i_iLen is 40, then the length of this buffer must be 5 bytes long.

Return:

ERR_SUCCESS, get the coil status data succeeded.

ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the i_byBuf is NULL.

4.6. MOD_SetServerHoldReg

LONG ADS_API MOD_SetServerHoldReg(int i_iStart, int i_iLen, unsigned char *i_byBuf);

Purpose:

Set the holding register data of the MODBUS server. In a process, the library has a single copy of holding register data that can be shared among threads.

Parameters:

i_iStart = The start index of the holding register data. The range of this value is from 1 to 65535.

i_iLen = The length of the holding register data to be set. The range of this value is from 1 to 65536. The sum of i_iStart and i_iLen must be less than or equal to 65536.

i_byBuf = The buffer pointer where to hold the holding register data to be set. For example, if the i_iLen is 40, then the length of this buffer must be 80 bytes long (each register is 2 bytes long) The order of the bytes is as follow:

[Reg-0 High byte], [Reg-0 Low byte], [Reg-1 High byte], [Reg-1 Low byte]…

Return:

ERR_SUCCESS, get the coil status data succeeded.

ERR_PARAMETER_INVALID, indicates some of the parameters are out of range or the i_byBuf is NULL.

5. MODBUS server call back functions

5.1. OnCoilChangedEvent

typedef void (*OnCoilChangedEvent)(int i_iStart, int i_iLen);

Purpose:

The prototype of call back function when the coil changed event occurred.

Parameters:

i_iStart = The start index of the coil status data that has been changed.

i_iLen = The length of the coil status data that has been changed.

Return:

None.

5.2. OnHoldRegChangedEvent

typedef void (*OnHoldRegChangedEvent)(int i_iStart, int i_iLen);

Purpose:

The prototype of the call back function when the holding register changed event occurred.

Parameters:

i_iStart = The start index of the holding register data that has been changed.

i_iLen = The length of the holding register data that has been changed.

Return:

None.

5.3. OnRemoteTcpClientConnectEvent

typedef void (*OnRemoteTcpClientConnectEvent)(char *i_szIp);

Purpose:

The prototype of the call back function when a remote client connects to the server event occurred.

Parameters:

i_szIp = The IP address of the remote client.

Return:

None.

5.4. OnRemoteTcpClientDisconnectEvent

typedef void (*OnRemoteTcpClientDisconnectEvent)(char *i_szIp);

Purpose:

The prototype of the call back function when a remote client disconnects from the server event

occurred.

Parameters:

i_szIp = The IP address of the remote client.

Return:

None.

5.5. MOD_SetServerCoilChangedEventHandler

LONG ADS_API MOD_SetServerCoilChangedEventHandler(OnCoilChangedEvent i_evtHandle);

Purpose:

Set the coil status data changed event handler of the MODBUS server. Any clients try to write the coil status data to the server will cause the call back function to be called.

Parameters:

i_evtHandle = The pointer of the call back function.

Return:

ERR_SUCCESS, get the coil status data succeeded.

ERR_PARAMETER_INVALID, indicates the i_evtHandle is NULL.

5.6. MOD_SetServerHoldRegChangedEventHandler

LONG ADS_API MOD_SetServerHoldRegChangedEventHandler(OnHoldRegChangedEvent i_evtHandle);

Purpose:

Set the holding register data changed event handler of the MODBUS server. Any clients try to write the holding register data to the server will cause the call back function to be called.

Parameters:

i_evtHandle = The pointer of the call back function.

Return:

ERR_SUCCESS, get the coil status data succeeded.

ERR_PARAMETER_INVALID, indicates the i_evtHandle is NULL.

5.7. MOD_SetTcpServerClientConnectEventHandler

LONG ADS_API MOD_SetTcpServerClientConnectEventHandler(OnRemoteTcpClientConnectEvent i_evtHandle);

Purpose:

Set the MODBUS-TCP client connect to the server event handler. Any clients connect to the server will cause the call back function to be called.

Parameters:

i_evtHandle = The pointer of the call back function.

Return:

ERR_SUCCESS, get the coil status data succeeded.

ERR_PARAMETER_INVALID, indicates the i_evtHandle is NULL.

5.8. MOD_SetTcpServerClientDisconnectEventHandler

LONG ADS_API

MOD_SetTcpServerClientDisconnectEventHandler(OnRemoteTcpClientDisconnectEvent i_evtHandle);

Purpose:

Set the MODBUS-TCP client disconnect from the server event handler. Any clients disconnect from the server will cause the call back function to be called.

Parameters:

i_evtHandle = The pointer of the call back function.

Return:

ERR_SUCCESS, get the coil status data succeeded.

ERR_PARAMETER_INVALID, indicates the i_evtHandle is NULL.

6. MODBUS-TCP client functions

6.1. MOD_AddTcpClientConnect

LONG ADS_API MOD_AddTcpClientConnect(char *i_szServerIp,
int i_iScanInterval,
int i_iConnectTimeout,
int i_iTransactTimeout,
OnConnectTcpServerCompletedEvent connEvtHandle,
OnDisconnectTcpServerCompletedEvent DisconnEvtHandle,
unsigned long *o_ulClientHandle);

Purpose:

Add a MODBUS-TCP client into the polling list. This function has to be called before calling MOD_StartTcpClient.

Parameters:

i_szServerIp = The remote server IP the client will connect to.

i_iScanInterval = The scan interval for tag data reading.

i_iConnectTimeout = The connection timeout.

i_iTransactTimeout = The read/write timeout.

connEvtHandle = The pointer of the call back function when connect to server completed.

disconnEvtHandle = The pointer of the call back function when disconnect from server completed.

o_ulClientHandle = The MODBUS-TCP client handle.

Return:

ERR_SUCCESS, add the MODBUS-TCP client succeeded.

ERR_THREAD_RUNNING, indicates the client thread is running.

ERR_MEMALLOC_FAILED, allocate memory failed.

6.2. MOD_AddTcpClientReadTag

LONG ADS_API MOD_AddTcpClientReadTag(unsigned long i_ulClientHandle,
unsigned char i_byAddr,
unsigned char i_byType,
unsigned short i_iStartIndex,
unsigned short i_iTotalPoint,
OnModbusReadCompletedEvent evtHandle);

Purpose:

Add a MODBUS-TCP client tag into the data polling list. This function has to be called after

calling MOD_AddTcpClientConnect, and before calling MOD_StartTcpClient.

Parameters:

i_ulClientHandle = The MODBUS-TCP client handle.

i_byAddr = The slave (server) address, normally is set to 1.

i_byType = The MODBUS reading type.

The following values are possible for this member.

MODBUS_READCOILSTATUS

MODBUS_READINPUTSTATUS

MODBUS_READHOLDREG

MODBUS_READINPUTREG

i_iStartIndex = The start address for reading.

i_iTotalPoint = The total point for reading.

evtHandle = The pointer of the call back function when reading data completed.

Return:

ERR_SUCCESS, add the MODBUS-TCP client succeeded.

ERR_THREAD_RUNNING, indicates the client thread is running.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_PARAMETER_INVALID, indicates the i_byType is invalid.

ERR_TCPCLIENT_INVALID, indicates i_ulClientHandle is invalid

6.3. MOD_StartTcpClient

LONG ADS_API MOD_StartTcpClient();

Purpose:

Start the MODBUS-TCP client. After calling with ERR_SUCCESS returned, the client thread will start polling all clients and their reading tags.

Parameters:

None

Return:

ERR_SUCCESS, started MODBUS-TCP client succeeded.

ERR_CREATECLN_FAILED, failed to create the MODBUS-TCP client.

6.4. MOD_StopTcpClient

LONG ADS_API MOD_StopTcpClient();

Purpose:

Stop the MODBUS-TCP client. After calling this function, the client thread will terminate and all polling will stop.

Parameters:

None

Return:

ERR_SUCCESS, stopped MODBUS-TCP client succeeded.

6.5. MOD_SetTcpClientPriority

LONG ADS_API MOD_SetTcpClientPriority(int iPriority);

Purpose:

Set the priority of the MODBUS-TCP client running thread.

Before calling this function, the MODBUS-TCP client has to be successfully started.

Parameters:

i_iPriority = Specifies the priority value for the client thread.

This parameter can be one of the following values

THREAD_PRIORITY_TIME_CRITICAL indicates 3 points above normal priority.

THREAD_PRIORITY_HIGHEST indicates 2 points above normal priority.

THREAD_PRIORITY_ABOVE_NORMAL indicates 1 point above normal priority.

THREAD_PRIORITY_NORMAL indicates normal priority.

THREAD_PRIORITY_BELOW_NORMAL indicates 1 point below normal priority.

THREAD_PRIORITY_LOWEST indicates 2 points below normal priority.

THREAD_PRIORITY_ABOVE_IDLE indicates 3 points below normal priority.

THREAD_PRIORITY_IDLE indicates 4 points below normal priority.

Return:

ERR_SUCCESS, set MODBUS-TCP client thread priority succeeded.

ERR_SETPRIO_FAILED, failed to set the thread priority.

6.6. MOD_ReleaseTcpClientResource

LONG ADS_API MOD_ReleaseTcpClientResource();

Purpose:

Release all the allocated memory for clients. This function has to be called after calling MOD_StopTcpClient.

Parameters:

None

Return:

ERR_SUCCESS, released MODBUS-TCP client succeeded.

ERR_THREAD_RUNNING, indicates the client thread is running.

6.7. MOD_ForceTcpClientSingleCoil

LONG ADS_API MOD_ForceTcpClientSingleCoil(unsigned long i_ulClientHandle,
unsigned char i_byAddr,
unsigned short i_iIndex,

unsigned short i_iData,
OnModbusWriteCompletedEvent evtHandle);

Purpose:

Set the single coil output.

Parameters:

i_ulClientHandle = The MODBUS-TCP client handle.

i_byAddr = The slave (server) address, normally is set to 1.

i_iIndex = The coil address for writing.

i_iData = The data to write. Set this value to 0 means the coil will be set to OFF, otherwise the coil will be set to ON.

evtHandle = The pointer of the call back function when writing data completed.

Return:

ERR_SUCCESS, released MODBUS-TCP client succeeded.

ERR_THREAD_STOP, indicates the client thread is stopped.

ERR_TCPCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

ERR_SOCKET_CLOSED, indicates the client socket is closed.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_THREAD_BUSY, indicates the client is busy for previous writing.

6.8. MOD_PresetTcpClientSingleReg

LONG ADS_API MOD_PresetTcpClientSingleReg(unsigned long i_ulClientHandle,
unsigned char i_byAddr,
unsigned short i_iIndex,
unsigned short i_iData,
OnModbusWriteCompletedEvent evtHandle);

Purpose:

Set the single holding register output.

Parameters:

i_ulClientHandle = The MODBUS-TCP client handle.

i_byAddr = The slave (server) address, normally is set to 1.

i_iIndex = The holding register address for writing.

i_iData = The data to write.

evtHandle = The pointer of the call back function when writing data completed.

Return:

ERR_SUCCESS, released MODBUS-TCP client succeeded.

ERR_THREAD_STOP, indicates the client thread is stopped.

ERR_TCPCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

ERR_SOCKET_CLOSED, indicates the client socket is closed.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_THREAD_BUSY, indicates the client is busy for previous writing.

6.9. MOD_ForceTcpClientMultiCoils

LONG ADS_API MOD_ForceTcpClientMultiCoils(unsigned long i_ulClientHandle,
unsigned char i_byAddr,
unsigned short i_iStartIndex,
unsigned short i_iTotalPoint,
unsigned char i_byTotalByte,
unsigned char *i_byData,
OnModbusWriteCompletedEvent evtHandle);

Purpose:

Set the multiple coils output.

Parameters:

i_ulClientHandle = The MODBUS-TCP client handle.

i_byAddr = The slave (server) address, normally is set to 1.

i_iStartIndex = The starting coil address for writing.

i_iTotalPoint = The total of coil points to write.

i_byTotalByte = The total byte length of data.

i_byData = The byte data of coils to write. For example, if the i_iTotalPoint is 40, then the i_byTotalByte must be 5 (8 coils form 1 byte). The order of this buffer is as follow:
[Coil 0~7], [Coil 8~15]…

evtHandle = The pointer of the call back function when writing data completed.

Return:

ERR_SUCCESS, released MODBUS-TCP client succeeded.

ERR_THREAD_STOP, indicates the client thread is stopped.

ERR_TCPCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

ERR_SOCKET_CLOSED, indicates the client socket is closed.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_THREAD_BUSY, indicates the client is busy for previous writing.

6.10. MOD_ForceTcpClientMultiRegs

LONG ADS_API MOD_ForceTcpClientMultiRegs(unsigned long i_ulClientHandle,
unsigned char i_byAddr,
unsigned short i_iStartIndex,
unsigned short i_iTotalPoint,
unsigned char i_byTotalByte,
unsigned char *i_byData,

OnModbusWriteCompletedEvent evtHandle);

Purpose:

Set the multiple holding registers output.

Parameters:

i_ulClientHandle = The MODBUS-TCP client handle.

i_byAddr = The slave (server) address, normally is set to 1.

i_iStartIndex = The starting holding register address for writing.

i_iTotalPoint = The total of holding registers to write.

i_byTotalByte = The total byte length of data.

i_byData = The byte data of holding registers to write. For example, if the i_iTotalPoint is 40, then the i_byTotalByte must be 80 (each register is 2 bytes long). The order of this buffer is as follow:

[Reg-0 High byte], [Reg-0 Low byte], [Reg-1 High byte], [Reg-1 Low byte]…

evtHandle = The pointer of the call back function when writing data completed.

Return:

ERR_SUCCESS, released MODBUS-TCP client succeeded.

ERR_THREAD_STOP, indicates the client thread is stopped.

ERR_TCPCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

ERR_SOCKET_CLOSED, indicates the client socket is closed.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_THREAD_BUSY, indicates the client is busy for previous writing.

- **MODBUS-TCP client sample code**

```
#include <stdio.h>
#include <windows.h>
#include <conio.h>
#include "ADSMOD.h"

void ConnectTcpServerCompletedEventHandler(long lResult, char *i_szIp);
void DisconnectTcpServerCompletedEventHandler(long lResult, char *i_szIp);
void ModbusWriteCompletedEventHandler(long lResult);

void ClientReadCoil_1_32_Handler(long lResult, unsigned char *i_byData, int i_iLen);
void ClientReadCoil_65_96_Handler(long lResult, unsigned char *i_byData, int i_iLen);

DWORD dwStartTick, dwCurTick, dwWriteTick;
DWORD dwReadFail, dwWriteFail, dwConnect, dwDisconnect;

int main(int argc, char* argv[])
{
	unsigned long ulClientHandle;
	int iAddr = 1, iIndex = 24;
	int iCount = 0;
	int iCoil = 0, iPoints = 8, iLen = 1;
	unsigned char byData[8] = {0};
	char ch;
	DWORD dwWriteInterval = 1000;
	DWORD dwCount;
	LONG lRet;

	if (ERR_SUCCESS == MOD_Initialize())
	{
		if (ERR_SUCCESS == MOD_AddTcpClientConnect("10.0.0.1",	// remote server IP
							100,		// scan interval (ms)
							3000,		// connection timeout (ms)
							100,		// transaction timeout (ms)
							ConnectTcpServerCompletedEventHandler,
							DisconnectTcpServerCompletedEventHandler,
							&ulClientHandle))
		{
```

```
                MOD_AddTcpClientReadTag(ulClientHandle,
                                                1,
                                                MODBUS_READCOILSTATUS,
                                                1,
                                                32,
                                                ClientReadCoil_1_32_Handler);
                MOD_AddTcpClientReadTag(ulClientHandle,
                                                1,
                                                MODBUS_READCOILSTATUS,
                                                65,
                                                32,
                                                ClientReadCoil_65_96_Handler);
                if (ERR_SUCCESS == MOD_StartTcpClient())
                {
                      printf("MODBUS-TCP client started...\n");
                      dwStartTick = GetTickCount();
                      dwCount = 1;
                      dwReadFail = 0;
                      dwWriteFail = 0;
                      dwConnect = 0;
                      dwDisconnect = 0;
                      while (_kbhit() == 0)
                      {
                            dwCurTick = GetTickCount();
                            if (dwCurTick >= (dwStartTick + dwWriteInterval * dwCount))
                            {
                                  dwCount++;
                                  lRet = MOD_ForceTcpClientMultiCoils(ulClientHandle, iAddr, iIndex,
iPoints, iLen, byData, ModbusWriteCompletedEventHandler);
                                  if (ERROR_SUCCESS == lRet)
                                  {
                                        byData[0]++;
                                        dwWriteTick = dwCurTick;
                                       printf("MOD_ForceTcpClientMultiCoils start (%d) = %d\n", dwCount,
dwWriteTick);
                                  }
                                  else
                                  {
```

```
                                        printf("MOD_ForceTcpClientMultiCoils ret = %d\n", lRet);
                                }
                        }
                        Sleep(1);
                }
                MOD_StopTcpClient();
        }
        MOD_ReleaseTcpClientResource();
    }
    MOD_Terminate();
}
ch = _getch();
printf("Loop count = %d\n", dwCount);
printf("Read failed = %d\n", dwReadFail);
printf("Write failed = %d\n", dwWriteFail);
printf("Connect count = %d\n", dwConnect);
printf("Disconnect count = %d\n", dwDisconnect);
printf("Press any key to exit!\n");
ch = _getch();
return 0;
}

void ConnectTcpServerCompletedEventHandler(long lResult, char *i_szIp)
{
    printf("Connect to '%s' result = %d\n", i_szIp, lResult);
    dwConnect++;
}

void DisconnectTcpServerCompletedEventHandler(long lResult, char *i_szIp)
{
    printf("Disconnect from '%s' result = %d\n", i_szIp, lResult);
    dwDisconnect++;
}

void ModbusWriteCompletedEventHandler(long lResult)
{
    DWORD gapTick = GetTickCount() - dwWriteTick;
    printf("Coil write result = %d (%d ms)\n", lResult, gapTick);
```

```
	if (lResult != ERR_SUCCESS)
		dwWriteFail++;
}

void ClientReadCoil_1_32_Handler(long lResult, unsigned char *i_byData, int i_iLen)
{
	int i;

	if (lResult == ERR_SUCCESS)
	{
		printf("Read[1~32] = ");
		for (i=0; i<i_iLen; i++)
		{
			printf("%02X ", i_byData[i]);
		}
		printf("\n");
	}
	else
	{
		printf("Failed to read err = %d\n", lResult);
		dwReadFail++;
	}
}

void ClientReadCoil_65_96_Handler(long lResult, unsigned char *i_byData, int i_iLen)
{
	int i;

	if (lResult == ERR_SUCCESS)
	{
		printf("Read[65~96] = ");
		for (i=0; i<i_iLen; i++)
		{
			printf("%02X ", i_byData[i]);
		}
		printf("\n");
	}
	else
```

```
        {
                printf("Failed to read err = %d\n", lResult);
                dwReadFail++;
        }
}
```

7. MODBUS-RTU client functions

7.1. MOD_AddRtuClientConnect

```
LONG ADS_API MOD_AddRtuClientConnect(int i_iCom,
                                     unsigned long i_iBaudrate,
                                     unsigned char i_byDataBits,
                                     unsigned char i_byParity,
                                     unsigned char i_byStopBits,
                                     int i_iScanInterval,
                                     int i_iTransactTimeout,
                                     OnConnectRtuServerCompletedEvent connEvtHandle,
                                     OnDisconnectRtuServerCompletedEvent disconnevtHandle,
                                     unsigned long *o_ulClientHandle);
```

Purpose:

Add a MODBUS-RTU client into the polling list. This function has to be called before calling MOD_StartRtuClient.

Parameters:

i_iCom = The COM port number the client will connect to.

i_iBaudrate = The baud rate of the COM port.

i_byDataBits = The data bits of the COM port.

i_byParity = The parity of the COM port.

i_byStopBits = The stop bits of the COM port

i_iScanInterval = The scan interval for tag data reading.

i_iTransactTimeout = The read/write timeout.

connEvtHandle = The pointer of the call back function when open the COM port completed.

disconnEvtHandle = The pointer of the call back function when close the COM port completed.

o_ulClientHandle = The MODBUS-RTU client handle.

Return:

ERR_SUCCESS, add the MODBUS-RTU client succeeded.

ERR_THREAD_RUNNING, indicates the client thread is running.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_RTUCOM_INUSED, indicates the COM port is used by other application.

7.2. MOD_AddRtuClientReadTag

```
LONG ADS_API MOD_AddRtuClientReadTag(unsigned long i_ulClientHandle,
                                     unsigned char i_byAddr,
```

unsigned char i_byType,
unsigned short i_iStartIndex,
unsigned short i_iTotalPoint,
OnModbusReadCompletedEvent evtHandle);

Purpose:

Add a MODBUS-RTU client tag into the data polling list. This function has to be called after calling MOD_AddRtuClientConnect, and before calling MOD_StartRtuClient.

Parameters:

i_ulClientHandle = The MODBUS-RTU client handle.

i_byAddr = The slave (server) address.

i_byType = The MODBUS reading type.

The following values are possible for this member.

MODBUS_READCOILSTATUS

MODBUS_READINPUTSTATUS

MODBUS_READHOLDREG

MODBUS_READINPUTREG

i_iStartIndex = The start address for reading.

i_iTotalPoint = The total point for reading.

evtHandle = The pointer of the call back function when reading data completed.

Return:

ERR_SUCCESS, add the MODBUS-RTU client succeeded.

ERR_THREAD_RUNNING, indicates the client thread is running.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_PARAMETER_INVALID, indicates the i_byType is invalid.

ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

7.3. MOD_StartRtuClient

LONG ADS_API MOD_StartRtuClient();

Purpose:

Start the MODBUS-RTU client. After calling with ERR_SUCCESS returned, the client thread will start polling all clients and their reading tags.

Parameters:

None

Return:

ERR_SUCCESS, started MODBUS-RTU client succeeded.

ERR_CREATECLN_FAILED, failed to create the MODBUS-RTU client.

7.4. MOD_StopRtuClient

LONG ADS_API MOD_StopRtuClient();

Purpose:

Stop the MODBUS-RTU client. After calling this function, the client thread will terminate and all polling will stop.

Parameters:

None

Return:

ERR_SUCCESS, stopped MODBUS-RTU client succeeded.

7.5. MOD_SetRtuClientPriority

LONG ADS_API MOD_SetRtuClientPriority(int iPriority);

Purpose:

Set the priority of the MODBUS-RTU client running thread.

Before calling this function, the MODBUS-RTU client has to be successfully started.

Parameters:

i_iPriority = Specifies the priority value for the client thread.

This parameter can be one of the following values

THREAD_PRIORITY_TIME_CRITICAL indicates 3 points above normal priority.

THREAD_PRIORITY_HIGHEST indicates 2 points above normal priority.

THREAD_PRIORITY_ABOVE_NORMAL indicates 1 point above normal priority.

THREAD_PRIORITY_NORMAL indicates normal priority.

THREAD_PRIORITY_BELOW_NORMAL indicates 1 point below normal priority.

THREAD_PRIORITY_LOWEST indicates 2 points below normal priority.

THREAD_PRIORITY_ABOVE_IDLE indicates 3 points below normal priority.

THREAD_PRIORITY_IDLE indicates 4 points below normal priority.

Return:

ERR_SUCCESS, set MODBUS-RTU client thread priority succeeded.

ERR_SETPRIO_FAILED, failed to set the thread priority.

7.6. MOD_ReleaseRtuClientResource

LONG ADS_API MOD_ReleaseRtuClientResource();

Purpose:

Release all the allocated memory for clients. This function has to be called after calling MOD_StopRtuClient.

Parameters:

None

Return:

ERR_SUCCESS, released MODBUS-RTU client succeeded.

ERR_THREAD_RUNNING, indicates the client thread is running.

7.7. MOD_ForceRtuClientSingleCoil

LONG ADS_API MOD_ForceRtuClientSingleCoil(unsigned long i_ulClientHandle,
unsigned char i_byAddr,
unsigned short i_iIndex,
unsigned short i_iData,
OnModbusWriteCompletedEvent evtHandle);

Purpose:

Set the single coil output.

Parameters:

i_ulClientHandle = The MODBUS-RTU client handle.

i_byAddr = The slave (server) address.

i_iIndex = The coil address for writing.

i_iData = The data to write. Set this value to 0 means the coil will be set to OFF, otherwise the coil will be set to ON.

evtHandle = The pointer of the call back function when writing data completed.

Return:

ERR_SUCCESS, released MODBUS-RTU client succeeded.

ERR_THREAD_STOP, indicates the client thread is stopped.

ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_THREAD_BUSY, indicates the client is busy for previous writing.

7.8. MOD_PresetRtuClientSingleReg

LONG ADS_API MOD_PresetRtuClientSingleReg(unsigned long i_ulClientHandle,
unsigned char i_byAddr,
unsigned short i_iIndex,
unsigned short i_iData,
OnModbusWriteCompletedEvent evtHandle);

Purpose:

Set the single holding register output.

Parameters:

i_ulClientHandle = The MODBUS-RTU client handle.

i_byAddr = The slave (server) address.

i_iIndex = The holding register address for writing.

i_iData = The data to write.

evtHandle = The pointer of the call back function when writing data completed.

Return:

ERR_SUCCESS, released MODBUS-RTU client succeeded.

ERR_THREAD_STOP, indicates the client thread is stopped.

ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_THREAD_BUSY, indicates the client is busy for previous writing.

7.9. MOD_ForceRtuClientMultiCoils

```
LONG ADS_API MOD_ForceRtuClientMultiCoils(unsigned long i_ulClientHandle,
                                          unsigned char i_byAddr,
                                          unsigned short i_iStartIndex,
                                          unsigned short i_iTotalPoint,
                                          unsigned char i_byTotalByte,
                                          unsigned char *i_byData,
                                          OnModbusWriteCompletedEvent evtHandle);
```

Purpose:

Set the multiple coils output.

Parameters:

i_ulClientHandle = The MODBUS-RTU client handle.

i_byAddr = The slave (server) address.

i_iStartIndex = The starting coil address for writing.

i_iTotalPoint = The total of coil points to write.

i_byTotalByte = The total byte length of data.

i_byData = The byte data of coils to write. For example, if the i_iTotalPoint is 40, then the i_byTotalByte must be 5 (8 coils form 1 byte). The order of this buffer is as follow:

[Coil 0~7], [Coil 8~15]…

evtHandle = The pointer of the call back function when writing data completed.

Return:

ERR_SUCCESS, released MODBUS-RTU client succeeded.

ERR_THREAD_STOP, indicates the client thread is stopped.

ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_THREAD_BUSY, indicates the client is busy for previous writing.

7.10. MOD_ForceRtuClientMultiRegs

```
LONG ADS_API MOD_ForceRtuClientMultiRegs( unsigned long i_ulClientHandle,
                                          unsigned char i_byAddr,
                                          unsigned short i_iStartIndex,
```

unsigned short i_iTotalPoint,
unsigned char i_byTotalByte,
unsigned char *i_byData,
OnModbusWriteCompletedEvent evtHandle);

Purpose:

Set the multiple holding registers output.

Parameters:

i_ulClientHandle = The MODBUS-RTU client handle.

i_byAddr = The slave (server) address.

i_iStartIndex = The starting holding register address for writing.

i_iTotalPoint = The total of holding registers to write.

i_byTotalByte = The total byte length of data.

i_byData = The byte data of holding registers to write. For example, if the i_iTotalPoint is 40, then the i_byTotalByte must be 80 (each register is 2 bytes long). The order of this buffer is as follow:

[Reg-0 High byte], [Reg-0 Low byte], [Reg-1 High byte], [Reg-1 Low byte]…

evtHandle = The pointer of the call back function when writing data completed.

Return:

ERR_SUCCESS, released MODBUS-RTU client succeeded.

ERR_THREAD_STOP, indicates the client thread is stopped.

ERR_RTUCLIENT_INVALID, indicates the i_ulClientHandle is invalid.

ERR_MEMALLOC_FAILED, allocate memory failed.

ERR_THREAD_BUSY, indicates the client is busy for previous writing.

● MODBUS-RTU client sample code

```
#include <stdio.h>
#include <windows.h>
#include <conio.h>
#include "ADSMOD.h"

void ConnectRtuServerCompletedEventHandler(long lResult, int i_iCom);
void DisconnectRtuServerCompletedEventHandler(long lResult, int i_iCom);
void ModbusWriteCompletedEventHandler(long lResult);

void ClientReadCoil_1_32_Handler(long lResult, unsigned char *i_byData, int i_iLen);
void ClientReadCoil_65_96_Handler(long lResult, unsigned char *i_byData, int i_iLen);

DWORD dwStartTick, dwCurTick, dwWriteTick;
DWORD dwReadFail, dwWriteFail, dwConnect, dwDisconnect;

int main(int argc, char* argv[])
{
	unsigned long ulClientHandle;
	int iAddr = 1, iIndex = 24;
	int iCount = 0;
	int iCoil = 0, iPoints = 8, iLen = 1;
	unsigned char byData[8] = {0};
	char ch;
	DWORD dwWriteInterval = 1000;
	DWORD dwCount;
	LONG lRet;

	if (ERR_SUCCESS == MOD_Initialize())
	{
		if (ERR_SUCCESS == MOD_AddRtuClientConnect( 4,
							CBR_9600,
							8,
							NOPARITY,
							ONESTOPBIT,
							500,
							200,
							ConnectRtuServerCompletedEventHandler,
```

```
                                        DisconnectRtuServerCompletedEventHandler,
                                        &ulClientHandle))
        {
            MOD_AddRtuClientReadTag(ulClientHandle,
                                        1,
                                        MODBUS_READCOILSTATUS,
                                        1,
                                        32,
                                        ClientReadCoil_1_32_Handler);
            MOD_AddRtuClientReadTag(ulClientHandle,
                                        1,
                                        MODBUS_READCOILSTATUS,
                                        65,
                                        32,
                                        ClientReadCoil_65_96_Handler);
            if (ERR_SUCCESS == MOD_StartRtuClient())
            {
                MOD_SetRtuClientPriority(THREAD_PRIORITY_HIGHEST);
                printf("MODBUS-RTU client started...\n");
                dwStartTick = GetTickCount();
                dwCount = 1;
                dwReadFail = 0;
                dwWriteFail = 0;
                dwConnect = 0;
                dwDisconnect = 0;
                while (_kbhit() == 0)
                {
                    dwCurTick = GetTickCount();
                    if (dwCurTick >= dwStartTick + dwWriteInterval * dwCount)
                    {
                        dwCount++;
                        lRet = MOD_ForceRtuClientMultiCoils(ulClientHandle, iAddr, iIndex,
iPoints, iLen, byData, ModbusWriteCompletedEventHandler);
                        if (ERROR_SUCCESS == lRet)
                        {
                            byData[0]++;
                            dwWriteTick = dwCurTick;
                            printf("MOD_ForceRtuClientMultiCoils start = %d\n",
```

```
dwWriteTick);
                                    }
                                    else
                                    {
                                          printf("MOD_ForceRtuClientMultiCoils ret = %d\n", lRet);
                                    }
                              }
                              Sleep(1);
                        }
                        MOD_StopRtuClient();
                  }
                  MOD_ReleaseRtuClientResource();
            }
            MOD_Terminate();
      }
      ch = _getch();
      printf("Loop count = %d\n", dwCount);
      printf("Read failed = %d\n", dwReadFail);
      printf("Write failed = %d\n", dwWriteFail);
      printf("Connect count = %d\n", dwConnect);
      printf("Disconnect count = %d\n", dwDisconnect);
      printf("Press any key to exit!\n");
      ch = _getch();
      return 0;
}

void ConnectRtuServerCompletedEventHandler(long lResult, int i_iCom)
{
      printf("Connect to COM-%d result = %d\n", i_iCom, lResult);
      dwConnect++;
}

void DisconnectRtuServerCompletedEventHandler(long lResult, int i_iCom)
{
      printf("Disconnect from COM-%d result = %d\n", i_iCom, lResult);
      dwDisconnect++;
}
```

```
void ModbusWriteCompletedEventHandler(long lResult)
{
	DWORD gapTick = GetTickCount() - dwWriteTick;
	printf("Coil write result = %d (%d ms)\n", lResult, gapTick);
	if (lResult != ERR_SUCCESS)
		dwWriteFail++;
}

void ClientReadCoil_1_32_Handler(long lResult, unsigned char *i_byData, int i_iLen)
{
	int i;

	if (lResult == ERR_SUCCESS)
	{
		printf("Read[1~32] = ");
		for (i=0; i<i_iLen; i++)
		{
			printf("%02X ", i_byData[i]);
		}
		printf("\n");
	}
	else
	{
		printf("Failed to read err = %d\n", lResult);
		dwReadFail++;
	}
}

void ClientReadCoil_65_96_Handler(long lResult, unsigned char *i_byData, int i_iLen)
{
	int i;

	if (lResult == ERR_SUCCESS)
	{
		printf("Read[65~96] = ");
		for (i=0; i<i_iLen; i++)
		{
			printf("%02X ", i_byData[i]);
```

```
            }
            printf("\n");
        }
        else
        {
            printf("Failed to read err = %d\n", lResult);
            dwReadFail++;
        }
}
```

8. MODBUS client call back functions

8.1. OnConnectTcpServerCompletedEvent

typedef void (*OnConnectTcpServerCompletedEvent)(long lResult, char *i_szIp);

Purpose:

The prototype of call back function when the connection to the MODBUS-TCP server completed event occurred.

Parameters:

lResult = The result of the connection.

i_szIp = The IP of the MODBUS-TCP server the client connects to.

Return:

None.

8.2. OnDisconnectTcpServerCompletedEvent

typedef void (*OnDisconnectTcpServerCompletedEvent)(long lResult, char *i_szIp);

Purpose:

The prototype of call back function when the connection to the MODBUS-TCP server closed event occurred.

Parameters:

lResult = The result of the disconnection.

i_szIp = The IP of the MODBUS-TCP server the client disconnect from.

Return:

None.

8.3. OnConnectRtuServerCompletedEvent

typedef void (*OnConnectRtuServerCompletedEvent)(long lResult, int i_iCom);

Purpose:

The prototype of call back function when the open COM port completed event occurred.

Parameters:

lResult = The result of the open process.

i_iCom = The COM port the client opened.

Return:

None.

8.4. OnDisconnectRtuServerCompletedEvent

typedef void (*OnDisconnectRtuServerCompletedEvent)(long lResult, int i_iCom);

Purpose:

The prototype of call back function when the close COM port completed event occurred.

Parameters:

lResult = The result of the close process.

i_iCom = The COM port the client closed.

Return:

None.

8.5. OnModbusReadCompletedEvent

typedef void (*OnModbusReadCompletedEvent)(long lResult, unsigned char *i_byData, int i_iLen);

Purpose:

The prototype of call back function when the read data completed event occurred.

Parameters:

lResult = The result of the reading.

i_byData = The pointer of buffer that hold the read data.

i_iLen = The length of the data read.

Return:

None.

8.6. OnModbusWriteCompletedEvent

typedef void (*OnModbusWriteCompletedEvent)(long lResult);

Purpose:

The prototype of call back function when the write data completed event occurred.

Parameters:

lResult = The result of the reading.

Return:

None.

9. ADSMOD library error code

Code	Description	Name
0		ERR_SUCCESS
801		ERR_WSASTART_FAILED
802		ERR_WSACLEAN_FAILED
803		ERR_CREATESOCK_FAILED
804		ERR_CREATESVR_FAILED
805		ERR_PARAMETER_INVALID
806		ERR_WAIT_TIMEOUT
807		ERR_CONNECT_FAILED
808		ERR_CLOSESOCKET_FAILED
809		ERR_MEMALLOC_FAILED
810		ERR_CREATECLN_FAILED
811		ERR_TCPCLIENT_INVALID
812		ERR_TCPSEND_FAILED
813		ERR_TCPRECV_FAILED
814		ERR_SOCKET_CLOSED
815		ERR_THREAD_RUNNING
816		ERR_THREAD_STOP
817		ERR_THREAD_BUSY
818		ERR_RTUCLIENT_INVALID
819		ERR_RTUCOM_INUSED
820		ERR_RTUCRC_INVALID
821		ERR_SETPRIO_FAILED
901	Illegal function	ERR_MODEXCEPT_01
902	Illegal data address	ERR_MODEXCEPT_02
903	Illegal data value	ERR_MODEXCEPT_03
904	Slave device failure	ERR_MODEXCEPT_04
905	Acknowledge	ERR_MODEXCEPT_05
906	Slave device busy	ERR_MODEXCEPT_06
907	Negative acknowledge	ERR_MODEXCEPT_07
908	Memory parity error	ERR_MODEXCEPT_08
999	Invalid packet	ERR_MODEXCEPT_99