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);
			}
```