

Identium UDR-111 UHF Desktop Reader

Interface Functions — DLLProject_x86.dll / DLLProject_x64.dll

© 2026 Identium Tech Solutions Pvt. Ltd. · info@identium.in · <https://identiumone.com>

Catalogue

Interface Functions.....	3
Callback Function Definition.....	3
Establish Connection with Device.....	4
Disconnect From Device.....	5
Get Informations of the Reader Module.....	6
Single Polling Read.....	7
Set “Select” Parameters.....	8
Set “Select” Mode.....	9
Read RFID Data Storage.....	10
Write RFID Data Storage.....	11
Lock RFID Data Storage.....	12
Kill RFID.....	13
Get Query Parameters.....	14
Set Query Parameters.....	15
Set Working Region.....	16
Set Working Channel.....	17
Get Working Channel.....	18
Set FHSS.....	19
Set Transmitting Power.....	20
Get Transmitting Power.....	21
Set Continuous Carrier Wave.....	22
Set Modem Parameters.....	23
Get Modem Parameters.....	24
Scan Jammer on RF Input.....	25
Scan RSSI.....	26
NXP ReadProtect/Reset ReadProtect.....	27
NXP Change EAS.....	28

NXP EAS_Alarm.....	29
NXP ChangeConfig.....	30
Impinj Monza QT.....	31
NetCfg_Open.....	32
NetCfg_Close.....	33
NetCfg_SearchForDevices.....	34
NET_DEVICE_CONFIG Structure.....	35
DEVICEHW_CONFIG Structure.....	36
DEVICEPORT_CONFIG Structure.....	37
NetCfg_GetInfo.....	38
NetCfg_SetInfo.....	39
NetCfg_FactoryReset.....	40

Interface Functions

Callback Function Definition

Function			
DWORD(WINAPI *Async_Receive)(BYTE Type, BYTE Command, DWORD ParamSize, LPBYTE ParamData);			
Remarks			
When there is notice created by devices, call this function to inform the host.			
Parameter	Data Type		Description
Type	Byte	out	Type 0x02; Notice frame: from device to host
Command	Byte	out	Type of command
ParamSize	DWORD	out	Length of content
ParamData	Byte[]	out	Content
Return			
Return	DWORD	out	Return value: 0 for success

Establish Connection with Device

Function			
DWORD Connect(BYTE ConnType, LPSTR ConnChar, Async_Receive Ar);			
Parameter	DataType		Description
ConnType	Byte	in	Type of connection; 1;Serial ports; 2:USB; 3:TCP;
ConnChar	String	in	Content of connection; Serial ports:("COM1"); USB:(NULL); TCP:("IP Address:Port")
Ar	Async_Receive	in	Callback function handle, in order to pass notification, pass NULL if notice is not needed.
Return			
Return	DWORD	out	Return value: 0 for success

Disconnect From Device

Function			
DWORD Disconnect();			
Return			
Return	DWORD	out	Return value: 0 for success

Get Informations of the Reader Module

Function			
DWORD GetModuleInfo(LPBYTE InfoType, LPSTR InfoData, LPDWORD DataSize)			
Remarks			
Get Informations of the reader module, such as hardware version, software version and manufacturer info.			
Parameters	Data Type		Description
InfoType	Byte	in,out	Type; Hardware version: 0x00; Software version: 0x01
InfoData	String	out	Content
DataSize	DWORD	out	Length of content
Return			
Return	DWORD	out	Return value: 0 for success

Single Polling Read

Function			
DWORD ReadSingle();			
Remarks			
If this function is called with a card attached to the device, result will be passed by function Async_Receive.			
Return			
Return	DWORD	out	Return value: 0 for success

Set “Select” Parameters

Function			
DWORD SetSelectParam(BYTE Target, BYTE Action, BYTE MemBank, DWORD Pointer, BYTE Truncated, LPBYTE MaskData, BYTE MaskSize);			
Remarks			
Set Select parameters and set Select mode to 0x02 at the same time. Select command should be sent before polling read the RFIDs. When multiple RFID exists, Select parameters are used to operate on single specific RFID.			
Parameters	DataType		Description
Target	Byte	in	
Action	Byte	in	
MemBank	Byte	in	00: RFU data storage area 01: EPC data storage area 02: TID data storage area 03: User data storage area
Truncated	Byte	in	0x00(0x00 for Disable truncation, 0x80 for Enable truncation)
MaskData	Byte[]		EPC Code
MaskSize	Byte		Length of MaskData
Return			
Return	DWORD	out	Return value: 0 for success

Set “Select” Mode

Function			
DWORD SetSelectMode(BYTE Mode);			
Remarks			
After setting Select parameters, call this function to set Select mode.			
Parameters	Data Type		Description
Mode	Byte	in	0x00: do Select to specify single RFID before ALL operations. 0x01: no Select done before ANY operations. 0x02: do Select before all operations EXCEPT POLLING INVENTORY.
Return			
Return	DWORD	out	Return value: 0 for success

Read RFID Data Storage

Function			
DWORD ReadData(LPBYTE AccessPassword, BYTE MemBank, DWORD StartIndex, DWORD Length, LPBYTE PC, LPBYTE EPC, LPBYTE Data, LPDWORD Size);			
Remarks			
This function is used to read data with specific beginning address and length from memory bank of a single RFID. Unit size of the address offset SA and data length DL is the size of a WORD, which is also the length of 2 bytes or 16 bits. Select command must be sent before reading to choose RFID to be read from. If AccessPassword is made of zeros then Access command will not be sent.			
Parameters	DataType		Description
AccessPassword	Byte[]	in	Access Password
MemBank	Byte	in	00: RFU data storage area 01: EPC data storage area 02: TID data storage area 03: User data storage area
StartIndex	DWORD	in	Beginning address of data to be read
Length	DWORD	in	Length of data to be read
PC	Byte[]	out	PC of the RFID to be read
EPC	Byte[]	out	EPC of the RFID to be read
Data	Byte[]	out	Return data
Size	Byte[]	out	Length of return data
Return			
Return	DWORD	out	Return value: 0 for success

Write RFID Data Storage

Function			
DWORD WriteData(LPBYTE AccessPassword, BYTE MemBank, DWORD StartIndex, LPBYTE Data, DWORD Size, LPBYTE PC, LPBYTE EPC);			
Remarks			
This function is used to write data with specific beginning address and length to memory bank of a single RFID. Unit size of the address offset SA and data length DL is the size of a WORD, which is also the length of 2 bytes or 16 bits. Select command must be sent before writing to choose RFID to be write to. If AccessPassword is made of zeros then Access command will not be sent.			
Parameters	DataType		Description
AccessPassword	Byte[]	in	Access Password
MemBank	Byte	in	00: RFU data storage area 01: EPC data storage area 02: TID data storage area 03: User data storage area
StartIndex	DWORD	in	Beginning address of data to be written
Length	DWORD	in	Length of data to be written
PC	Byte[]	out	PC of the RFID to be written
EPC	Byte[]	out	EPC of the RFID to be written
Data	Byte[]	out	Return data
Size	Byte[]	out	Length of return data
Return			
Return	DWORD	out	Return value: 0 for success

Lock RFID Data Storage

Function			
DWORD LockUnlock(LPBYTE AccessPassword, LPBYTE LD, LPBYTE PC, LPBYTE EPC);			
Remarks			
Lock or Unlock Data Storage of a single RFID. Select command must be sent before locking or unlocking to choose RFID to be operated.			
Parameters	DataType		Description
AccessPassword	Byte[]	in	Access Password
LD	Byte[]	in	The highest 4 bits of LD is preserved, and the remaining 20 bits contains the Payload of the Lock command, including Mask and Action. Mask and Action each takes 10 bits from higher bits to lower ones. For further information, please refer to section 6.3.2.11.3.5 of EPC Gen2 Protocol (v1.2.0). Mask stands for a mask code, Actions are only effective when corresponding mask bit is 1. There are 2 bits of Action in each data storage area, 00~11, each means Open, Open permanently, Locked and Locked permanently.
PC	Byte[]	out	PC of the RFID to be operated
EPC	Byte[]	out	EPC of the RFID to be operated
Return			
Return	DWORD	out	Return value: 0 for success

Kill RFID

Function			
DWORD Kill(LPBYTE AccessPassword, LPBYTE PC, LPBYTE EPC);			
Remarks			
Kill a single RFID. Select command must be sent before this operation to specify single RFID.			
Parameters	DataType		Description
AccessPassword	Byte[]	in	Access Password
PC	Byte[]	out	PC of the RFID to be operated
EPC	Byte[]	out	EPC of the RFID to be operated
Return			
Return	DWORD	out	Return value: 0 for success

Get Query Parameters

Function			
DWORD GetQuery(LPBYTE DR, LPBYTE M, LPBYTE TRext, LPBYTE Sel, LPBYTE Session, LPBYTE Target, LPBYTE Q);			
Remarks			
Get parameters related to Query command from firmware.			
Parameters	Data Type		Description
DR	Byte	out	=0: Mode8 =1: Mode64/3 Only Mode8 supported
M	Byte	out	=0: Mode1 =1: Mode2 =2: Mode4 =3: Mode8 Only Mode1 supported
TRext	Byte	out	=0: No pilot tone =1: Use pilot tone Only Use pilot tone supported
Sel	Byte	out	=0: ALL =1: ALL =2: ~SL =3: SL
Session	Byte	out	=0: S0 =1: S1 =2: S2 =3: S3
Target	Byte	out	=0: A =1: B
Q	Byte	out	0-15;
Return			
Return	DWORD	out	Return value: 0 for success

Set Query Parameters

Function			
DWORD SetQuery(BYTE DR, BYTE M, BYTE TRext, BYTE Sel, BYTE Session, BYTE Target, BYTE Q);			
Remarks			
Set parameters related to Query command.			
Parameters	DataType		Description
DR	Byte	in	=0: Mode8 =1: Mode64/3 Only Mode8 supported
M	Byte	in	=0: Mode1 =1: Mode2 =2: Mode4 =3: Mode8 Only Mode1 supported
TRext	Byte	in	=0: No pilot tone =1: Use pilot tone Only Use pilot tone supported
Sel	Byte	in	=0: ALL =1: ALL =2: ~SL =3: SL
Session	Byte	in	=0: S0 =1: S1 =2: S2 =3: S3
Target	Byte	in	=0: A =1: B
Q	Byte	in	0-15;
Return			
Return	DWORD	out	Return value: 0 for success

Set Working Region

Function			
<code>DWORD SetRegion(BYTE Region);</code>			
Remarks			
Set work region of the reader.			
Parameters	Data Type		Description
Region	Byte	in	Region code 1: China 900MHz 2: USA 3: Europe 4: China 800MHz 6: South Korea
Return			
Return	DWORD	out	Return value: 0 for success

Set Working Channel

Function			
DWORD SetRfChannel(BYTE CH_Index) ;			
Remarks			
Set working channel of the reader.			
Parameters	Data Type		Description
CH_Index	Byte	in	Calculation formula, Freq_CH is the frequency of the channel: China 900MHz $CH_Index = (Freq_CH - 920.125M) / 0.25M$ China 800MHz $CH_Index = (Freq_CH - 840.125M) / 0.25M$ USA $CH_Index = (Freq_CH - 902.25M) / 0.5M$ Europe $CH_Index = (Freq_CH - 865.1M) / 0.2M$ South Korea $CH_Index = (Freq_CH - 917.1M) / 0.2M$
Return			
Return	DWORD	out	Return value: 0 for success

Get Working Channel

Function			
DWORD GetRfChannel(LPBYTE CH_Index);			
Remarks			
Get working channel of the reader.			
Parameters	Data Type		Description
CH_Index	Byte	in	Calculation formula, Freq_CH is the frequency of the channel: China 900MHz $\text{Freq_CH} = \text{CH_Index} * 0.25\text{M} + 920.125\text{M}$ China 800MHz $\text{Freq_CH} = \text{CH_Index} * 0.25\text{M} + 840.125\text{M}$ USA $\text{Freq_CH} = \text{CH_Index} * 0.5\text{M} + 902.25\text{M}$ Europe $\text{Freq_CH} = \text{CH_Index} * 0.2\text{M} + 865.1\text{M}$ South Korea $\text{Freq_CH} = \text{CH_Index} * 0.2\text{M} + 917.1\text{M}$
Return			
Return	DWORD	out	Return value: 0 for success

Set FHSS

Function			
<code>DWORD SetFhss(BOOL Param);</code>			
Remarks			
Set to or cancel frequency-hopping spread spectrum (FHSS).			
Parameters	DataType		Description
Param	Bool	in	TRUE : set FHSS FALSE : cancel FHSS
Return			
Return	DWORD	out	Return value: 0 for success

Set Transmitting Power

Function			
DWORD SetPower(DWORD Power);			
Remarks			
Set transmitting power for current reader.			
Parameters	Data Type		Description
Power	DWORD	in	2000, as 20dBm
Return			
Return	DWORD	out	Return value: 0 for success

Get Transmitting Power

Function			
DWORD GetPower(LPDWORD Power);			
Remarks			
Get transmitting power of current reader.			
Parameters	Data Type		Description
Power	DWORD	out	2000, as 20dBm
Return			
Return	DWORD	out	Return value: 0 for success

Set Continuous Carrier Wave

Function			
DWORD SetCW(BOOL Param);			
Remarks			
Turn on or off transmitting continuous carrier wave.			
Parameters	DataType		Description
Param	Bool	in	TRUE : turn on FALSE : turn off
Return			
Return	DWORD	out	Return value: 0 for success

Set Modem Parameters

Function			
DWORD SetModemPara(BYTE Mixer_G, BYTE IF_G, DWORD Thrd);			
Remarks			
Set modem parameters for current reader, including Mixer Gain, IF AMP Gain and Demodulation threshold.			
Parameters	DataType		Description
Mixer_G	Byte	in	0x00: 0(dB) 0x01: 3(dB) 0x02: 6(dB) 0x03: 9(dB) 0x04: 12(dB) 0x05: 15(dB) 0x06: 16(dB)
IF_G	Byte	in	0x00: 12(dB) 0x01: 18(dB) 0x02: 21(dB) 0x03: 24(dB) 0x04: 27(dB) 0x05: 30(dB) 0x06: 36(dB) 0x07: 40(dB)
Thrd	DWORD	in	When the threshold is lower, the minimum RSSI that could be demodulated is lower, yet more unstable. Demodulation will totally fail if RSSI is too low. On the contrary, higher threshold means only higher RSSI could be demodulated and more stable. 432 is the lowest suggested value.
Return			
Return	DWORD	out	Return value: 0 for success

Get Modem Parameters

Function			
<i>DWORD</i> GetModemPara(<i>LPBYTE</i> Mixer_G, <i>LPBYTE</i> IF_G, <i>LPDWORD</i> Thrd);			
Remarks			
Get modem parameters of current reader, including Mixer Gain, IF AMP Gain and Demodulation threshold.			
Parameters	DataType		Description
Mixer_G	Byte	out	0x00: 0(dB) 0x01: 3(dB) 0x02: 6(dB) 0x03: 9(dB) 0x04: 12(dB) 0x05: 15(dB) 0x06: 16(dB)
IF_G	Byte	out	0x00: 12(dB) 0x01: 18(dB) 0x02: 21(dB) 0x03: 24(dB) 0x04: 27(dB) 0x05: 30(dB) 0x06: 36(dB) 0x07: 40(dB)
Thrd	DWORD	out	When the threshold is lower, the minimum RSSI that could be demodulated is lower, yet more unstable. Demodulation will totally fail if RSSI is too low. On the contrary, higher threshold means only higher RSSI could be demodulated and more stable. 432 is the lowest suggested value.
Return			
Return	DWORD	out	Return value: 0 for success

Scan Jammer on RF Input

Function			
<code>DWORD ScanJammer(LPBYTE CH_L, LPBYTE CH_H, LPBYTE JMR);</code>			
Remarks			
Scan jammer on RF input to check jammer strength on each channel.			
Parameters	Data Type		Description
CH_L	Byte	out	Index of first channel
CH_H	Byte	out	Index of last channel
JMR	Byte[]	out	Each byte refers a strength value of a channel; For example: (0xF2 is for 14dBm) Conversion formula: int jammer = JMR[n]; if (jammer > 127) { jammer = -((-jammer) & 0xFF); }
Return			
Return	DWORD	out	Return value: 0 for success

Scan RSSI

Function			
<i>DWORD</i> ScanRSSI(<i>LPBYTE</i> CH_L, <i>LPBYTE</i> CH_H, <i>LPBYTE</i> JMR);			
Remarks			
Scan RSSI on RF input to check if any reader is currently working.			
Parameters	Data Type		Description
CH_L	Byte	out	Index of first channel
CH_H	Byte	out	Index of last channel
JMR	Byte[]	out	Each byte refers a strength value of a channel; For example : (0xBA = -70 dBm) Conversion formula: int RSSI = JMR[n]; if (RSSI > 127) { RSSI = -((-RSSI) & 0xFF); }
Return			
Return	DWORD	out	Return value: 0 for success

NXP ReadProtect/Reset ReadProtect

Function			
<i>DWORD</i> NxpReadProtect(<i>LPBYTE</i> AccessPassword, <i>BYTE</i> Protect, <i>LPBYTE</i> PC, <i>LPBYTE</i> EPC);			
Remarks			
NXP G2X RFID supports ReadProtect/Reset ReadProtect command. When ReadProtect command is executed, the ProtectEPC and ProtectTID of the RFID will be set to 1, and it will get into the state of data protecting. In order to cancel data protecting and come back to normal, Reset ReadProtect should be executed. Select command must be sent before this operation to specify single RFID.			
Parameters	Data Type		Description
AccessPassword	Byte[]	in	Access Password
Protect	Byte	in	0x00: ReadProtect 0x01: Reset ReadProtect
PC	Byte[]	out	PC of the RFID to be operated
EPC	Byte[]	out	EPC of the RFID to be operated
Return			
Return	DWORD	out	Return value: 0 for success

NXP Change EAS

Function			
<i>DWORD</i> NxpChangeEas(<i>LPBYTE</i> AccessPassword, <i>BYTE</i> Protect, <i>LPBYTE</i> PC, <i>LPBYTE</i> EPC);			
Remarks			
<p>NXP G2X RFID supports Change EAS command. When Change EAS command is executed, the PSF of the RFID will be set to 1 or 0. when PSF of the RFID is set to 1, RFID will respond to EAS_Alarm command, otherwise it won't.</p> <p>Select command must be sent before this operation to specify single RFID.</p>			
Parameters	Data Type		Description
AccessPassword	Byte[]	in	Access Password
Protect	Byte	in	0x00: Set PSF to '0' 0x01: Set PSF to '1'
PC	Byte[]	out	PC of the RFID to be operated
EPC	Byte[]	out	EPC of the RFID to be operated
Return			
Return	DWORD	out	Return value: 0 for success

NXP EAS_Alarm

Function			
<i>DWORD</i> NxpEasAlarm(<i>LPBYTE</i> EASAlarmCode);			
Remarks			
NXP G2X RFID supports EAS_Alarm command. When a RFID receives EAS_Alarm command, it responds 64bits EAS-Alarm code immediately. Notice that only when PSF of the RFID is set to 1 will it respond to this command. This command contributes to a shoplift-preventing system.			
Parameters	Data Type		Description
EASAlarmCode	Byte[]	out	64bits EAS-Alarm code
Return			
Return	DWORD	out	Return value: 0 for success

NXP ChangeConfig

Function			
<i>DWORD</i> NxpChangeConfig(<i>LPBYTE</i> AccessPassword, <i>LPBYTE</i> Config, <i>LPBYTE</i> PC, <i>LPBYTE</i> EPC);			
Remarks			
Some series of NXP G2X RFIDs (such as G2iM and G2iM+) support ChangeConfig command. 16bits Config-Word of the NXP G2X can be read or modified by this command. Config-Word of a NXP G2X RFID is located on Memory Bank 01(EPC zone)at offset 20h (word address, which can be read by common Read command. When RFID is at the state of Secured (Secured State), Config-Word can be modified. Notice that modifying Config-Word means flipping data bits, to be specific, flip bits (1 to 0, 0 to 1) where you input an 1 and hold where you input a 0. Select command must be sent before this operation to specify single RFID.			
Parameters	Data Type		Description
AccessPassword	Byte[]	in	Access Password
Config	Byte[]	in,out	0x0000(returns original Config-Word, same as reading)
PC	Byte[]	out	PC of the RFID to be operated
EPC	Byte[]	out	EPC of the RFID to be operated
Return			
Return	DWORD	out	Return value: 0 for success

Impinj Monza QT

Function			
<i>DWORD</i> ImpinjMonzaQT(<i>LPBYTE</i> AccessPassword, <i>BYTE</i> RW, <i>BYTE</i> Persistence, <i>BYTE</i> Payload, <i>LPBYTE</i> PC, <i>LPBYTE</i> EPC, <i>LPDWORD</i> QTControl);			
Remarks			
<p>Impinj Monza 4QT RFID supports QT command, which modifies the QT Control word of a RFID. Setting QT_SR can cut down operating distance when a RFID is at Open state or at Secured state or about to change state into Open or Secured. Modifying QT_MEM can switch the RFID between using Public Memory Map and using Private Memory Map.</p> <p>Select command must be sent before this operation to specify single RFID.</p>			
Parameters	DataType		Description
AccessPassword	Byte[]	in	Access Password
RW	Byte	in	0x00: Read 0x01: Write
Persistence	Byte	in	0x00: Write into volatile storage area 0x01: Write into non-volatile storage area
Payload	Byte	in	0x01: QT_SR 0x02: QT_MEM
PC	Byte[]	out	PC of the RFID to be operated
EPC	Byte[]	out	EPC of the RFID to be operated
QTControl	Byte[]	out	QT Control Word
Return			
Return	DWORD	out	Return value: 0 for success

NetCfg_Open

Function			
<i>DWORD</i> NetCfg_Open(<i>LPSTR</i> IP);			
Remarks			
Access to NetCfg configurations function.			
Parameter	DataType		Description
IP	String	in	IP address to bind
Return			
Return	DWORD	out	Return value: 0 for success

NetCfg_Close

Function			
<i>DWORD</i> NetCfg_Close();			
Remarks			
Close NetCfg configurations.			
Parameter	DataType		Description
Return			
Return	DWORD	out	Return value: 0 for success

NetCfg_SearchForDevices

Function			
<i>DWORD</i> NetCfg_SearchForDevices(<i>BYTE</i> DeviceType, <i>LPDWORD</i> Count, <i>LPBYTE</i> Data, <i>LPDWORD</i> Length);			
Remarks			
Search for NetCfg devices.			
Parameter	Data Type		Description
DeviceType	BYTE	in	Type of the device
Count	DWORD	out	Number of devices which responded.
Data	NET_DeviceInfo[Count]	out	NET_DeviceInfo structure; typedef struct NET_DeviceInfo { BYTE MAC[6]; //MAC Address BYTE IP[4]; //IP Address BYTE VER; //Version BYTE LEN; //Length of the name of devices BYTE NAME[16]; //name of devices };
Length	DWORD	out	Length of data
Return			
Return	DWORD	out	Return value: 0 for success

NET_DEVICE_CONFIG Structure

```
typedef struct NET_DEVICE_CONFIG
{
    DEVICEHW_CONFIG    HWCfg;                /*Configuration info of devices*/
    DEVICEPORT_CONFIG  PortCfg[2];           /*Configuration info of children devices*/
}
```

DEVICEHW_CONFIG Structure

```
typedef struct DEVICEHW_CONFIG
{
    UCHAR    bDevType;                /* Type of the device (please refer to Table of type of devices) */
    UCHAR    bAuxDevType;             /* Aux type of the device */
    UCHAR    bIndex;                  /* Index of the device */
    UCHAR    bDevHardwareVer;         /* Hardware version of the device */
    UCHAR    bDevSoftwareVer;         /* Software version of the device */
    UCHAR    szModulename[21];        /* name of the module */
    UCHAR    bDevMAC[6];              /* MAC address of the module */
    UCHAR    bDevIP[4];               /* IP address of the module */
    UCHAR    bDevGWIP[4];             /* Gateway IP address of the module */
    UCHAR    bDevIPMask[4];           /* IP Mask of the module */
    UCHAR    bDhcpEnable;             /* DHCP Enable. DHCP Status,1:Enabled, 0: Disabled*/
    UCHAR    breserved[29];           /* Reserved property*/
};
```

DEVICEPORT_CONFIG Structure

```
typedef struct DEVICEPORT_CONFIG
```

```
{  
    UCHAR  bIndex;           /* Index of the port */  
    UCHAR  bPortEn;          /* Is port Enabled: 1: Enabled ; 0: Disabled */  
    UCHAR  bNetMode;          /* Net mode: 0: TCP SERVER;1: TCP CLIENT; 2: UDP SERVER 3: UDP CLIENT; */  
    UCHAR  bRandSportFlag;    /* Is local port random (TCP CLIENT), 1: random 0: not random*/  
    USHORT wNetPort;          /* Port */  
    UCHAR  bDesIP[4];         /* destination IP address */  
    USHORT wDesPort;          /* Local port (as TCP SERVER) */  
    ULONG  dBaudRate;         /* Baud rate of serial port: 300---921600bps */  
    UCHAR  bDataSize;         /* Bits which indicate size of data on serial port: bit5--bit8 */  
    UCHAR  bStopBits;         /* Stop bits on serial port: 1: one stop bit; 2: 2 stop bits */  
    UCHAR  bParity;           /* Check bits: 0: Odd parity; 1: Even parity; 2: MARK bit is 1; 3 SPACE(set bits to 0); */  
    UCHAR  bPHYChangeHandle; /* what should be done to socket when PHY disconnects, 1: close socket 2, nothing*/  
    ULONG  dRxPktlength;      /* Package size of Rx data on serial port. (1024 at max) */  
    ULONG  dRxPktTimeout;     /* Timeout of Rx data on serial port (/10ms), 0 means infinite timeout */  
    UCHAR  bReConnectCnt;     /* Max times of trying to connect to TCP SERVER (when working as TCP CLIENT)*/  
    UCHAR  bResetCtrl;        /* Reset control on serial port: 0: preserve data buffer; 1: flush data buffer on serial port  
when connected */  
    UCHAR  bDNSFlag;          /* Is DNS enabled, 1: Enabled. 2: Disabled*/  
    UCHAR  szDomainname[20]; /* domain name*/  
    UCHAR  bDNSHostIP[4];     /* DNS host*/  
    USHORT wDNSHostPort;      /* DNS port*/  
    UCHAR  breserved[8];      /* Reserved property */  
};
```

NetCfg_GetInfo

Function			
<i>DWORD</i> NetCfg_GetInfo(<i>BYTE</i> DeviceType, <i>LPBYTE</i> MAC, <i>LPBYTE</i> Data, <i>LPDWORD</i> Length);			
Remarks			
Get NetCfg of device			
Parameter	Data Type		Description
DeviceType	BYTE	in	Type of device
MAC	Byte[]	in	MAC address
Data	NET_DEVICE_CONFIG	out	Data Structure
Length	DWORD	out	length of data
Return			
Return	DWORD	out	Return value: 0 for success

NetCfg_SetInfo

Function			
<i>DWORD</i> NetCfg_SetInfo(<i>BYTE</i> DeviceType, <i>LPBYTE</i> LocalMAC, <i>LPBYTE</i> DevMAC, <i>LPBYTE</i> Data, <i>DWORD</i> Length);			
Remarks			
Set NetCfg of device			
Parameter	Data Type		Description
DeviceType	BYTE	in	Type of device
LocalMAC	Byte[]	in	Local MAC address
DevMAC	Byte[]	in	MAC address of the device
Data	NET_DEVICE_CONFIG	out	Data Structure
Length	DWORD	out	length of data
Return			
Return	DWORD	out	Return value: 0 for success

NetCfg_FactoryReset

Function			
<i>DWORD</i> NetCfg_FactoryReset(<i>BYTE</i> DeviceType, <i>LPBYTE</i> MAC);			
Remarks			
Reset configuration			
Parameter	Data Type		Description
DeviceType	BYTE	in	Type of device
MAC	Byte[]	in	MAC address
Return			
Return	DWORD	out	Return value: 0 for success