Catalogue

[Interface Functions 3](#_Toc18938025)

[Callback Function Definition 3](#_Toc18938026)

[Establish Connection with Device 4](#_Toc18938027)

[Disconnect From Device 5](#_Toc18938028)

[Get Informations of the Reader Module 6](#_Toc18938029)

[Single Polling Read 7](#_Toc18938030)

[Set “Select” Parameters 8](#_Toc18938031)

[Set “Select” Mode 9](#_Toc18938032)

[Read RFID Data Storage 10](#_Toc18938033)

[Write RFID Data Storage 11](#_Toc18938034)

[Lock RFID Data Storage 12](#_Toc18938035)

[Kill RFID 13](#_Toc18938036)

[Get Query Parameters 14](#_Toc18938037)

[Set Query Parameters 15](#_Toc18938038)

[Set Working Region 16](#_Toc18938039)

[Set Working Channel 17](#_Toc18938040)

[Get Working Channel 18](#_Toc18938041)

[Set FHSS 19](#_Toc18938042)

[Set Transmitting Power 20](#_Toc18938043)

[Get Transmitting Power 21](#_Toc18938044)

[Set Continuous Carrier Wave 22](#_Toc18938045)

[Set Modem Parameters 23](#_Toc18938046)

[Get Modem Parameters 24](#_Toc18938047)

[Scan Jammer on RF Input 25](#_Toc18938048)

[Scan RSSI 26](#_Toc18938049)

[NXP ReadProtect/Reset ReadProtect 27](#_Toc18938050)

[NXP Change EAS 28](#_Toc18938051)

[NXP EAS\_Alarm 29](#_Toc18938052)

[NXP ChangeConfig 30](#_Toc18938053)

[Impinj Monza QT 31](#_Toc18938054)

[NetCfg\_Open 32](#_Toc18938055)

[NetCfg\_Close 33](#_Toc18938056)

[NetCfg\_SearchForDevices 34](#_Toc18938057)

[NET\_DEVICE\_CONFIG Structure 35](#_Toc18938058)

[DEVICEHW\_CONFIG Structure 36](#_Toc18938059)

[DEVICEPORT\_CONFIG Structure 37](#_Toc18938060)

[NetCfg\_GetInfo 38](#_Toc18938061)

[NetCfg\_SetInfo 39](#_Toc18938062)

[NetCfg\_FactoryReset 40](#_Toc18938063)

# 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** | **DataType** |  | **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** | **DataType** |  | **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 attatched 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** | **DataType** |  | **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** | **DataType** |  | **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** | | | |
| DWORD SetRegion(BYTE Region); | | | |
| **Remarks** | | | |
| Set work region of the reader. | | | |
| **Parameters** | **DataType** |  | **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** | **DataType** |  | **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** | **DataType** |  | **Description** |
| CH\_Index | Byte | in | Calculation formula，Freq\_CH is the frequency of the channel：  China 900MHz  Freq\_CH = CH\_Index \* 0.25M + 920.125M  China 800MHz  Freq\_CH = CH\_Index \* 0.25M + 840.125M  USA  Freq\_CH = CH\_Index \* 0.5M + 902.25M  Europe  Freq\_CH = CH\_Index \* 0.2M + 865.1M  South Korea  Freq\_CH = CH\_Index \* 0.2M + 917.1M |
| **Return** | | | |
| Return | DWORD | out | Return value: 0 for success |

## Set FHSS

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | | | |
| DWORD SetFhss(BOOL Param); | | | |
| **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** | **DataType** |  | **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** | **DataType** |  | **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 threshhold. | | | |
| **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 threshhold 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 threshhold 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** | | | |
| *DWORD* GetModemPara(*LPBYTE* Mixer\_G, *LPBYTE* IF\_G, *LPDWORD* Thrd); | | | |
| **Remarks** | | | |
| Get modem parameters of current reader, including Mixer Gain, IF AMP Gain and Demodulation threshhold. | | | |
| **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 threshhold 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 threshhold 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** | | | |
| *DWORD* ScanJammer(*LPBYTE* CH\_L, *LPBYTE* CH\_H, *LPBYTE* JMR); | | | |
| **Remarks** | | | |
| Scan jammer on RF input to check jammer strength on each channel. | | | |
| **Parameters** | **DataType** |  | **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** | | | |
| *DWORD* ScanRSSI(*LPBYTE* CH\_L, *LPBYTE* CH\_H, *LPBYTE* JMR); | | | |
| **Remarks** | | | |
| Scan RSSI on RF input to check if any reader is currently working. | | | |
| **Parameters** | **DataType** |  | **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 为-70dBm)  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** | | | |
| *DWORD* NxpReadProtect(*LPBYTE* AccessPassword, *BYTE* Protect, *LPBYTE* PC, *LPBYTE* 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** | **DataType** |  | **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** | | | |
| *DWORD* NxpChangeEas(*LPBYTE* AccessPassword, *BYTE* Protect, *LPBYTE* PC, *LPBYTE* EPC); | | | |
| **Remarks** | | | |
| 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.  Select command must be sent before this operation to specify single RFID. | | | |
| **Parameters** | **DataType** |  | **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** | | | |
| *DWORD* NxpEasAlarm(*LPBYTE* 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** | **DataType** |  | **Description** |
| EASAlarmCode | Byte[ ] | out | 64bits EAS-Alarm code |
| **Return** | | | |
| Return | DWORD | out | Return value: 0 for success |

## NXP ChangeConfig

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | | | |
| *DWORD* NxpChangeConfig(*LPBYTE* AccessPassword, *LPBYTE* Config, *LPBYTE* PC, *LPBYTE* 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** | **DataType** |  | **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** | | | |
| *DWORD* ImpinjMonzaQT(*LPBYTE* AccessPassword, *BYTE* RW, *BYTE* Persistence, *BYTE* Payload, *LPBYTE* PC, *LPBYTE* EPC, *LPDWORD* QTControl); | | | |
| **Remarks** | | | |
| 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.  Select command must be sent before this operation to specify single RFID. | | | |
| **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** | | | |
| *DWORD* NetCfg\_Open(*LPSTR* 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** | | | |
| *DWORD* NetCfg\_Close(); | | | |
| **Remarks** | | | |
| Close NetCfg configurations. | | | |
| **Parameter** | **DataType** |  | **Description** |
| **Return** | | | |
| Return | DWORD | out | Return value: 0 for success |

## NetCfg\_SearchForDevices

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | | | |
| *DWORD* NetCfg\_SearchForDevices(*BYTE* DeviceType, *LPDWORD* Count, *LPBYTE* Data, *LPDWORD* Length); | | | |
| **Remarks** | | | |
| Search for NetCfg devices. | | | |
| **Parameter** | **DataType** |  | **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** | | | |
| *DWORD* NetCfg\_GetInfo(*BYTE* DeviceType, *LPBYTE* MAC, *LPBYTE* Data, *LPDWORD* Length); | | | |
| **Remarks** | | | |
| Get NetCfg of device | | | |
| **Parameter** | **DataType** |  | **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** | | | |
| *DWORD* NetCfg\_SetInfo(*BYTE* DeviceType, *LPBYTE* LocalMAC, *LPBYTE* DevMAC, *LPBYTE* Data, *DWORD* Length); | | | |
| **Remarks** | | | |
| Set NetCfg of device | | | |
| **Parameter** | **DataType** |  | **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** | | | |
| *DWORD* NetCfg\_FactoryReset(*BYTE* DeviceType, *LPBYTE* MAC); | | | |
| **Remarks** | | | |
| Reset configuration | | | |
| **Parameter** | **DataType** |  | **Description** |
| DeviceType | BYTE | in | Type of device |
| MAC | Byte[ ] | in | MAC address |
| **Return** | | | |
| Return | DWORD | out | Return value: 0 for success |