AsReader P51N

SDK Reference Guide V1.2

Asterisk, Inc.

Revision History

Version	Description	Date
V1.0	Initial version	2019/03/29
V1.1	1， updates appendix2 2， Adds function: SetStaticIP	2019/6/17
V1.2	Adds function GetSdkVersion	2019/6/21

Contents

System Requirements

Windows version：

Development software: Visual Studio 2012+

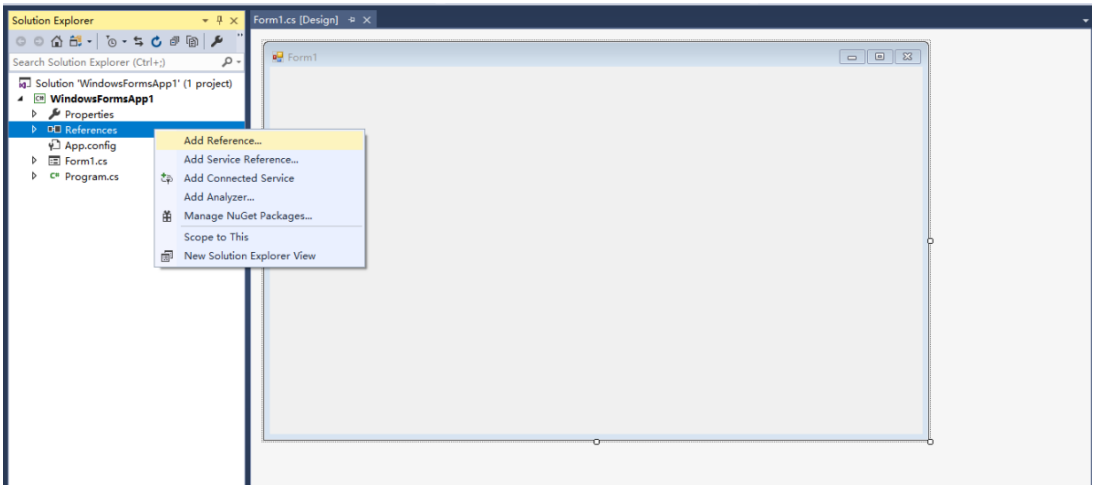
Development language: C#

1 SDK Usage

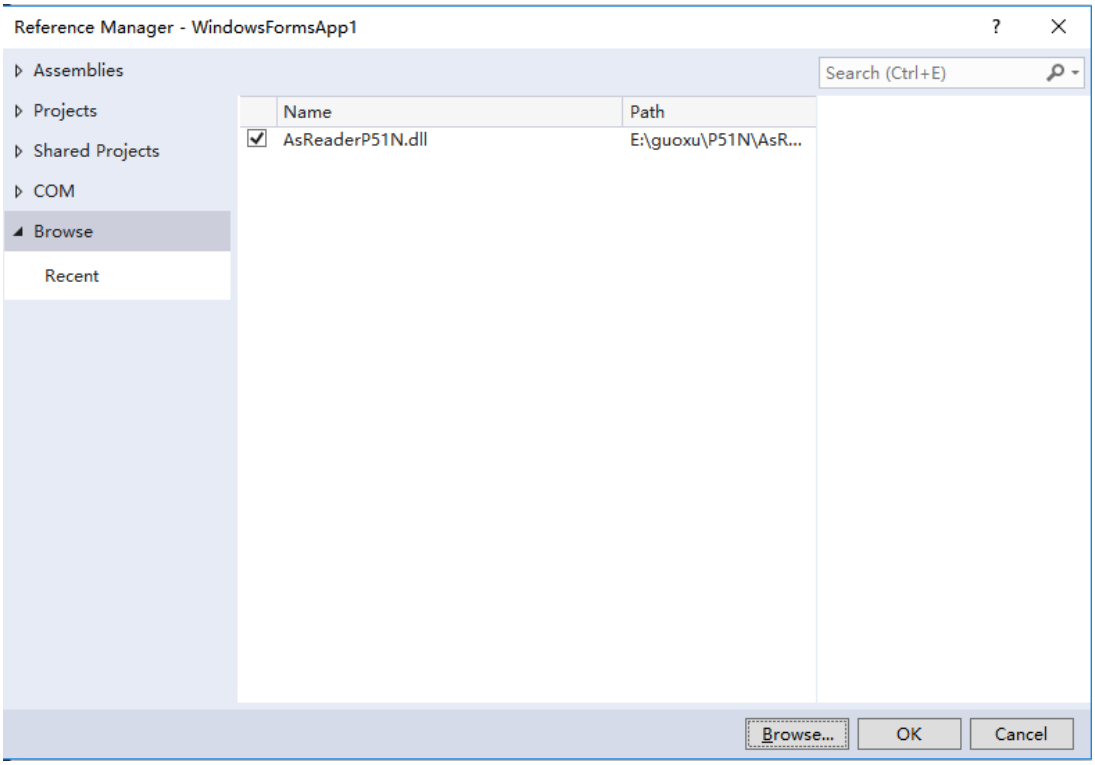
1.1 Add SDK

1. New project —— a Windows forms application.
   Copy asreaderp51n.dll and asreader.dll into the project folder.

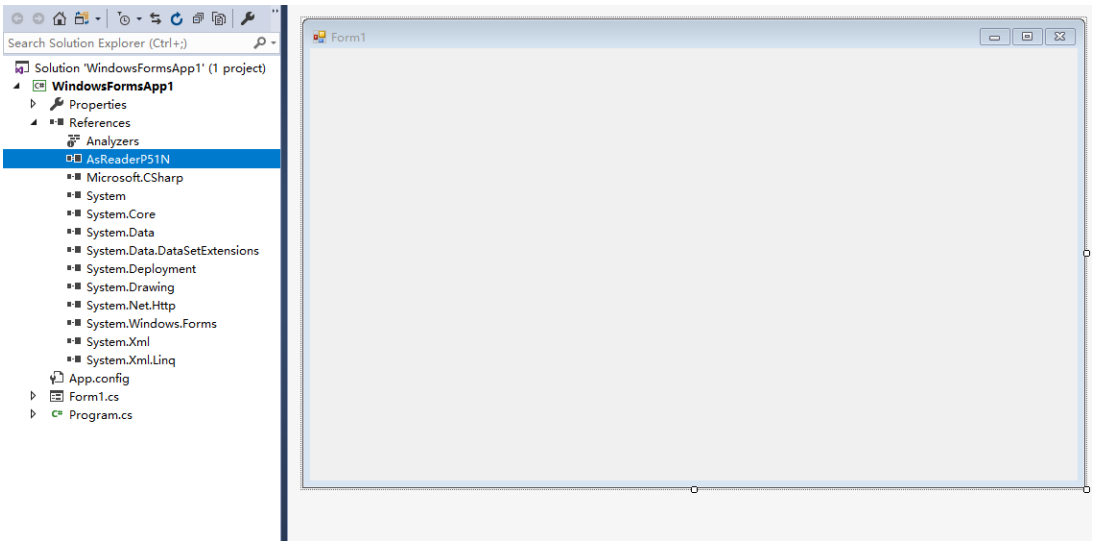
2. Add a reference to the project ——AsReaderP51N.dll.
   Right-click "reference" to add a reference.



3. Click “Browse”, check “AsreaderP51N.dll”, and click “OK”.



4. The page looks like the following figure after completing the addition.

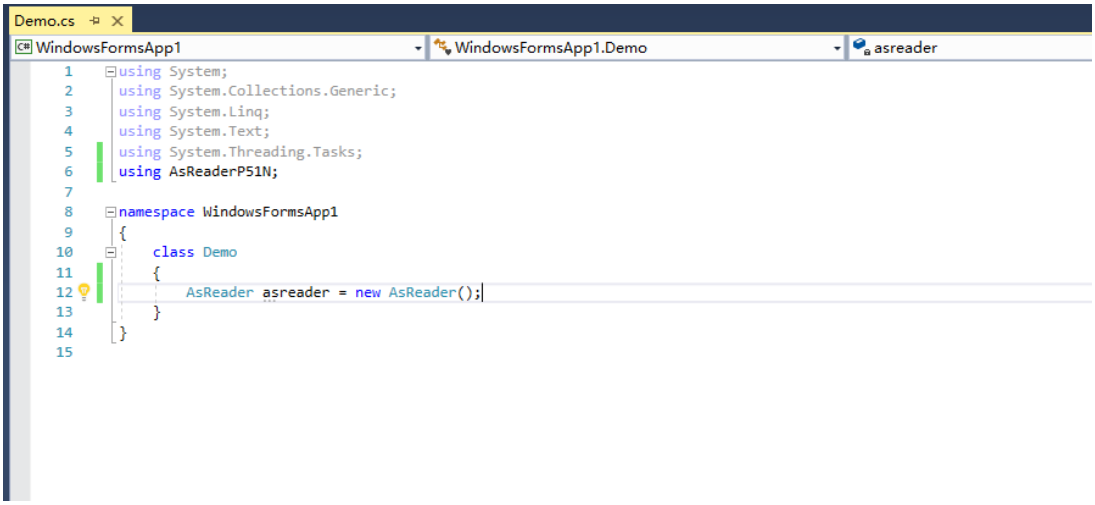


1.2 SDK Import

1. Reference namespace

```
using AsReaderP51N;
```

2. Instantiate objects.



3. Run the ConnectWithVCP function (Refer to 2.1.1) to Connect to the AsReader.
   Note: if successful, return 0x0; If it fails, return 0x2000002.

```
asreader.ConnectWithVCP("COM1");
```

2 API Description

2.1 AsReader Class

The AsReader Class provides API interfaces such as RFID inventory, read tags, write tags, lock tags, and so on.

2.1.1 ConnectWithVCP

Function	UInt32 ConnectWithVCP(string comPort)		
Parameter	**IN/OUT**	**Type**	**Description**
comPort	IN	string	serial number
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Open the COM to established the connection between the host and device， then initializing all the device and registering all the callback handler.			
For example: ConnectWithVCP(“COM1”);			

2.1.2 ConnectWithTCP

Function	UInt32 ConnectWithTCP(string ipAddress,string tcpPort)		
Parameter	**IN/OUT**	**Type**	**Description**
ipAddress	IN	string	IP Address of AsReader
tcpPort	IN	string	Port number of AsReader
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Open the TCP/IP to established the connection between the host and device， then initializing all the device and registering all the callback handler.			
For example: ConnectWithTCP(“192.168.1.100”,“9600”);			

Note: TCP port number: 5000. UDP port number: 50001.

2.1.3 Disconnect

Function	UInt32 Disconnect()		
Parameter	**IN/OUT**	**Type**	**Description**
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Disconnect the terminal from the AsReader device and reset the AsReader device.			
For example: Disconnect the terminal from the AsReader device. Disconnect();			

2.1.4 StartInventory

Function	UInt32 StartInventory(bool epc,bool tid,bool pc,bool rssi,bool phase,bool channel,bool antenna)		
Parameter	**IN/OUT**	**Type**	**Description**
epc	IN	bool	true: shown; false: not shown.
tid	IN	bool	true: shown; false: not shown.
pc	IN	bool	true: shown; false: not shown.
rssi	IN	bool	true: shown; false: not shown.
phase	IN	bool	true: shown; false: not shown.
channel	IN	bool	true: shown; false: not shown.
antenna	IN	bool	true: shown; false: not shown.
Return Value	OUT	UInt32	Refer to Appendix I.
Description: AsReader starts to inventory tags.			
For example: When inventorying tags, EPC and TID data of tags are displayed, but PC, RSSI, phase, channel, antenna data of tags are not displayed. StartInventory(true,true,false,false,false,false,false);			

2.1.5 StopInventory

Function	UInt32 StopInventory()

Parameter	IN/OUT	Type	Description
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Stop the IRI device from inventorying tags.			
For example: Stop inventorying tags. StopInventory();			

2.1.6 SetSelectMask

Function	UInt32 SetSelectMask(MemBankType memBank,TargetType target,ActionType action unit startAddressWord,byte[] selectMask)		
Parameter	**IN/OUT**	**Type**	**Description**
memBank	IN	enum	The memory bank of the tag to be selection-mask compared. Refer to 2.2.1.7.
target	IN	enum	The session for the Selection Mask to be applied. Refer to 2.2.1.5
action	IN	enum	Action after tags be flagged. Refer to 2.2.1.6
startAddressWord	IN	uint	Offset of the selected tag bank, which is the starting address. (unit: word)
selectMask	IN	byte	Value of Selection Mask. [0-16]. (unit: word)
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Set the select parameter, so that in the case of counting multiple tags, the select parameter can be used for inventorying/reading/writing/locking and other operations on specific tags.			
For example: Select the label that meets the following criteria： Bank: EPC; Session: SESSION_S0; Action: ACTION_ASLINVA_DSLINVB; Offset: 2; Mask length: 6; Mask contents: byte[]selectMask={0x12,0x34,0x56,0x78,0x12,0x34,0x56x,0x78,0x12,0x34,0x56,0x78}; SetSelectMask(Mem_EPC,SESSION_S0,ACTION_ASLINVA_DSLINVB,0x02,selectMask);			

2.1.7 WriteMemory

Function	UInt32 WriteMemory(MemBankType memBank,unit startAddressWord,

	unit accessPassword, byte[] writeData,byte[] epcData)		
Parameter	**IN/OUT**	**Type**	**Description**
memBank	IN	enum	Sets the tag memory bank to be written to. Refer to 2.2.1.7
startAddressWord	IN	uint	Sets start address of data. (unit: word)
accessPassword	IN	uint	The access password for the target tag. (0 if no password)
writeData	IN	byte	The value to be written.
epcData	IN	byte	EPC number of the target tag.
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Target a tag with epcData and writes data to the target bank of the tag. The length of data to be written should not exceed 32 Words(64 Bytes).			
For example: Write tags. Target bank: EPC; Offset: 2; Access password: 0x12345678; Value: byte[] writedata = {0x12,0x34}; EPC number of target tag: byte[] epcData= {0x12,0x34,0x56,0x78,0x12,0x34, 0x56x,0x78,0x12,0x34,0x56,0x78}; WriteMemory(Mem_EPC,0x02,0x12345678,writeData,epcData);			

2.1.8 ReadMemory

Function	UInt32 ReadMemory(MemBankType memBank,unit startAddressWord, unit lengthWord, unit accessPassword,byte[] epcData)		
Parameter	**IN/OUT**	**Type**	**Description**
memBank	IN	enum	The tag memory bank to be read. Refer to 2.2.1.7
startAddressWord	IN	uint	Offset of target tag bank. (unit: word)
lengthWord	IN	uint	Length of the data to be read. (unit: word)
accessPassword	IN	uint	The access password for the target tag. (0 if no password)
epcData	IN	byte	EPC number of the target tag.
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Target a tag with epcData and reads data of the target bank. The length of data to be read should not exceed 32 Words(64 Bytes).			
For example: Read tags. Target bank: EPC;			

Offset: 2; Length to read: 2; Access password: 0x12345678; EPC number of target tag: byte[] epcData= {0x12,0x34,0x56,0x78,0x12,0x34, 0x56x,0x78,0x12,0x34,0x56,0x78}; ReadMemory(Mem_EPC,0x02,0x02,0x12345678,epcData);

2.1.9 Kill

Function	UInt32 Kill(unit KillPassword,byte[] epcData)		
Parameter	**IN/OUT**	**Type**	**Description**
KillPassword	IN	uint	Kill password of the target tag.
epcData	IN	byte	EPC number of the target tag.
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Before killing the tag, the kill password needs to be written to the RESERVED bank, starting with offset 00 and writing two words. **Note:** Kill operation is irreversible once executed.			
For example: Kill password is 0x12345678. EPC number of target tag: byte[] epcData= {0x12,0x34,0x56,0x78,0x12,0x34, 0x56x,0x78,0x12,0x34,0x56,0x78}; Kill(0x12345678,epcData);			

2.1.10 LockMemory

Function	UInt32 LockMemory(TagMask tagMask,TagAction tagAction, unit accessPassword,byte[] epcData)		
Parameter	**IN/OUT**	**Type**	**Description**
tagMask	IN	TagMask	Mask for the Lock operation. Refer to Appendix IV.
tagAction	IN	TagAction	Action for the Lock operation. Refer to Appendix IV.
accessPassword	IN	uint	The access password for the target tag. (0 if no password)
epcData	IN	byte	EPC number of the target tag.
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to lock (Lock)/permanently lock(PermaLock)/unlock(Unlock)or permanently unlock(PermaUnlock)the target bank of the tag.			

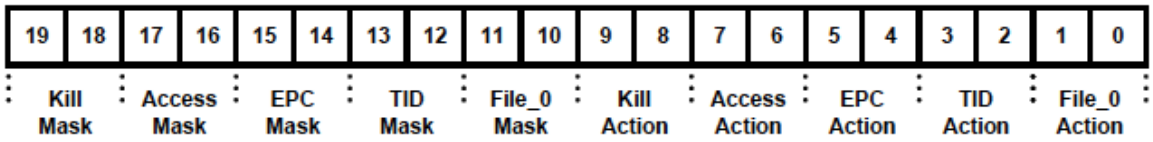
Before locking the tag, the access password needs to be written to the RESERVED bank, starting with offset 0X20 and writing two words.

The high 4 bits of the Lock operation parameter target are reserved bits. The remaining 20 bits are the Payload of the Lock operation, including the Mask and the Action, 10 bits each from high to low. Only Actions with Mask bit equals 1 are valid.

Each data area has 2bits for Action.00 01 10 11 indicate unlock, permanently unlock, lock, and permanently lock.

The meanings of each bit of Mask and Action are shown in the following table:

Lock-Command Payload



Masks and Associated Action Fields

	Kill pwd		Access pwd		EPC memory		TID memory		File_0 memory	
	19	18	17	16	15	14	13	12	11	10
Mask	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write
	9	8	7	6	5	4	3	2	1	0
Action	pwd read/ write	perma lock	pwd read/ write	perma lock	pwd write	perma lock	pwd write	perma lock	pwd write	perma lock

For example: Lock the EPC bank of the tag.

Access password of the target tag is 12345678.

EPC number of target tag: byte[] epcData= {0x12,0x34,0x56,0x78,0x12,0x34,

0x56x,0x78,0x12,0x34,0x56,0x78};

LockMemory(0x008020,0x12345678,epcData);

2.1.11 GetFirmwareVersion

Function	UInt32 GetFirmwareVersion(ref string firmwareVersion)		
Parameter	**IN/OUT**	**Type**	**Description**
firmwareVersion	OUT	string	FW version of the AsReader
Return Value	OUT	UInt32	Refer to Appendix I.
Description: To get the FW version of the AsReader.			
For example: To get the FW version of the AsReader. GetFirmVersion(firmwareVersion);			

2.1.12 SetRegion

Function	UInt32 SetRegion(RegionType region)		
Parameter	**IN/OUT**	**Type**	**Description**
region	IN	enum	Country or region currently located. (Refer to 2.2.1.1)
Return Value	OUT	UInt32	Refer to Appendix I.
Description: To set the country or region currently located.			
For example: Set the country or region currently located to be REGION_FCC_PART_15_247. SetRegion(REGION_FCC_PART_15_247);			

2.1.13 GetRegion

Function	UInt32 GetRegion(ref RegionType region)		
Parameter	**IN/OUT**	**Type**	**Description**
region	OUT	enum	Country or region currently located. (Refer to 2.2.1.1)
Return Value	OUT	UInt32	Refer to Appendix I.
Description: To get the country or region currently located.			
For example: To get the country or region currently located. GetRegion(Value);			

2.1.14 SetValue

Function	UInt32 SetValue(Unit key, Unit value)		
Parameter	**IN/OUT**	**Type**	**Description**
key	IN	uint	Parameters for the AsReader device. (Refer to 2.2.1.8)
value	IN	uint	The value of this parameter
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to set parameters for the AsReader device.			
For example: Set TX power to 30dBm. SetValue(E_IPJ_KEY_ANTENNA_TX_POWER,30);			

2.1.15 GetValue

Function	UInt32 GetValue(unit key, ref unit value)		
Parameter	**IN/OUT**	**Type**	**Description**
key	IN	uint	parameters for the AsReader device. (Refer to 2.2.1.8)
value	OUT	uint	The value of this parameter
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to get parameters for the AsReader device.			
For example: To get the TX power. GetValue(E_IPJ_KEY_ANTENNA_TX_POWER,value);			

2.1.16 GetAntennaPorts

Function	GetAntennaPorts(ref bool ant1, ref bool ant2, ref bool ant3, ref bool ant4)		
Parameter	**IN/OUT**	**Type**	**Description**
ant1	OUT	bool	The state of antenna 1 (enabled or not enabled). [true, false]
ant2	OUT	bool	The state of antenna 2 (enabled or not enabled). [true, false]
ant3	OUT	bool	The state of antenna 3 (enabled or not enabled). [true, false]
ant4	OUT	bool	The state of antenna 4 (enabled or not enabled). [true, false]
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to get the status (enabled or not enabled) of all antenna ports.			
For example: To get the status (enabled or not enabled) of all antenna ports. GetAntennaPorts(ant1,ant2,ant3,ant4);			

2.1.17 SetAntennaPorts

Function	SetAntennaPorts(bool ant1,bool ant2,bool ant3, bool ant4)		
Parameter	**IN/OUT**	**Type**	**Description**
ant1	IN	bool	The state of antenna 1 (enabled or not enabled). [true, false]
ant2	IN	bool	The state of antenna 2 (enabled or not enabled).

			[true, false]
ant3	IN	bool	The state of antenna 3 (enabled or not enabled). [true, false]
ant4	IN	bool	The state of antenna 4 (enabled or not enabled). [true, false]
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to set the status (enabled or not enabled) of all antenna ports.			
For example: Set the antenna1 enabled and the antenna2/3/4 not enabled. SetAntennaPorts(true,false,false,false);			

2.1.18 GetTxPower

Function	UInt32 GetTxPower(ref uint power)		
Parameter	**IN/OUT**	**Type**	**Description**
power	OUT	uint	Value of TX power. [0-30]
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to get the TX power for the AsReader device.			
For example: To get the TX power for the AsReader device. GetTxPower(power);			

2.1.19 SetTxPower

Function	UInt32 SetTxPower(uint txPower)		
Parameter	**IN/OUT**	**Type**	**Description**
txPower	IN	uint	Value of TX power. [0-30]
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to set the TX power for the AsReader device.			
For example: Set TX power to 30dBm. SetTxPower(30);			

2.1.20 SetRFMode

Function	UInt32 SetRFMode(RFModeType mode)		
Parameter	**IN/OUT**	**Type**	**Description**

mode	IN	enum	RFID Working mode. (Refer to 2.2.1.2)
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to set the RFID working mode of the AsReader.			
For example: Set the RFID working mode of the AsReader to RFMODE0(Auto-Mode). SetRFMode(RFMODE0);			

2.1.21 GetRFMode

Function	UInt32 GetRFMode(ref RFModeType mode)		
Parameter	**IN/OUT**	**Type**	**Description**
mode	OUT	enum	RFID Working mode. (Refer to 2.2.1.2)
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to get the current RFID working mode of the AsReader.			
For example: To get the current RFID working mode of the AsReader. GetRFMode(mode);			

2.1.22 SetSession

Function	UInt32 SetSession(SessionType session)		
Parameter	**IN/OUT**	**Type**	**Description**
session	IN	enum	RFID Session. (Refer to 2.2.1.3)
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to set RFID Session.			
For example: Set RFID Session to S0. SetSession(SESSION_S0);			

2.1.23 GetSession

Function	UInt32 GetSession(ref SessionType session)		
Parameter	**IN/OUT**	**Type**	**Description**
session	OUT	enum	RFID Session. (Refer to 2.2.1.3)
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to get the current RFID Session.			

For example: Get the current RFID Session. GetSession(session);

2.1.24 SetSearchMode

Function	UInt32 SetSearchMode(SearchModeType mode)		
Parameter	**IN/OUT**	**Type**	**Description**
mode	IN	enum	Search mode of RFID. (Refer to 2.2.1.4)
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to set the search mode of RFID			
For example: Set the search mode of RFID to INVENTORY_SEARCH_MODE_AUTO_SEARCH. SetSearchMode(INVENTORY_SEARCH_MODE_AUTO_SEARCH);			

2.1.25 GetSearchMode

Function	UInt32 GetSearchMode(ref SearchModeType mode)		
Parameter	**IN/OUT**	**Type**	**Description**
mode	OUT	enum	Search mode of RFID. (Refer to 2.2.1.4)
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Used to get the current search mode of RFID.			
For example: Get the current search mode of RFID. GetSession(mode);			

2.1.26 SetCWOn

Function	UInt32 SetCWOn()		
Parameter	**IN/OUT**	**Type**	**Description**
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Open Carrier Wave (CW). Carrier Wave(CW): Carrier wave describes a reader transmitted signal at the configured channel or frequency without any forward link modulation.			
For example: Open Carrier Wave (CW). SetCWOn();			

2.1.27 SetCWOff

Function	UInt32 SetCWOff()		
Parameter	**IN/OUT**	**Type**	**Description**
Return Value	OUT	UInt32	Refer to Appendix I.
Description: Close Carrier Wave (CW). Carrier Wave(CW): Carrier wave describes a reader transmitted signal at the configured channel or frequency without any forward link modulation.			
For example: Close Carrier Wave (CW). SetCWOff();			

2.1.28 GetDeviceInformation

Function	void GetDeviceInformation(DeviceInformation deviceInformation)		
Parameter	**IN/OUT**	**Type**	**Description**
deviceInformation	OUT	DeviceInformation	Device information. Refer to Appendix III.
Return Value	OUT	void	
Description: Used to get the device information.			
For example: Get the device information. GetDeviceInformation(mode);			

2.1.29 SetStaticIP

Function	bool SetStaticIP(string ip,string subnet,string gateway,string dnsServer)		
Parameter	**IN/OUT**	**Type**	**Description**
ip	IN	string	IP address
subnet	IN	string	Subnet mask
gateway	IN	string	Gateway
dnsServer	IN	string	DNS server
Return Value	OUT	bool	
Description: Sets a static IP address.			
For example: SetStaticIP(“192.168.3.1”,”255.255.255.0”,”192.168.3.1”,”192.168.3.1”);			

2.1.30 GetSdkVersion

Function	void GetSdkVersion(ref string sdkVersion)		
Parameter	**IN/OUT**	**Type**	**Description**
sdkVersion	OUT	string	SDK version
Return Value	OUT	Void	
Description: This function is used to get the SDK version.			
For example: string sdkVersion; GetSdkVersion(sdkVersion);			

2.1.31 SetDelegate

Function	void SetDelegate(CallBackReadTagData readTagData,CallBackError errorString)		
Parameter	**IN/OUT**	**Type**	**Description**
readTagData	IN	CallBackReadTagData	The delegate function for data processing.
errorString	IN	CallBackError	The delegate function used to execute the error output
Return Value	OUT	void	
Description: Used to set the delegate functions.			

For example:
CallBackReadTagData Rec = null;
CallBackError Rec1 = null;
void test(InventoryResult ReadTagStruct);
void test1(string Error);
Rec = test;
Rec1 = test1;
SetDelegate(Rec,Rec1);			

Note: Defines the delegate function for data processing and the delegate function for error output.
public delegate void CallBackReadTagData(InventoryResult tagcallbackdata);
public delegate void CallBackErrorString(string error);
InventoryResult. Refer to Appendix II.			

2.2 Types Class

The Types class is used to define the region type/RFID mode type/session type/search mode type/Selection Mask target session type/action type/memory bank type and key value type.

2.2.1 Enumeration Types

2.2.1.1 RegionType

Region	Value
REGION_FCC_PART_15_247	0x0
REGION_HONG_KONG_920_925_MHZ	0x3
REGION_TAIWAN_922_928_MHZ	0x4
REGION_ETSI_EN_302_208_V1_4_1	0x7
REGION_KOREA_917_921_MHZ	0x8
REGION_MALAYSIA_919_923_MHZ	0x9
REGION_CHINA_920_925_MHZ	0xA
REGION_SOUTH_AFRICA_915_919_MHZ	0xC
REGION_BRAZIL_902_907_AND_915_928_MHZ	0xD
REGION_THAILAND_920_925_MHZ	0xE
REGION_SINGAPORE_920_925_MHZ	0xF
REGION_AUSTRALIA_920_926_MHZ	0x10
REGION_INDIA_865_867_MHZ	0x11
REGION_URUGUAY_916_928_MHZ	0x12
REGION_VIETNAM_920_925_MHZ	0x13
REGION_ISRAEL_915_917_MHZ	0x14
REGION_PHILIPPINES_918_920_MHZ	0x15
REGION_INDONESIA_923_925_MHZ	0x17
REGION_NEW_ZEALAND_921P5_928_MHZ	0x18
REGION_JAPAN_916_921_MHZ_NO_LBT	0x19
REGION_PERU_916_928_MHZ	0x1A
REGION_RUSSIA_916_921_MHZ	0x1B
REGION_CUSTOM	0x100

2.2.1.2 RfModeType

RFID Mode	Value	Description
RFMODE0	0x0	Auto (Defaults to mode 1)
RFMODE1	0x1	Dense Reader Mode profile for FCC operation
RFMODE2	0x2	Dense Reader Mode profile for ETSI operation
RFMODE3	0x3	Very fast mode with reduced sensitivity and noise immunity
RFMODE4	0x4	Very sensitive mode with reduced speed and reduced noise immunity

2.2.1.3 SessionType

Session	Value
SESSION_S0	0x0
SESSION_S1	0x1
SESSION_S2	0x2
SESSION_S3	0x3

2.2.1.4 SearchModeType

Search Mode	Value
INVENTORY_SEARCH_MODE_AUTO_SEARCH	0x0
INVENTORY_SEARCH_MODE_DUAL_TARGET	0x1
INVENTORY_SEARCH_MODE_SINGLE_TARGET_A_TO_B	0x2
INVENTORY_SEARCH_MODE_SINGLE_TARGET_B_TO_A	0x3

2.2.1.5 TargetType

The target session for the Selection Mask is applied.

Target	Value	Description
SESSION_S0	0x0	inventoried S0
SESSION_S1	0x1	inventoried S1
SESSION_S2	0x2	inventoried S2
SESSION_S3	0x3	inventoried S3
SL_FLAG	0x4	Selection Flags

2.2.1.6 ActionType

ActionType enumeration type that returns what happens to the Session and Session Flag of a tag that matches and does not match with a Selection Mask when using the Selection Mask function.

Action	Value	Description
ACTION_ASLINVA_DSLINVB	0x0	Tag Matching: assert SL or inventoried → A Tag Not-Matching: retract SL or inventoried → B
ACTION_ASLINVA_NOTHING	0x1	Tag Matching: assert SL or inventoried → A Tag Not-Matching: do nothing
ACTION_NOTHING_DSLINVB	0x2	Tag Matching: do nothing Tag Not-Matching: retract SL or inventoried → B
ACTION_NSLINVS_NOTHING	0x3	Tag Matching: negate SL or (A → B, B → A) Tag Not-Matching: do nothing
ACTION_DSLINVB_ASLINVA	0x4	Tag Matching: retract SL or inventoried → B Tag Not-Matching: assert SL or inventoried → A
ACTION_DSLINVB_NOTHING	0x5	Tag Matching: retract SL or inventoried → B Tag Not-Matching: do nothing
ACTION_NOTHING_ASLINVA	0x6	Tag Matching: do nothing Tag Not-Matching: assert SL or inventoried → A
ACTION_NOTHING_NSLINVS	0x7	Tag Matching: do nothing Tag Not-Matching: negate SL or (A → B, B → A)

2.2.1.7 MemBankType

MemBankType enumeration type that returns memory bank of tag which mask data of Selection Mask will be compared to.

MemBank	Value	Description
MEM_RESERVED	0x0	Reserved Bank
MEM_EPC	0x1	EPC
MEM_TID	0x2	TID
MEM_USER	0x3	User Bank

2.2.1.8 KeyValueType

Key_Value	Value	Description
KEY_BOOTSTRAP_VERSION	0x1	The version of the bootstrap image loaded in the Indy Module.
KEY_BOOTSTRAP_CRC	0x2	The CRC (cyclical redundancy check) value of the bootstrap image on the Indy

		Module.
KEY_APPLICATION_VERSION	0x3	The version of the application image loaded in the Indy Module.
KEY_APPLICATION_CRC	0x4	The CRC (cyclical redundancy check) value of the application image on the Indy Module.
KEY_SECONDARY_IMAGE_VERSION	0x5	Secondary (non-application) image version numbers.
KEY_SECONDARY_IMAGE_CRC	0x6	Secondary (non-application) image CRC (cyclical redundancy check) values.
KEY_SECONDARY_IMAGE_TYPE	0x7	Secondary (non-application) image types
KEY_APPLICATION_REVISION_ID	0x8	Application image revision ID
KEY_APPLICATION_BUILD_ID	0x9	Application image build ID
KEY_PRODUCT_ID	0xA	A Product ID identifying the Indy Module product. See the ipj_product_id type enumeration for more details.
KEY_SERIAL_NUMBER	0xB	Serial Number of the Indy Module within a particular lot.
KEY_TRANSCEIVER_ID	0xC	The ID number of the Indy reader chip inside the module.
KEY_MICROPROCESSOR_ID	0xD	The unique ID of the individual microcontroller inside the Indy Module.
KEY_CUSTOMER_VERSION	0xE	Contains a version identifier for specially modified module hardware. 0 in most cases.
KEY_CUSTOMER_ID	0xF	Contains an identifier for specially modified module hardware. 0 in most cases.
KEY_CUSTOMER_PRODUCT_ID	0x10	Contains an identifier for specially modified module hardware. 0 in most cases.
KEY_CALIBRATION_INFO	0x11	Calibration source information.
KEY_TEST_INFO	0x12	The number of test commands executed.
KEY_PRODUCT_SKU	0x13	The hardware SKU of the module.
KEY_LOT_DATE_CODE	0x14	Combined lot and date code in hexadecimal format 0xZZWWYY where ZZ is lot number, WW is work week, and YY is year produced.
KEY_PRODUCT_KEY	0x15	A unique identifier for the module generated based on various parameters.
KEY_SECONDARY_IMAGE_LOCATION	0x16	Secondary Image Location

KEY_SECONDARY_IMAGE_SIZE	0x17	Secondary Image Size
KEY_UNIQUE_ID	0x18	A 64-bit Unique ID for each device XXZZWWYYAAAA where XX is SKU, ZZWWYY is Lot Date Code, and AAAA is Serial Number within the lot.
KEY_HARDWARE_REVISION	0x19	The hardware revision of the module.
KEY_SYSTEM_TIMESTAMP_MS	0x1A	The current timestamp of the system. This value is initialized to 0 at startup, and counts the number of ms that have elapsed since then.
KEY_ACTIVE_ACTIONS	0x1B	The commands actively running on the device.
KEY_REGION_ID	0x20	Sets the regulatory region parameters that the module will use.
KEY_REGION_CHANNEL_TABLE	0x21	The channel table for the configured custom regulatory region.
KEY_REGION_CHANNEL_TABLE_SIZE	0x22	The size of the custom region channel hop table.
KEY_REGION_ON_TIME_NOMINAL	0x23	Custom regulatory region dwell time (in ms).
KEY_REGION_ON_TIME_ACCESS	0x24	Custom regulatory region dwell time (in ms) when an Access operation is performed.
KEY_REGION_OFF_TIME	0x25	Off-time (in ms) when hopping to a different channel in the custom regulatory region.
KEY_REGION_OFF_TIME_SAME_CHANNEL	0x26	Off-time (in ms) when switching to the same channel in the custom regulatory region.
KEY_REGION_START_FREQUENCY_KHZ	0x27	Custom regulatory region channel 1 frequency in kHz.
KEY_REGION_CHANNEL_SPACING_KHZ	0x28	Custom regulatory region channel spacing in kHz.
KEY_REGION_RANDOM_HOP	0x29	Custom regulatory region random hop enable.
KEY_REGION_INDY_PLL_R_DIVIDER	0x2A	Indy PLL parameter. Recommended values are 24, 30, 48, and 60.
KEY_REGION_RF_FILTER	0x2B	Selects the RF SAW Filter for the user-configured custom regulatory region.
KEY_ANTENNA_TX_POWER	0x31	Antenna transmit power in cdBm (eg. 2300 cdBm = 23 dBm).
KEY_ANTENNA_SEQUENCE	0x32	The sequence of antennas through which

		the reader will cycle.
KEY_INVENTORY_TAG_POPULATION	0x40	A population estimate of the tags in view of the RF field of the antenna.
KEY_INVENTORY_SELECT_FLAG	0x41	Inventory Select Flag.
KEY_INVENTORY_SESSION	0x42	Inventory Session Number (0 - 3).
KEY_INVENTORY_SEARCH_MODE	0x43	Inventory Search Mode.
KEY_FAST_ID_ENABLE	0x45	Enable FastID capability for Monza tags.
KEY_TAG_FOCUS_ENABLE	0x46	Enable TagFocus capability for Monza tags. Session must be S1 and Search Mode must be A->B only.
KEY_TAG_OPERATION_ENABLE	0x47	Tag operation enable during inventory.
KEY_TAG_OPERATION_RETRIES	0x48	Maximum number of retries on a failed tag Access operation.
KEY_SELECT_ENABLE	0x50	Enables the transmission of a Select operation before an inventory rounds.
KEY_SELECT_TARGET	0x51	Determines if the Select command modifies a tag's selected flag or its inventoried flag.
KEY_SELECT_ACTION	0x52	Specifies what flag modification should be performed during the Select command.
KEY_SELECT_MEM_BANK	0x53	The memory bank on which the Select command 's tag filter is applied.
KEY_SELECT_POINTER	0x54	The bit offset in the specified memory bank at which the Select command 's tag mask begins.
KEY_SELECT_MASK_LENGTH	0x55	This key code along with the SELECT_POINTER key determines the memory range over which the Select command mask(configured with key SELECT_MASK_VALUE) is applied.
KEY_SELECT_MASK_VALUE	0x56	Defines the bit pattern that the Select command 's tag filter must match on.
KEY_TAG_OPERATION	0x60	Configures the specific Access command (tag operation) issued to a tag when an access command is enabled using the key TAG_OPERATION_ENABLE.
KEY_ACCESS_PASSWORD	0x61	Specifies the 32-bit Access password that is used in conjunction with the Access command to move the Tag to the Secured state.
KEY_KILL_PASSWORD	0x62	Specifies the 32-bit Kill password used to kill the Tag, if a Kill operation is enabled.

KEY_READ_MEM_BANK	0x63	Tag memory bank specified for Read command in an Access operation.
KEY_READ_WORD_POINTER	0x64	Word pointer for the Read command in an Access operation.
KEY_READ_WORD_COUNT	0x65	Number of words to read within the specified memory bank and at the specified word pointer in a Read command in an Access operation.
KEY_WRITE_MEM_BANK	0x66	Tag memory bank configured for a Write command in an Access operation.
KEY_WRITE_WORD_POINTER	0x67	Word Pointer to access for Write command in an Access operation.
KEY_WRITE_WORD_COUNT	0x68	Number of words to write in a Write command in an Access operation.
KEY_WRITE_DATA	0x69	Data to write into the tag memory in a Write command in an Access operation.
KEY_LOCK_PAYLOAD	0x6A	Payload field for the Lock command. Specified in a 20-bit value.
KEY_BLOCKPERMALOCK_ACTION	0x6B	BlockPermalock action for the BlockPermalock operation.
KEY_BLOCKPERMALOCK_MEM_BANK	0x6C	Memory Bank to target for the BlockPermalock operation.
KEY_BLOCKPERMALOCK_BLOCK_POINTER	0x6D	Block Pointer for the BlockPermalock operation.
KEY_BLOCKPERMALOCK_BLOCK_RANGE	0x6E	BlockPermalock mask range in units of 16 blocks for the BlockPermalock operation.
KEY_BLOCKPERMALOCK_MASK	0x6F	BlockPermalock mask for the BlockPermalock operation.
KEY_WRITE_EPC_LENGTH_CONTROL	0x70	EPC length handling control for the WRITE_EPC tag operation, when selected using the key TAG_OPERATION.
KEY_WRITE_EPC_LENGTH_VALUE	0x71	The user specified EPC length value when user value EPC length control is selected using key WRITE_EPC_LENGTH_CONTROL.
KEY_WRITE_EPC_AFI_CONTROL	0x72	Enables the write of the AFI bits in the tag EPC memory during a Write EPC operation if set to TRUE.
KEY_WRITE_EPC_AFI_VALUE	0x73	The user specified AFI Bits when AFI Control is enabled during a Write EPC operation.

KEY_QT_ACTION	0x74	QT Action for the QT operation.
KEY_QT_PERSISTENCE	0x75	QT Persistence for the QT operation.
KEY_QT_DATA_PROFILE	0x76	QT Data Profile for the QT operation.
KEY_QT_ACCESS_RANGE	0x77	QT Access Range for the QT operation.
KEY_QT_TAG_OPERATION	0x78	Determines the specific Access command issued to a Tag when Tag Operation enabled after the QT Command.
KEY_AUTOSTOP_DURATION_MS	0x89	Specifies the duration of time to perform continuous Inventory rounds.
KEY_AUTOSTOP_TAG_COUNT	0x8B	Specifies the maximum number of tags to inventory over multiple consecutive Inventory rounds.
KEY_AUTOSTOP_ROUND_COUNT	0x8C	Specifies the maximum number of Inventory rounds to be performed before stopping.
KEY_REPORT_CONTROL_TAG	0xA1	Controls which Tag fields are present in a tag operation report.
KEY_REPORT_CONTROL_STATUS	0xA2	Controls which status reports will be generated.
KEY_REPORT_CONTROL_TIMESTAMP	0xA3	Controls which reports from the module will contain timestamps.
KEY_RESPONSE_CONTROL_TIMESTAMP	0xA	Controls which responses from the module will contain timestamps.
KEY_TEMPERATURE_INTERNAL	0xB0	Current value of the temperature measured by the sensor in the microcontroller inside the module, in degrees Celsius.
KEY_TEMPERATURE_EXTERNAL	0xB1	Current extrapolated external temperature of the module in degrees Celsius.
KEY_TEMPERATURE_PA	0xB2	Current value of the temperature measured by the power amplifier (PA) temperature sensor in degrees Celsius.
KEY_GPIO_MODE	0xC0	Selects the mode of the GPIO pins of the module (input, input with action, output, etc.).
KEY_GPIO_STATE	0xC1	Controls the GPIO Logic level output (+3.3 V/0.0 V).
KEY_GPIO_HI_ACTION	0xC2	Controls the action performed by the module when a GPIO transitions to a High logical state.
KEY_GPIO_LO_ACTION	0xC3	Controls the action performed by the module when the signal input on a GPIO

		transitions to a Low logical state.
KEY_GPIO_PULSE_DURATION	0xC4	Control the pulse duration of the GPIO operation.
KEY_GPIO_DEBOUNCE_MS	0xC5	Controls internal debounce timeout for GPIO actions.
KEY_GPIO_CURRENT_STATE	0xC6	Reflects the current logic level (+3.3 V/0.0 V) of the GPIO pins.
KEY_RF_MODE	0xD0	RF Mode for inventory operation.
KEY_FIRST_ERROR	0xE0	Contains the first error that has occurred since this key was last cleared in value_index 0, as well as additional error parameters in value_index 1-4.
KEY_LAST_ERROR	0xE1	Contains the last error that occurred.
KEY_SYSTEM_ERROR	0xE2	Contains the last system level error that occurred (Hard Faults).
KEY_CAL_DATA_1	0xF0	DATA1.
KEY_CAL_DATA_2	0xF1	DATA2.
KEY_CAL_DATA_3	0xF2	DATA3.
KEY_CAL_DATA_4	0xF3	DATA4.
KEY_CAL_DATA_5	0xF4	DATA5.
KEY_CAL_DATA_6	0xF5	DATA6.
KEY_CAL_DATA_7	0xF6	DATA7.
KEY_CAL_DATA_8	0xF7	DATA8.
KEY_DEVICE_BAUDRATE	0x100	The serial baud rate that the module UART1 (IRI) interface operates at after boot.
KEY_DEVICE_IDLE_POWER_MODE	0x101	Power consumption mode when the device is idle.
KEY_ONBOOT_START_ACTION	0x102	Action to perform immediately after module boot.
KEY_ENABLE_LT_REPORTS	0x103	Configures the module to automatically send IRI-LT formatted reports over the UART1 (IRI) interface.
KEY_IN_BOOTSTRAP	0x104	Returns true if reader is in bootstrap.
KEY_EXTERNAL_ANTENNA_MUX_ENABLE	0x200	External antenna multiplexing enable.
KEY_EXTERNAL_ANTENNA_MUX_NUM_ANTENNAS	0x201	The number of antennas that the external antenna mux supports.
KEY_EXTERNAL_ANTENNA_MUX_PHYSICAL_PORT	0x202	The physical antenna port that the common port of the external antenna mux is connected to.
KEY_EXTERNAL_ANTENNA_MUX_DELAY	0x203	Delay in microseconds between RF

_MICROSECONDS		activity on externally mux antennas.
KEY_ANTENNA_SEQUENCE_OPTION	0x220	Configures whether configurations sequenced using the antenna descriptors are attached to different physical antenna ports, or are simply multiple logical descriptors for the same physical antenna.
KEY_ANTENNA_DESCRIPTORS	0x221	Banked key to hold key-value pairs of logical antenna parameters.
KEY_ANTENNA_PHYSICAL_PORT	0x222	Physical antenna port a specific antenna in the antenna descriptors sequence is connected to.
KEY_TEST_ID	0x400	Test Command ID is used to select the specific test command.
KEY_TEST_PARAMETERS	0x401	Test Command Parameters provide inputs for specific test commands.
KEY_TEST_RESULT_1	0x402	Test Command Result 1.
KEY_TEST_RESULT_2	0x403	Test Command Result 2.
KEY_TEST_RESULT_3	0x404	Test Command Result 3.
KEY_TEST_DATA	0x405	Test Command Data.
KEY_TEST_FREQUENCY	0x406	Last locked frequency.
KEY_TEST_POWER	0x407	Most recent transmit power.
KEY_TEST_RF_MODE	0x408	RF Profile identifier.
KEY_TEST_TIME	0x409	Last transmit on time.
KEY_TEST_EVENT	0x40A	Event Info.
KEY_TEST_REPORTS	0x40B	Report Info.
KEY_TEST_SYSTEM	0x40C	System Info.
KEY_TEST_DEBUG_PORT	0x40D	Serial Debug Port Configuration.
KEY_GENERIC_DATA	0xC00	Generic data storage for third parties to add custom data to the device.
KEY_OEM_DATA	0xC01	OEM data storage to add custom data to the device during calibration.
KEY_BLOCK_WRITE_OVERRIDE	0x1018	Controls block write override control.
KEY_SJC_CONTROL	0x101B	
KEY_STORED_SETTINGS_LOAD_STATUS	0x1021	Loading status for stored settings.
KEY_PA_DIE_TEMPERATURE_TX_DUTY_CYCLE_LIMIT	0x1022	RS1000 only. When the PA die, temperature exceeds this limit the transmitter on time duty cycle will proportionally be reduced in order to avoid hitting the PA die maximum thermal limit of 150 C. Setting this value to 150 C will

		disable this behavior.
KEY_TEMP_COMP_OFFSETS	0x1023	Constant offset added to each of the TempComp power curves.

Appendix I

Return Value	Value	Description
E_IPJ_ERROR_SUCCESS	0x0	Successfully performed last operation, or no error since boot.
E_IPJ_ERROR_GENERAL_ERROR	0x1	This error occurs in various unlikely scenarios that are not given their own error codes, including many errors that occur when using the test commands.
E_IPJ_ERROR_SET_KEY_INVALID	0x2	This error occurs if a key set operation is attempted with an invalid key ID, or some other invalid parameter.
E_IPJ_ERROR_SET_KEY_READ_ONLY	0x3	This error occurs when there is an attempt to set a key that is read only.
E_IPJ_ERROR_SET_KEY_OUT_OF_RANGE	0x4	This error occurs when a key is set to a value that is out of its valid range.
E_IPJ_ERROR_GET_KEY_INVALID	0x5	This error occurs when a key get or bulk get is performed with an invalid key ID, or some other invalid parameter.
E_IPJ_ERROR_GET_KEY_WRITE_ONLY	0x6	This error occurs when there is an attempt to get a key value that is write only.
E_IPJ_ERROR_COMMAND_INVALID	0x7	This error occurs when a start() API function call has an invalid action argument.
E_IPJ_ERROR_COMMAND_START_FAILURE	0x8	This error occurs when a start() API function call fails for a reason other than the others listed here.
E_IPJ_ERROR_COMMAND_DECODE_FAILURE	0x9	This error occurs when the IRI host library API fails to decode a packet from a module.
E_IPJ_ERROR_COMMAND_ENCODE_FAILURE	0xA	This error occurs when the IRI host library API fails to encode a packet to be sent to a module.
E_IPJ_ERROR_COMMAND_STALLED	0xB	This error occurs when an IRI host configures the module to perform a

		command or action that must complete within a specified amount of time, and the action fails to complete in that time.
E_IPJ_ERROR_VALUE_INVALID	0xC	This error occurs when the internal logic of the module determines that some aspect of the configuration is invalid.
E_IPJ_ERROR_MORE_THAN_ONE_COMMAND_RECEIVED	0xD	Module received multiple conflicting commands.
E_IPJ_ERROR_NOT_IMPLEMENTED	0xE	This error occurs when an IRI host requests a module use a specific feature that was not yet implemented in the version of the application image that is loaded on the module.
E_IPJ_ERROR_INVALID_PRODUCT_CONFIGURATION	0xF	This error occurs when a start() API function call is attempted on a module with an invalid product configuration.
E_IPJ_ERROR_INVALID_FACTORY_SETTINGS	0x10	This error occurs when a start() API function call is attempted on a module with invalid factory settings.
E_IPJ_ERROR_RESPONSE_ENCODE_FAILURE	0x11	Module encountered an error while trying to encode a report or response.
E_IPJ_ERROR_COMMAND_VERIFY_FAILURE	0x12	Module detected that the intended data was not properly written.
E_IPJ_ERROR_INTERNAL_NON_RECOVERABLE	0x13	Module encountered an internal non-recoverable error.
E_IPJ_ERROR_TEMPLATE_DECODE_FAILURE	0x14	Module was unable to properly decode the command template.
E_IPJ_ERROR_SYSTEM_IN_ERROR_STATE	0x15	This error occurs when a non-recoverable error has occurred and the module is in an error state.
E_IPJ_ERROR_TEST_ERROR	0x16	This error occurs when an IRI host configures a module to execute the error control test by setting the E_IPJ_KEY_TEST_ID key to the value E_IPJ_TEST_ID_ERROR_CONTROL.
E_IPJ_ERROR_STORED_SETTING_DECODE	0x17	This error occurs when decoding stored settings fails at a particular key code.
E_IPJ_ERROR_VALUE_INDEX_OUT_OF_RANGE	0x18	This error occurs when there is an attempt to set or get a key value with an invalid value_index.

E_IPJ_ERROR_BANK_INDEX_OUT_OF_RANGE	0x19	This error occurs when there is an attempt to set or get a key value with an invalid bank_index.
E_IPJ_ERROR_INVALID_PRODUCT_CALIBRATION	0x1A	This error occurs when a start() API function call is attempted on a module with invalid calibration data.
E_IPJ_ERROR_REPORT_SIZE_WOULD_OVERFLOW	0x1B	This error occurs when a tag operation report, with contents as configured by key E_IPJ_KEY_REPORT_CONTROL_TAG, would exceed the maximum size.
E_IPJ_ERROR_FIXED_HARDWARE_SETTINGS_LOAD_ERROR	0x1C	This error occurs when the module's firmware fails to load the fixed hardware settings that are programmed into it at calibration.
E_IPJ_ERROR_GEN2_TAG_OTHER_ERROR	0x1000001	Tag error catch-all for tag errors not covered by other codes
E_IPJ_ERROR_GEN2_TAG_MEMORY_OVERRUN	0x1000002	This error occurs when tag access operations are attempted with invalid parameters, such as an invalid memory locations or an invalid EPC length field.
E_IPJ_ERROR_GEN2_TAG_MEMORY_LOCKED	0x1000003	This can occur because the specified tag memory location is locked or permalocked, or is either not writeable or not readable.
E_IPJ_ERROR_GEN2_TAG_INSUFFICIENT_POWER	0x1000004	This error occurs when a tag memory write is attempted and the tag responds with an error indicating that it does not have sufficient power to perform the write.
E_IPJ_ERROR_GEN2_TAG_NON_SPECIFIC_ERROR	0x1000005	Tag error when tag does not support specific error codes
E_IPJ_ERROR_API_DEVICE_NOT_INITIALIZED	0x2000001	This error occurs when the IRI host library does not properly initialize the iri_device structure.
E_IPJ_ERROR_API_CONNECTION_PORT_ERROR	0x2000002	This error occurs when the IRI host library is unable to control the UART serial port used to implement IRI communication with the module.
E_IPJ_ERROR_API_CONNECTION_READ_TIMEOUT	0x2000003	This error occurs when the IRI host library runs out of time while waiting for IRI traffic to return from the module.

E_IPJ_ERROR_API_CONNECTION_WRITE_TIMEOUT	0x2000004	IRI connection write timeout
E_IPJ_ERROR_API_CONNECTION_WRITE_ERROR	0x2000005	This error occurs when the IRI host library attempts to transmit an IRI packet via the UART and fails to send a complete packet.
E_IPJ_ERROR_API_RX_BUFF_TOO_SMALL	0x2000006	IRI host receive buffer too small
E_IPJ_ERROR_API_MESSAGE_INVALID	0x2000007	IRI host detected an invalid message
E_IPJ_ERROR_API_NO_HANDLER	0x2000008	This error occurs when the IRI host library's iri_device structure does not have a valid handler function defined to handle a required action for IRI operation.
E_IPJ_ERROR_API_INVALID_LOADER_BLOCK	0x2000009	This error occurs when the IRI host library's ipj_flash_handle_loader_block() API is passed a data block with an invalid CRC during an application image bootload operation.
E_IPJ_ERROR_API_RESPONSE_MISMATCH	0x200000A	This error occurs when the IRI host library sends a command, then receives a response that does not correspond to that command.
E_IPJ_ERROR_API_INVALID_PARAMETER	0x200000B	This error occurs when the IRI host library APIs are called with invalid parameters.
E_IPJ_ERROR_API_NON_LT_PACKET_DETECTED	0x200000C	This error occurs when the host is configured for IRI-LT operation, and the module communicates a non-IRI-LT packet to it unexpectedly.
E_IPJ_ERROR_IRI_FRAME_DROPPED	0x3000001	This error occurs when the IRI host library detects a packet from a module without the proper length or structure.
E_IPJ_ERROR_IRI_FRAME_INVALID	0x3000002	This error occurs when the calculated IRI frame CRC or parity did not match the value that was sent. It usually indicates some error in packet formation or transmission.
E_IPJ_ERROR_MAC_GENERAL	0x4000001	A non-specific communication error occurred inside the module.
E_IPJ_ERROR_MAC_CRC_MISMATCH	0x4000002	Module detected a CRC mismatch on

		tag response.
E_IPJ_ERROR_MAC_NO_TAG_RESPONSE	0x4000003	Module detected no tag response.
E_IPJ_ERROR_MAC_TAG_LOST	0x4000004	Module detected the tag was lost mid-transaction.
E_IPJ_ERROR_BTS_DEVICE_WATCHDOG_RESET	0x5000001	This error occurs when a module's watchdog reset functionality detects that it has reached an unrecoverable firmware operating state, and has reset it.
E_IPJ_ERROR_BTS_VALUE_INVALID	0x5000002	This error occurs when a module's bootloader has detected that a command was sent from a host with an invalid command argument.
E_IPJ_ERROR_BTS_FLASH_WRITE	0x5000003	This error occurs when a module's bootloader is incapable of performing a flash write as instructed by the IRI host, due to an invalid address or other configuration error.
E_IPJ_ERROR_BTS_FLASH_READ	0x5000004	This error occurs when a module's bootloader is incapable of performing a flash read as instructed by the IRI host, due to an invalid address or other configuration error.
E_IPJ_ERROR_BTS_FLASH_ADDRESS	0x5000005	This error occurs when a module's bootloader is incapable of erasing the flash as instructed by the IRI host, because the address is protected from erasure.
E_IPJ_ERROR_BTS_FLASH_ERASE	0x5000006	This error occurs when a module's bootloader is incapable of erasing the flash as instructed by the IRI host, because of a reason other than flash protection.
E_IPJ_ERROR_BTS_UNKNOWN_COMMAND	0x5000007	This error occurs when a module's bootloader receives an unknown command from the IRI host.
E_IPJ_ERROR_BTS_COMMAND_DECODE_FAILURE	0x5000008	This error occurs when a module's bootloader receives a command from the IRI host but is unable to properly decode it.
E_IPJ_ERROR_TRANSCEIVER_FAILURE	0x6000001	This error occurs when internal

		communications with the radio transceiver inside the module fail.
E_IPJ_ERROR_LIMIT_PA_TEMPERATURE_MAX	0x7000001	This error occurs when the power amplifier temperature exceeds the maximum allowed value.

Appendix II

- InventoryResult

Items displayed during inventory operation
rssi
channel
phase
antenna
TagData tagData

- TagData

Tag Data
pc
epc
tid
data

Appendix III：

Device Info.
macAddress
version
status
ipAddress
subnetMask
gateway
dns
dhcp
password

Appendix IV：

TagMask
userMemoryBit1
userMemoryBit2
tidMemoryBit1
tidMemoryBit2
epcMemoryBit1
epcMemoryBit2
accessMemoryBit1
accessMemoryBit2
killMemoryBit1
killMemoryBit2

TagAction
userMemoryBit1
userMemoryBit2
tidMemoryBit1
tidMemoryBit2
epcMemoryBit1
epcMemoryBit2
accessMemoryBit1
accessMemoryBit2
killMemoryBit1
killMemoryBit2