Introduction

- Provides command-line tools for automating tasks using scripts
- Provide dll underlying API, written in C language, which can be called by various application layer languages (C# Python Matlab Qt, etc.)
- You can use the ID number to distinguish between multiple identical devices
- The fastest implementation of upper-level development, only a few api functions to achieve the required functions

Simple example

JTool CMD and DLL are command line script tools and secondary development libraries for USB to I2C, USB to IO and other tools.

- CMD tools are usually useful in production test environments, such as implementing automated actions or detection in bat batch scripts or Powershell scripts.
- DLL is used to provide API called by application layer to facilitate secondary development (C#, Python, QT, Matlab, etc.)

CMD application example in Powershell

Get the ADC sample value of IO3 and judge whether it is within the range.

```
jtool adcon 3 0 # Channel 3 as ADC sampling, do not use differential
$output = & "jtool" adcget 3 # Get the sample value of Channel 3
= 0
# Try to convert the output to an integer
if ([int]::TryParse($output, [ref] )) {
    = [Math]::Round( * 3.3 / 4096,2)
    "Conversion succeeded, output number: $number"
    = [Math]::Abs(-3.3)
    "Comparison deviation value: $number"
    if($number -gt 0.3){ # judgment deviation
        Write-Error "Compare failed, out of range"
    }
} else {
    $output
    Write-Error "Conversion failed, the output is not a valid number"
}
```

Application example of DLL in C# Program

Read I2C data

```
// Import DLLAPI
[DllImport("jtool.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern IntPtr DevOpen(DevType DevType, string Sn, int Id);
[DllImport("jtool.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern bool DevClose(IntPtr DevHandle);
[DllImport("jtool.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int I2CRead(IntPtr DevHandle, byte slave_addr, int reg_type,
UInt32 reg_addr, UInt16 len, byte[] buf);

private void Read()
{
    // Open the device without specifying SN and ID, as long as there is an I2C
device
    IntPtr p = DevOpen(DevType.dev_i2c, null, -1);
    if (p == IntPtr.Zero)
    {
        Console.WriteLine("Failed to Open Device");
        return;
    }

    int readlen = 16;
    byte[] buffread = new byte[readlen];
    I2CRead(p, 0xa0, 1, 0x 00, readlen, buffer);//Read slave address A0, register
address 00, read 16 bytes to buffer

    Console.WriteLine(BitConverter.ToString (buffer).Replace("-", ""));// Print
the read data
    DevClose(p);// Shut down the device
}
```

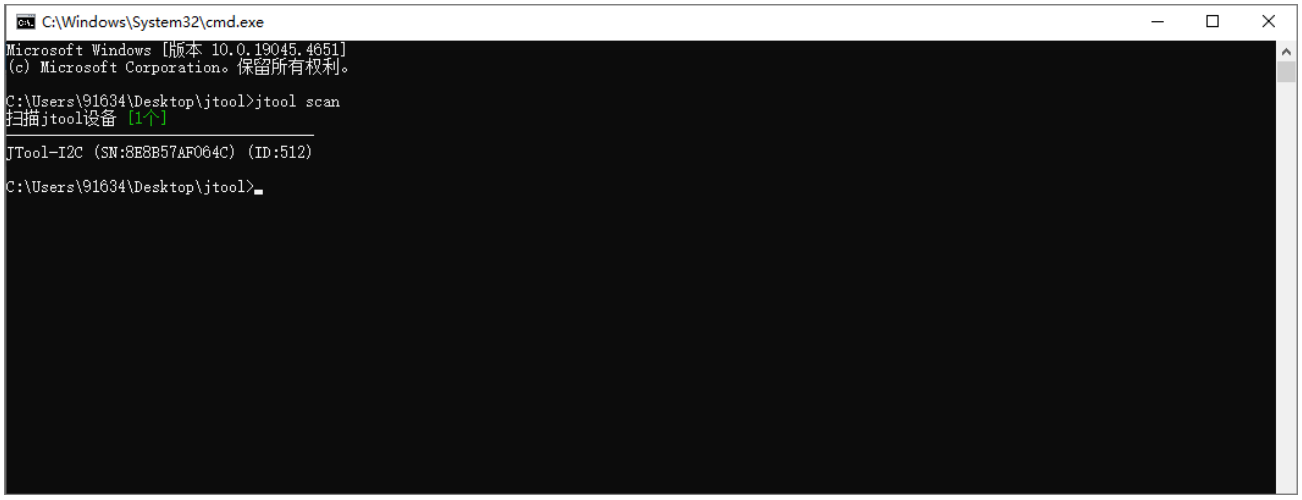
> Before using cmd or dll, it is recommended to use the upper computer provided by us to ensure normal use.

CMD command set

The command line is called in the system CMD program. If you double-click to open jtool.exe directly, it will not run

Please call in CMD, as shown in the figure, start CMD in the jtool Directory

After opening the command line window, call jtool with the command and parameters.



The meaning and parameters of each command will be explained in detail below.

> jtool [command] [-Option 1] [-Option 2] [-Option n] [parameter 1] [parameter 2] [parameter n]

help

View all command collections

```
jtool --help
```

View the meaning and parameters of a single command

```
jtool [command] -- help
```

scan

View all inserted jtool devices.

```
jtool scan
```

delay

Delay time (ms)

```
jtool delay 1000
```

--------JIO --------

ioh

Set the IO port to high level.

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The pin number that needs to be set high

```
# Set IO1 high
jtool ioh 1

# Set the device IO1 with ID 0 High
jtool ioh -i 0 1

# Set all IO1 ~ IO4 high
jtool ioh -m 0x0f
```

iol

Set the IO port low

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The pin number that needs to be set low

```
# Set IO1 low
jtool iol 1

# Set the device IO1 with ID 0 to low
jtool iol -i 0 1

# Set all IO1 ~ IO4 low
jtool iol -m 0x0f
```

iow

Write IO port to fixed level

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The PIN number of the write level required
param1	The value of the level to be written

```
# Write IO1 low
jtool iow 1 0

# Write the device IO1 with ID 0 High
jtool iow -i 0 1 1

# Write IO1 and IO2 to high level, and IO3 and IO4 to low level
jtool iow -m 0x0f 0x03
```

ior

Read IO port level

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The pin number that requires the read level

```
# Read the level of IO1
jtool ior 1

# Read the level of device IO1 with ID 0
jtool ior -i 0 1

# Read the levels of IO1 and IO3 at the same time
jtool ior -m 0x05
```

pulseon

Output pulse normally on

Options/ Parameters	Description
-i	Use the device with the specified id

Options/ Parameters	Description
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The pin number that requires a pulse output
param1	Frequency of pulse output (refer to JIO upper computer for optional values)

```
# IO1 output pulse normally open 400kHz
jtool pulseon 1 400000

# The IO1 output pulse of the device with ID 0 is normally on 400kHz
jtool pulseon -i 0 1 400000

#4 channel simultaneous output pulse normally open 400kHz
jtool pulseon -m 0x0f 400000
```

pulseoff

Turn off output pulse

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The pin number that requires the turn-off pulse

```
# IO1 off pulse
jtool pulseoff 1

# Device IO1 shutdown pulse with ID 0
jtool pulseoff -i 0 1

#4 Channel Simultaneous Off Pulse
jtool pulseoff -m 0x0f
```

pulsecnt

Output fixed pulse number

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The pin number that requires the pulse output
param1	Number of pulse outputs
param2	Frequency of pulse output (refer to JIO upper computer for optional values)

```
# IO1 output 1000 pulses 100kHz
jtool pulsecnt 1 1000 100000

# Device IO1 with ID 0 outputs 1000 pulses 100kHz
jtool pulsecnt -i 0 1 1000 100000

#4 channel output 1000 pulses at the same time 100kHz
jtool pulsecnt -m 0x0f 1000 100000
```

pwmon

Turn on PWM output

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The pin number that requires a pulse output
param1	Duty cycle of the PWM output (0 to 1000 corresponds to a duty cycle of 0% to 100)

```
# IO1 output PWM with 50% duty cycle
jtool pwmon 1 500

# Device IO1 with ID 0 outputs PWM with 50% duty cycle
jtool pwmon -i 0 1 500

#4 channel simultaneous output PWM with 50% duty cycle
jtool pwmon -m 0x0f 500
```

pwmoff

Turn off PWM output

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The pin number that needs to close the PWM

```
# IO1 Close PWM
jtool pwmoff 1

# Disable PWM for device IO1 with ID 0
jtool pwmoff -i 0 1

#4 channel at the same time off PWM
jtool pwmoff -m 0x0f
```

pwmfreq

Set PWM frequency (applies to all PWM channels)

Options/ Parameters	Description
-i	Use the device with the specified id
param0	PWM output frequency (refer to JIO host computer for optional values)

```
# Set PWM frequency to 1kHz
jtool pwmfreq 1000

# Set PWM frequency to 1kHz for devices with ID 0
jtool pwmfreq -i 0 1000
```

capclear

Clear PWM captured pulse count (Channel 1 only)

Options/ Parameters	Description
-i	Use the device with the specified id
param0	The pin number that needs to clear the count (this parameter can only be 1, only channel 1 supports PWM capture)

```
# Clear the capture Pulse count of IO1
jtool capclear 1

# The device with ID 0 clears the capture Pulse count of IO1
jtool capclear -i 0 1
```

capon

Enable PWM capture (Channel 1 only)

Options/ Parameters	Description
-i	Use the device with the specified id
param0	The pin number that needs to enable PWM capture (this parameter can only be 1, only channel 1 supports PWM capture)

```
# Turn on the capture function of IO1
jtool capon 1

# The device with ID 0 clears the capture Pulse count of IO1
jtool capon -i 0 1
```

capget

Get PWM capture data (only channel 1 is supported)

Options/ Parameters	Description
-i	Use the device with the specified id
param0	The PIN number of the capture value to be obtained (this parameter can only be 1, only channel 1 supports PWM capture)

```
# Get the captured data of IO1
jtool capget 1

# The device with ID 0 obtains the captured data of IO1
jtool capget -i 0 1
```

adcsamp

Set the ADC sampling rate (applies to all ADC channels)

Options/ Parameters	Description
-i	Use the device with the specified id
param0	Configure the sampling rate of ADC (refer to JIO host computer for optional values)

```
# Set the ADC frequency to 50Hz
jtool adcsamp 50

# Set the ADC frequency to 50Hz for devices with ID 0
jtool adcsamp -i 0 50
```

adcon

Turn on ADC sampling

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	Pin number that requires ADC sampling to be enabled
param1	Whether it is Differential sampling (0 single-ended 1 differential) (set to differential will occupy the next pin)

```
# Enable ADC sampling of IO1
jtool adcon 1 0

# Enable ADC sampling of IO1 (differential IO1 + IO-)
jtool adcon 1 1

# Enable ADC sampling of IO1 for devices with ID 0
jtool adcon -i 0 1 0

#4 channel at the same time open ADC sampling (two differential)
jtool adcon -m 0x05 0x05
```

adcget

Get ADC sample values

Options/ Parameters	Description
-i	Use the device with the specified id
-m	Whether to execute in bitwise mask mode, used to execute multiple pins at the same time
param0	The pin number needed to get the ADC sample value

```
# Obtain the ADC sampling value of IO1
jtool adcget 1

# The device with ID 0 obtains the ADC sample value of IO1
jtool adcget -i 0 1

# Get the ADC sample values of IO1 and IO3
jtool adcget -m 0x05
```

jiovcc

Set the VCC output voltage of the JIO device

Options/ Parameters	Description
-i	Use the device with the specified id
param0	VCC option enumeration value, different models may support different, see The JIO interface drop-down box (for example: 0 - 5V; 1 - 3.3V; 2-closed)

```
# VCC voltage set to 5V
jtool jiovcc 0

# VCC voltage set to 3.3V
jtool jiovcc 1

# VCC voltage set to off
jtool jiovcc 2
```

jiovio

Set the VIO level voltage of the JIO device

Options/ Parameters	Description
-i	Use the device with the specified id
param0	VIO option enumeration value. Different models may support different values. For details, please refer to the drop-down box of the JIO interface (for example: 0 - 3.3V; 1 - 1.8V)

```
# VIO voltage set to 3.3V
jtool jiovio 0
```

jioid

Set the ID of the JIO device.
This option is set to take effect after a reboot

Options/Parameters	Description
-i	Use the device with the specified id
param0	The id to be modified.

```
# Set the device id value to 1
jtool jioid 1

# Set the id value of the device with the original id of 0 to 1
jtool jioid -i 0 1
```

jioreboot

Restart the JIO device

Options/Parameters	Description
-i	Use the device with the specified id

```
jtool jioreboot
```

jioinboot

JIO device into bootloader

Options/Parameters	Description
-i	Use the device with the specified id

```
jtool jioinboot
```

-------- JI2C--------

i2cscan

Scan the I2C slave address

Options/ Parameters	Description
-i	Use the device with the specified id
-s	slave7 uses 7-bit address mode (does not include read and write bits)

```
# Scan the I2C slave address
jtool i2cscan

# Use the device with id 0 to scan the I2C slave address
jtool i2cscan -i 0

# Scan the I2C slave address and display it as a 7-bit address
jtool i2cscan -s
```

i2cwrite

I2C write data

Options/ Parameters	Description
-i	Use the device with the specified id
-s	slave7 uses 7-bit address mode (does not include read and write bits)
param0	Slave address
param1	Register address (determine the type of register according to the length, please fill in none without Register)
param2	Data to be written (array)

```
# Slave address a0 register address 00 write 11 22 33 44 55
jtool i2cwrite A0 00 11 22 33 44 55

# Slave address a0 register address 0000 (representing register address is 2
bytes) write 11 22 33 44 55
jtool i2cwrite A0 0000 11 22 33 44 55

# Slave address a0 register address 000000 (representing register address is 3
bytes) write 11 22 33 44 55
jtool i2cwrite A0 000000 11 22 33 44 55
```

i2cread

I2C read data

Options/ Parameters	Description
-i	Use the device with the specified id
-s	slave7 uses 7-bit address mode (does not include read and write bits)
-d	d1 read delay Sr plus delay (0 ~100ms)
-D	d2 read delay plus delay after reading address (0 ~100ms)
param0	Slave address
param1	Register address (determine the type of register according to the length, please fill in none without Register)
param2	Length to read (decimal)

```
# Slave address a0 register address 00 reads 5 bytes
jtool i2cread A0 00 5

# Slave address a0 register address 0000 (representing register address is 2
bytes) reads 5 bytes
jtool i2cread A0 0000 5

# Slave address a0 register address 000000 (representing register address is 3
bytes) reads 5 bytes
jtool i2cread A0 000000 5
```

eewrite

EEPROM write data
The read and write operations of EEPROM have been cross-page and cross-block processing. There is no need to worry about the call, and data of any length can be read and written from any address.
Need to make sure baseslave is correct
Need to make sure regaddr is the correct length
Need to make sure pagesize is correct (needed for cross-page writing)

Options/ Parameters	Description
-i	Use the device with the specified id
-s	slave7 uses 7-bit address mode (does not include read and write bits)
-B	blockhigh Block in high (for 24AA(LC/FC)1025)
-p	**(Required) page size of EEPROM for cross-page write processing (decimal, in Byte)**
param0	Slave address
param1	**register address (determine the type of register according to the length, eeprom some are 1 byte, some are 2 bytes)**
param2	Data to be written (array)

```
# Example 24C01 Register Address 00 Write 11 22 33 44 55
jtool eewrite -p 8 A0 00 11 22 33 44 55

# Example 24C32 register address 0000 (2 bytes for this model) write 11 22 33 44
55
jtool eewrite -p 32 A0 0000 11 22 33 44 55
```

eeread

EEPROM read data
The read and write operations of EEPROM have been cross-page and cross-block processing. There is no need to worry about the call, and data of any length can be read and written from any address.
Need to make sure baseslave is correct
Need to make sure regaddr is the correct length

Options/ Parameters	Description
-i	Use the device with the specified id
-s	slave7 uses 7-bit address mode (does not include read and write bits)
-B	blockhigh Block in high (for 24AA(LC/FC)1025)
param0	Slave address
param1	**register address (determine the type of register according to the length, eeprom some are 1 byte, some are 2 bytes)**
param2	Length to read (decimal)

```
# Example 24C01 register address 00 reads 5 bytes
jtool eeread A0 00 5

# Example 24C32 register address 0000 (2 bytes for this model) reads 5 bytes
jtool eeread A0 0000 5
```

eewritef

EEPROM write file

The read and write operations of EEPROM have been cross-page and cross-block processing. There is no need to worry about the call, and data of any length can be read and written from any address.
Need to make sure baseslave is correct
Need to make sure regaddr is the correct length
Need to make sure pagesize is correct (needed for cross-page writing)

Options/ Parameters	Description
-i	Use the device with the specified id
-s	slave7 uses 7-bit address mode (does not include read and write bits)
-B	blockhigh Block in high (for 24AA(LC/FC)1025)
-p	**(Required) page size of EEPROM for cross-page write processing (decimal, in Byte)**
param0	Slave address
param1	**register address (determine the type of register according to the length, eeprom some are 1 byte, some are 2 bytes)**
param2	The file path needs to be written.

```
# Example 24C01 Register Address 00 Write data.bin
jtool eewritef -p 8 A0 00 .\data.bin

# Example 24C32 register address 0000 (2 bytes for this model) writes data.bin
jtool eewritef -p 32 A0 0000 .\data.bin
```

eereadf

EEPROM read data to file
EEPROM read data
The read and write operations of EEPROM have been cross-page and cross-block processing. There is no need to worry about the call, and data of any length can be read and written from any address.
Need to make sure baseslave is correct
Need to make sure regaddr is the correct length

Options/ Parameters	Description
-i	Use the device with the specified id
-s	slave7 uses 7-bit address mode (does not include read and write bits)
-B	blockhigh Block in high (for 24AA(LC/FC)1025)
param0	Slave address
param1	**register address (determine the type of register according to the length, eeprom some are 1 byte, some are 2 bytes)**
param2	Length to read (decimal)
param3	The path to the folder to be saved (note that the folder is not a file)

```
# Example 24C01 register address 00 reads 5 bytes to the current directory
jtool eereadf A0 00 5 .\

# Example 24C32 register address 0000 (2 bytes for this model) reads 5 bytes into
the current directory
jtool eereadf A0 0000 5 .\
```

i2cint

Detect interrupt pin for Interrupt

Options/ Parameters	Description
-i	Use the device with the specified id
-w	wait Whether to wait for an interrupt to exit the process
param0	Interrupt type (0: None 1: rising edge 2: Falling Edge 3: High Level 4: Low Level 5: Double Edge)

```
# Detect rising edge interrupt
jtool i2cint 1

# Exit after waiting for rising edge interrupt
jtool i2cint -w 1
```

ji2cvcc

Set the VCC output voltage of the JI2C device

Options/ Parameters	Description
-i	Use the device with the specified id
param0	VCC option enumeration value, different models may support different, see the JI2C interface drop-down box (for example: 0 - 5V; 1 - "= VIO" ; 2-closed)

```
# VCC voltage set to 5V
jtool ji2cvcc 0

# VCC voltage set to "= VIO"
jtool ji2cvcc 1

# VCC voltage set to off
jtool ji2cvcc 2
```

ji2cvio

Set the VIO level voltage of the JI2C device

Options/ Parameters	Description
-i	Use the device with the specified id
param0	VIO option enumeration value, different models may support different, see the JI2C interface drop-down box (for example: 0 - 3.3V; 1 - 1.8V)

```
# VIO voltage set to 3.3V
jtool ji2cvio 0
```

ji2cspd

Set the clock rate of the JI2C device

Options/ Parameters	Description
-i	Use the device with the specified id
param0	I2C rate option enumeration value, different models may support different, refer to the JI2C interface drop-down box (for example: 0 - 10K; 1 - 50K)

```
# I2C rate set to 100K
jtool ji2cspd 2
```

ji2cid

Set the ID of the JI2C device.
This option is set to take effect after a reboot

Options/Parameters	Description
-i	Use the device with the specified id
param0	The id to be modified.

```
# Set the device id value to 1
jtool ji2cid 1

# Set the id value of the device with the original id of 0 to 1
jtool ji2cid -i 0 1
```

ji2creboot

Restart the JI2C device

Options/Parameters	Description
-i	Use the device with the specified id

```
jtool ji2creboot
```

ji2cinboot

JI2C device into bootloader

Options/Parameters	Description
-i	Use the device with the specified id

```
jtool ji2cinboot
```

--------JSPI --------

spiwrite

SPI write-only data

Options/ Parameters	Description
-i	Use the device with the specified id
-m	mode Specifies the SPI clock mode (default is 0): 0-LOW_1EDG; 1-LOW_2EDG; 2-HIGH_1EDG; 3-HIGH_2EDG
-e	endian specifies SPI first order (default is 0): 0-MSB; 1-LSB
param0	Data to be written (array)

```
# spi Write to array (default clock and bit order)
jtool spiwrite 00 01 02 03 04 05

# spi write array (clock HIGH_1EDG bit order LSB)
jtool spiwrite -m 2 -e 1 00 01 02 03 04 05
```

spiread

SPI read data only

Options/ Parameters	Description
-i	Use the device with the specified id
-m	mode Specifies the SPI clock mode (default is 0): 0-LOW_1EDG; 1-LOW_2EDG; 2-HIGH_1EDG; 3-HIGH_2EDG
-e	endian specifies SPI first order (default is 0): 0-MSB; 1-LSB
param0	Length to read

```
# spi read 5 bytes (default clock and bit order)
jtool spiread 5

# spi read 5 bytes (clock HIGH_1EDG bit order LSB)
jtool spiread -m 2 -e 1 5
```

spiwr

SPI write while reading

Options/ Parameters	Description
-i	Use the device with the specified id
-m	mode Specifies the SPI clock mode (default is 0): 0-LOW_1EDG; 1-LOW_2EDG; 2-HIGH_1EDG; 3-HIGH_2EDG
-e	endian specifies SPI first order (default is 0): 0-MSB; 1-LSB
param0	Data to be written (array)

```
# spi Write to array (default clock and bit order)
jtool spiwr 00 01 02 03 04 05

# spi write array (clock HIGH_1EDG bit order LSB)
jtool spiwr -m 2 -e 1 00 01 02 03 04 05
```

qspiwrite

QSPI write-only data

Options/ Parameters	Description
-i	Use the device with the specified id
-m	mode Specifies the QSPI clock mode (default is 0): 0-LOW_1EDG; 1-LOW_2EDG; 2-HIGH_1EDG; 3-HIGH_2EDG
-e	endian specifies QSPI first order (default is 0): 0-MSB; 1-LSB
param0	Data to be written (array)

```
# spi Write to array (default clock and bit order)
jtool qspiwrite 00 01 02 03 04 05

# spi write array (clock HIGH_1EDG bit order LSB)
jtool qspiwrite -m 2 -e 1 00 01 02 03 04 05
```

qspiread

QSPI read data only

Options/ Parameters	Description
-i	Use the device with the specified id
-m	mode Specifies the QSPI clock mode (default is 0): 0-LOW_1EDG; 1-LOW_2EDG; 2-HIGH_1EDG; 3-HIGH_2EDG
-e	endian specifies QSPI first order (default is 0): 0-MSB; 1-LSB

Options/ Parameters	Description
param0	Length to read

```
# spi read 5 bytes (default clock and bit order)
jtool qspiread 5

# spi read 5 bytes (clock HIGH_1EDG bit order LSB)
jtool qspiread -m 2 -e 1 5
```

spiwcmd

SPI writes data with instructions (CMD, ADDR, ALT, DUMMY)

Options/ Parameters	Description
-i	Use the device with the specified id
-m	mode Specifies the SPI clock mode (default is 0): 0-LOW_1EDG; 1-LOW_2EDG; 2-HIGH_1EDG; 3-HIGH_2EDG
-e	endian specifies SPI first order (default is 0): 0-MSB; 1-LSB
-q	qspi Specifies the SPI/QSPI combination type (default is 0): 0-all one-line; 1-all four-line; 2-data four-line only; 3-instruction one-line only
-c	cmd instruction (hexadecimal) For example: 00 represents a 1-byte instruction; 0000 represents a 2-byte instruction; Maximum 4 bytes
-a	addr address (hexadecimal) For example: 00 represents a 1-byte address; 0000 represents a 2-byte address; Maximum 4 bytes
-t	alt (hexadecimal) For example: 00 represents 1 byte alt; 0000 represents 2 bytes alt; Maximum 4 bytes
-d	dummy dummy empty period length (decimal) for example: 1 represents 1 byte dummy; 2 represents 2 bytes dummy; Maximum 4 bytes
param0	Data to be written (array)

```
# spi write array {0x 55,0x 55,0x 55} cmd instruction is 0x01
jtool spiwcmd -c 01 55 55 55

# spi write array {0x 55,0x 55,0x 55} cmd instruction is 0x 01 instruction phase
single line, data phase four line QSPI
jtool spiwcmd -q 2 -c 01 55 55 55
```

spircmd

SPI read data with instructions (CMD, ADDR, ALT, DUMMY)

Options/ Parameters	Description
-i	Use the device with the specified id
-m	mode Specifies the SPI clock mode (default is 0): 0-LOW_1EDG; 1-LOW_2EDG; 2-HIGH_1EDG; 3-HIGH_2EDG
-e	endian specifies SPI first order (default is 0): 0-MSB; 1-LSB
-q	qspi Specifies the SPI/QSPI combination type (default is 0): 0-all one-line; 1-all four-line; 2-data four-line only; 3-instruction one-line only
-c	cmd instruction (hexadecimal) For example: 00 represents a 1-byte instruction; 0000 represents a 2-byte instruction; Maximum 4 bytes
-a	addr address (hexadecimal) For example: 00 represents a 1-byte address; 0000 represents a 2-byte address; Maximum 4 bytes
-t	alt (hexadecimal) For example: 00 represents 1 byte alt; 0000 represents 2 bytes alt; Maximum 4 bytes
-d	dummy dummy empty period length (decimal) for example: 1 represents 1 byte dummy; 2 represents 2 bytes dummy; Maximum 4 bytes
param0	Length to read

```
# spi read 3 bytes cmd instruction is 0x01
jtool spircmd -c 01 3

# spi reading 3 bytes cmd instruction is 0x 01 instruction phase single line, data
 phase four line QSPI
jtool spircmd -q 2 -c 01 3
```

spiint

Detect interrupt pin for Interrupt

Options/ Parameters	Description
-i	Use the device with the specified id
-w	wait Whether to wait for an interrupt to exit the process
param0	Interrupt type (0: None 1: rising edge 2: Falling Edge 3: High Level 4: Low Level 5: Double Edge)

```
# Detect rising edge interrupt
jtool spiint 1

# Exit after waiting for rising edge interrupt
jtool spiint -w 1
```

jspivcc

Set the VCC output voltage of the JSPI device

Options/ Parameters	Description
-i	Use the device with the specified id
param0	VCC option enumeration value, different models may support different, see JSPI interface drop-down box (for example: 0 - 5V; 1 - "= VIO" ; 2-closed)

```
# VCC voltage set to 5V
jtool jspivcc 0

# VCC voltage set to "= VIO"
jtool jspivcc 1

# VCC voltage set to off
jtool jspivcc 2
```

jspivio

Set VIO-level voltage for JSPI devices

Options/ Parameters	Description
-i	Use the device with the specified id
param0	VIO option enumeration value, different models may support different, see JSPI interface drop-down box (for example: 0 - 3.3V; 1 - 1.8V)

```
# VIO voltage set to 3.3V
jtool jspivio 0
```

jspispd

Set the clock rate of the JSPI device

Options/ Parameters	Description
-i	Use the device with the specified id
param0	SPI rate option enumeration value, different models may support different, specific reference JSPI interface drop-down box (for example: 0 - 468.75K; 1 - 937.5K)

```
# SPI rate set to 937.5k
jtool jspispd 21
```

jspiid

Set the JSPI device ID.

This option is set to take effect after a reboot

Options/Parameters	Description
-i	Use the device with the specified id
param0	The id to be modified.

```
# Set the device id value to 1
jtool jspiid 1

# Set the id value of the device with the original id of 0 to 1
jtool jspiid -i 0 1
```

jspireboot

Restart the JSPI device

Options/Parameters	Description
-i	Use the device with the specified id

```
jtool jspireboot
```

jspiinboot

JSPI device into bootloader

Options/Parameters	Description
-i	Use the device with the specified id

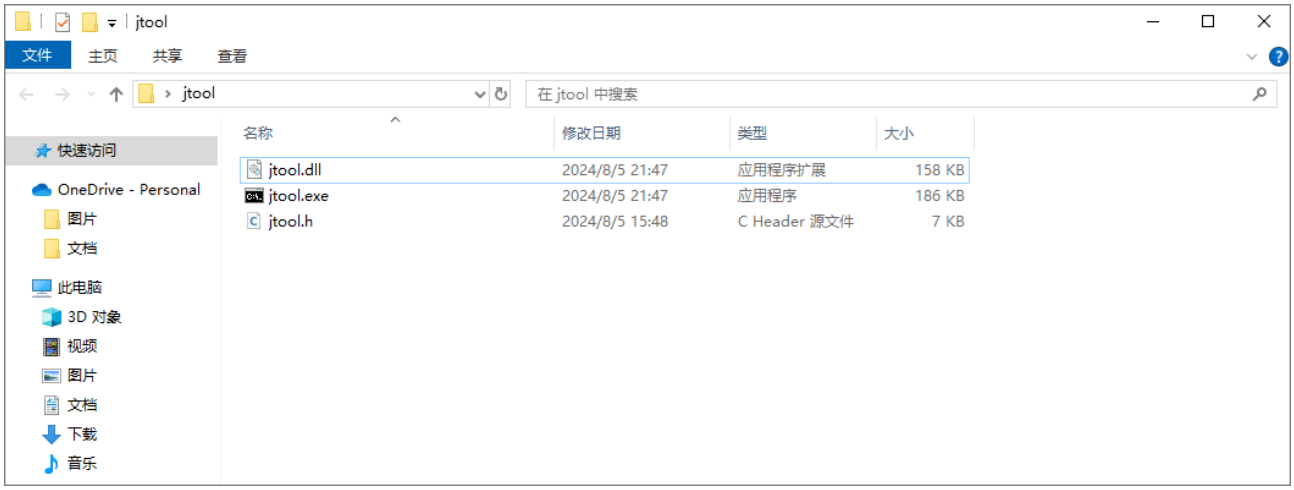
```
jtool jspiinboot
```

DLL API

jtool.dll is the lowest level dynamic link library written by C language to be compatible with all other application layer language calls

jtool.dll provides the most simple and easy-to-use API, which can realize the required functions most quickly.

Just import the jtool.dll file into your own project, where the jtool. H file contains all API interface and enumeration declarations



Enumeration type

```
typedef enum
{
        dev_all = -1,
        dev_i2c = 0,
        dev_io,
        dev_spi,
        dev_can,
        dev_max,
} dev_type_enum;

typedef enum
{
        ErrNone = 0,// success
        ErrParam = 1 << 0,// Parameter error
        ErrDisconnect = 1 << 1,// USB disconnect
        ErrBusy = 1 << 2,// USB send busy
        ErrWaiting = 1 << 3,// waiting for reply
        ErrTimeOut = 1 << 4,// communication timeout
        ErrDataParse = 1 << 5,// Communication data error
        ErrFailACK = 1 << 6,// return failed parameter
} ErrorType;

typedef enum
{
        REGADDR_NONE = 0,// address is not sent
        REGADDR_8Bit = 1,// send 8-bit address
        REGADDR_16Bit = 2,// send 16-bit address
        REGADDR_24Bit = 3,// send 24-bit address
        REGADDR_32Bit = 4,// send 32-bit address
} REGADDR_TYPE;

typedef enum
{
        SINGLEALL = 0,// all stages are single line
        QUADALL = 1,// all stages are four-wire
        QUADDATA = 2,// only four lines in the data phase, other single lines
        SINGLECMD = 3,// instruction phase single line only, other four lines
} QSPI_TYPE;

typedef enum
{
        LOW_1EDG = 0,
```

```
        LOW_2EDG = 1,
        HIGH_1EDG = 2,
        HIGH_2EDG = 3,
} SPICK_TYPE;

typedef enum
{
        ENDIAN_MSB = 0,// high before
        ENDIAN_LSB = 1,// Low before
} SPIFIRSTBIT_TYPE;

typedef enum
{
        FIELD_NONE = 0,// none
        FIELD_ONE,// 1 byte
        FIELD_TWO,// 2 bytes
        FIELD_THREE,// 3 bytes
        FIELD_FOUR,// 4 bytes
} FIELDLEN_TYPE;

typedef enum
{
        INT_NONE = 0,// none
        INT_RISE = 1,// rising edge
        INT_FALL = 2,// falling edge
        INT_HIGH = 3,// high
        INT_LOW = 4,// Low Level
        INT_RISE_FALL = 5,// bilateral edge
} INT_TYPE;

//(JI2C, JSPI)INT pin interrupt callback function type
typedef void (*I2CIntCallbackFun)(void);
typedef void (*SPIIntCallbackFun)(void);
```

Interface Overview

Public API interface

Before operating the device, you need to use DevOpen to open the device
After the device is opened, the device will be occupied. Other processes cannot open the occupied device again until the device is closed by DevClose or the process occupying the device exits.

DevicesScan scans inserted devices, returned as a string

Interface Name	Overview
DevicesScan	View currently connected devices
DevOpen	Turn on the device
DevClose	Shut down the device

JTool-IO API interface

Most functions of the JTool-IO module provide additional_M ending API, which means that it is executed in a bitwise mask mode to facilitate simultaneous execution of multiple pins. For example:
If the lower 4 bits of 0x0f are 1, IO1 ~ IO4 are set at the same time
0x 03 The lower 2 bits are 1, both IO1 and IO2 are set
If some functions are applied to all channels, the API at the end of_M is not provided.

Interface Name	Overview
IOSetNone	Set to empty mode
IOSetNone_m	Set to empty mode (mask multi-channel execution)
IOSetIn	Set to IO input
IOSetIn_m	Set to IO input (mask multi-channel execution)
IOSetOut	Set to IO output
IOSetOut_m	Set to IO output (mask multi-channel execution)
IOSetVal	Set the level of the output
IOSetVal_m	Setting the level of the output (mask multi-channel implementation)
IOSetOutWithVal	Set to IO output while setting the output level
IOSetOutWithVal_m	Set to IO output while setting the output level (mask multi-channel execution)
IOPulseOn	Pulse output normally open

Interface Name	Overview
IOPulseOn_m	Pulse output normally open (mask multi-channel implementation)
IOPulseOff	Pulse output stop
IOPulseOff_m	Pulse output stop (mask multi-channel execution)
IOPulseCnt	Output fixed pulse number
IOPulseCnt_m	Output fixed number of pulses (mask multi-channel execution)
IOPulseFreq	Set pulse output frequency
IOPulseFreq_m	Set pulse output frequency (mask multi-channel implementation)
PWMSetFreq	Set PWM output frequency
PWMSetOut	Set to PWM output
PWMSetOut_m	Set to PWM output (mask multi-channel execution)
PWMSetOn	Turn on PWM output
PWMSetOn_m	Turn on PWM output (mask multi-channel execution)
PWMSetOff	Stop PWM output
PWMSetOff_m	Stop PWM output (mask multi-channel execution)
PWMSetDuty	Set PWM duty cycle
PWMSetDuty_m	Set PWM duty cycle (mask multi-channel execution)
CapSetIn	Set to PWM capture
CapClearCnt	Clear Capture Pulse Count
ADCSetIn	Set to ADC acquisition
ADCSetIn_m	Set to ADC acquisition (mask multi-channel execution)
ADCSetSamp	Set the ADC sampling rate
IOGetInVal	Get IO input level value

Interface Name	Overview
IOGetInVal_m	Get IO input level value (mask multi-channel execution)
IOGetPulseRemain	Get Pulse Output Remaining Count
IOGetPulseRemain_m	Get pulse output remaining count (mask multi-channel implementation)
CapGetVal	Get PWM capture value
ADCGetVal	Get ADC sample values
ADCGetVal_m	Get ADC sample values (mask multi-channel execution)
JIOReboot	Restart the JIO device
JIOSetVcc	Set the JIO VCC output voltage
JIOSetVio	Set the JIO VIO level
JIOSetID	Set the ID of the JIO device.
JIOIntoBoot	Restart JIO and enter bootloader

JTool-I2C API interface

Interface Name	Overview
I2CScan	Scan the I2C slave address
I2CWrite	I2C write data
I2CRead	I2C read data
I2CReadWithDelay	I2C read data (with delay)
EEWrite	EEPROM write data
EERead	EEPROM read data
I2CRegisterIntCallback	Register interrupt callback function
I2CCloseIntCallback	Close interrupt callback function
JI2CReboot	Restart the JI2C device

Interface Name	Overview
JI2CSetVcc	Set the JI2C VCC output voltage
JI2CSetVio	Set the JI2C VIO level
JI2CSetSpeed	Set JI2C communication rate
JI2CSetID	Set the ID of the JI2C device.
JI2CIntoBoot	Restart JI2C and enter bootloader

JTool-SPI API interface

Interface Name	Overview
SPIWriteOnly	SPI write-only data
SPIReadOnly	SPI read data only
SPIWriteRead	SPI write and read data at the same time
QSPIWriteOnly	QSPI write-only data
QSPIReadOnly	QSPI read data only
SPIWriteWithCMD	SPI writes data with instructions (CMD, ADDR, ALT, DUMMY)
SPIReadWithCMD	SPI read data with instructions (CMD, ADDR, ALT, DUMMY)
SPIRegisterIntCallback	Register interrupt callback function
SPICloseIntCallback	Close interrupt callback function
JSPIReboot	Restart the JSPI device
JSPISetVcc	Setting the JSPI VCC output voltage
JSPISetVio	Set the JSPI VIO level
JSPISetSpeed	Set the JSPI communication rate
JSPISetID	Set the JSPI device ID.
JSPIIntoBoot	Restart JSPI and enter bootloader

-------- Public Interface--------

DevicesScan

View currently connected devices

Syntax

```
char* DevicesScan (
    int  DevType
    int* OutCnt
    );
```

Parameters

`[in] DevType` device type, see dev_type_enum enumeration value

`[out] OutCnt` returns the number of devices scanned

Return Value

The scanned device, as a string, if multiple devices are scanned, split by \r\n

DevOpen

Open Device

Syntax

```
void* DevOpen (
    int DevType
    char* Sn
    int Id
    );
```

Parameters

`[in] DevType` device type, see dev_type_enum enumeration value

`[in] Sn` specify the device SN to open, string type, if not specified, please use NULL

`[in] Id` specifies the ID of the device to be opened. The value range is from 0 to 65535. If not specified, use -1.

Return Value

If the device is opened successfully, the device handle (that is, the device pointer) is

returned. This handle is required for subsequent operations.
If open fails, return NULL

DevClose

Shut down device

Syntax

```
BOOL DevClose (
    void* DevHandle
    );
```

Parameters

`[in] DevHandle` device handle (I. E. Device pointer) that needs to be closed

Return Value

Returns TRUE if the close was successful
If the close fails, return FALSE

-------- JIO interface--------

IOSetNone

Single pin set to null mode

Syntax

```
ErrorType IOSetNone (
    void* DevHandle
    uint32_t ionum
    );
```

Parameters

`[in] DevHandle` device Handle
`[in] ionum` single pin serial number (starting from 1)

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

IOSetNone_m

Multiple pins set to null mode (mask multi-channel execution)

Syntax

```
ErrorType IOSetNone_m (
    void* DevHandle
    uint32_t iomask
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOSetIn

Single pin set to IO input

Syntax

```
ErrorType IOSetIn (
    void* DevHandle
    uint32_t ionum
    BOOL  pullup
    BOOL pulldown
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[in] pullup` whether to pull up

`[in] pulldown` drop-down

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOSetIn_m

Multiple pins set to IO input (mask multi-channel implementation)

Syntax

```
ErrorType IOSetIn_m (
    void* DevHandle
    uint32_t iomask
    uint32_t pullups
    uint32_t pulldowns
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[in] pullups` pull up (bitwise enable)

`[in] pulldowns` whether to pull down (bitwise enable)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOSetOut

Single pin set to IO output

Syntax

```
ErrorType IOSetOut (
    void* DevHandle
    uint32_t ionum
    BOOL pp
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[in] pp` whether push-pull output

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOSetOut_m

Multiple pins set to IO output (mask multi-channel implementation)

Syntax

```
ErrorType IOSetOut_m (
    void* DevHandle
    uint32_t iomask
    uint32_t pps
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[in] pps` whether push-pull output (bit-enabled)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOSetVal

A single pin sets the level of the output

Syntax

```
ErrorType IOSetVal (
    void* DevHandle
    uint32_t ionum
    BOOL val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[in] val` whether to output high level

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOSetVal_m

Multiple pins to set the level of the output (mask multi-channel implementation)

Syntax

```
ErrorType IOSetVal_m (
    void* DevHandle
    uint32_t iomask
    uint32_t vals
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[in] vals` whether to output high level (bitwise enable)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOSetOutWithVal

A single pin is set to IO output while setting the output level

Syntax

```
ErrorType IOSetOutWithVal (
    void*  DevHandle
    uint32_t ionum
    BOOL pp
    BOOL val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[in] pp` whether push-pull output

`[in] val` whether to output high level

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOSetOutWithVal_m

Multiple pins are set to IO output while setting the output level (mask multi-channel execution)

Syntax

```
ErrorType IOSetOutWithVal_m (
    void* DevHandle
    uint32_t iomask
    uint32_t pps
    uint32_t vals
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[in] pps` whether push-pull output (bit-enabled)

`[in] vals` whether to output high level (bitwise enable)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOPulseOn

Single pin pulse output normally on (pulse output must first be set as output using IOSetOut or IOSetOut_m)

Syntax

```
ErrorType IOPulseOn (
    void* DevHandle
    uint32_t ionum
    uint32_t freq
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[in] freq` pulse frequency (refer to JIO drop-down box for optional values)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOPulseOn_m

Multi-pin pulse output normally on (mask multi-channel implementation) (pulse output requires the pin to be set as output using IOSetOut or IOSetOut_m)

Syntax

```
ErrorType IOPulseOn_m (
    void* DevHandle
    uint32_t iomask
    uint32_t* freqs
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[in] freqs` pulse frequency array (please pass in the 32-Channel uint32_t array and fill in the required values in the corresponding positions) (for optional values, see The JIO drop-down box)

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

IOPulseOff

Single pin pulse output stop

Syntax

```
ErrorType IOPulseOff (
    void* DevHandle
    uint32_t ionum
    );
```

Parameters

`[in] DevHandle` device Handle
`[in] ionum` single pin serial number (starting from 1)

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

IOPulseOff_m

Multi-pin pulse output stop (mask multi-channel implementation)

Syntax

```
ErrorType IOPulseOff_m (
    void* DevHandle
    uint32_t iomask
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOPulseCnt

Single pin outputs a fixed number of pulses (pulse output requires the pin to be set as output first using IOSetOut or IOSetOut_m)

Syntax

```
ErrorType IOPulseCnt (
    void* DevHandle
    uint32_t ionum
    uint32_t cnt
    uint32_t freq
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[in] cnt` number of pulses

`[in] freq` pulse frequency (refer to JIO drop-down box for optional values)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOPulseCnt_m

Multiple pins output a fixed number of pulses (mask multi-channel implementation) (pulse output needs to use IOSetOut or IOSetOut_m to set the pin as output first)

Syntax

```
ErrorType IOPulseCnt_m (
    void* DevHandle
    uint32_t iomask
    uint32_t* cnts
    uint32_t* freqs
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[in] cnts` pulse quantity array (please pass in a 32-Channel uint32_t type array and fill in the required values in the corresponding positions)

`[in] freqs` pulse frequency array (please pass in the 32-Channel uint32_t array and fill in the required values in the corresponding positions) (for optional values, see The JIO drop-down box)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOPulseFreq

Single pin sets pulse output frequency

Syntax

```
ErrorType IOPulseFreq (
    void* DevHandle
    uint32_t ionum
    uint32_t freq
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[in] freq` pulse frequency (refer to JIO drop-down box for optional values)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOPulseFreq_m

Multiple pins set pulse output frequency (mask multi-channel implementation)

Syntax

```
ErrorType IOPulseFreq_m (
    void* DevHandle
    uint32_t iomask
    uint32_t* freqs
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[in] freqs` pulse frequency array (please pass in the 32-Channel uint32_t array and fill in the required values in the corresponding positions) (for optional values, see The JIO drop-down box)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

PWMSetFreq

Set PWM output frequency (applies to all PWM channels)

Syntax

```
ErrorType PWMSetFreq (
    void* DevHandle
    uint32_t freq
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] freq` PWM frequency (refer to the JIO drop-down box for optional values)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

PWMSetOut

Single pin set to PWM output

Syntax

```
ErrorType PWMSetOut (
    void* DevHandle
    uint32_t ionum
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

PWMSetOut_m

Multiple pins set to PWM output (mask multi-channel implementation)

Syntax

```
ErrorType PWMSetOut_m (
    void* DevHandle
    uint32_t iomask
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

PWMSetOn

Single pin turns on PWM output

Syntax

```
ErrorType PWMSetOn (
    void* DevHandle
    uint32_t ionum
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

PWMSetOn_m

Multiple pins on PWM output (mask multi-channel implementation)

Syntax

```
ErrorType PWMSetOn_m (
    void* DevHandle
    uint32_t iomask
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

PWMSetOff

Single pin stop PWM output

Syntax

```
ErrorType PWMSetOff (
    void* DevHandle
    uint32_t ionum
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

PWMSetOff_m

Multiple pins stop PWM output (mask multi-channel implementation)

Syntax

```
ErrorType PWMSetOff_m (
    void* DevHandle
    uint32_t iomask
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

PWMSetDuty

Single pin set PWM duty cycle

Syntax

```
ErrorType PWMSetDuty (
    void* DevHandle
    uint32_t ionum
    uint16_t duty
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[in] duty` PWM duty cycle (0~1000 corresponding duty cycle 0% ~ 100)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

PWMSetDuty_m

Multiple pins set PWM duty cycle (mask multi-channel implementation)

Syntax

```
ErrorType PWMSetDuty_m (
    void* DevHandle
    uint32_t iomask
    uint16_t* dutys
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[in] dutys` PWM duty cycle array (please pass in a 32-Channel uint16_t array and fill in the required values in the corresponding positions)(0~1000 corresponds to a duty cycle of 0% ~ 100)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

CapSetIn

Set to PWM capture (Channel 1 only)

Syntax

```
ErrorType CapSetIn (
    void* DevHandle
    uint32_t ionum
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

CapClearCnt

Clear Capture Pulse Count (only supported on channel 1)

Syntax

```
ErrorType CapClearCnt (
    void* DevHandle
    uint32_t ionum
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

ADCSetIn

Single pin set for ADC acquisition

Syntax

```
ErrorType ADCSetIn (
    void* DevHandle
    uint32_t ionum
    BOOL isdiff
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[in] isdiff` is it differential (1,3 channels support differential)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

ADCSetIn_m

Multiple pins set for ADC acquisition (mask multi-channel implementation)

Syntax

```
ErrorType ADCSetIn_m (
    void* DevHandle
    uint32_t iomask
    uint32_t isdiffs
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[in] isdiffs` is it differential (bitwise enable)(1,3 channels support differential)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

ADCSetSamp

Set the ADC sampling rate (applies to all ADC channels)

Syntax

```
ErrorType ADCSetSamp (
    void* DevHandle
    uint32_t samp
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] samp` ADC sampling rate

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOGetInVal

Single pin to get IO input level value

Syntax

```
ErrorType IOGetInVal (
    void* DevHandle
    uint32_t ionum
    BOOL* val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[out] val` the level value returned, 0 represents low level, 1 represents high level

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOGetInVal_m

Multiple pins get IO input level values (mask multi-channel execution)

Syntax

```
ErrorType IOGetInVal_m (
    void* DevHandle
    uint32_t iomask
    uint32_t* vals
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] iomask` multiple pin bit numbers (bitwise enable)

`[out] vals` the returned level value, (please pass in a single pointer of uint32_t type) (judge the high and low levels by bits)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

IOGetPulseRemain

Single pin to get pulse output residual count

Syntax

```
ErrorType IOGetPulseRemain (
    void* DevHandle
    uint32_t ionum
    uint32_t* remaincnt
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)
`[out] remaincnt` returns the number of remaining pulses for a single channel (pass in a single pointer of type uint32_t)

Return Value
If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

IOGetPulseRemain_m

Multiple Pins Get Pulse Output Remaining Count (mask Multi-Channel Execution)

Syntax

```
ErrorType IOGetPulseRemain_m (
    void* DevHandle
    uint32_t iomask
    uint32_t* remaincnts
    );
```

Parameters
`[in] DevHandle` device Handle
`[in] iomask` multiple pin bit numbers (bitwise enable)
`[out] remaincnts` return the number of remaining pulses of multiple channels, (please pass in an array of uint32_t type of 32 channels, the number of remaining pulses of the corresponding channel will be returned)

Return Value
If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

CapGetVal

Get PWM capture value (only supported on channel 1)

Syntax

```
ErrorType CapGetVal (
    void* DevHandle
    uint32_t ionum
    uint32_t* freq
    uint16_t* duty
    uint32_t* pulsecnt
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[out] freq` returns the captured frequency (pass in a single pointer of type uint32_t)

`[out] duty` returns the captured duty cycle (pass in a single pointer of type uint16_t)

`[out] pulsecnt` returns the number of captured pulses (pass in a single pointer of type uint32_t)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

ADCGetVal

Single pin to get ADC sample value

Syntax

```
ErrorType ADCGetVal (
    void* DevHandle
    uint32_t ionum
    uint16_t* adval
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] ionum` single pin serial number (starting from 1)

`[out] adval` returns a single channel ADC sample (pass in a single pointer of type uint16_t)

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

ADCGetVal_m

Multiple pins to get ADC sample values (mask multi-channel execution)

Syntax

```
ErrorType ADCGetVal_m (
    void* DevHandle
    uint32_t iomask
    uint16_t* advals
    );
```

Parameters

`[in] DevHandle` device Handle
`[in] iomask` multiple pin bit numbers (bitwise enable)
`[out] advals` return ADC sample values of multiple channels (pass in a 32-channel array of uint16_t type to return ADC sample values of the corresponding channels)

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

JIOReboot

Restart the JIO device

Syntax

```
ErrorType JIOReboot (
    void* DevHandle
    );
```

Parameters

`[in] DevHandle` device Handle

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JIOSetVcc

Setting the JIOVCC output voltage

Syntax

```
ErrorType JIOSetVcc (
    void* DevHandle
    uint8_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` VCC option enumeration value, different models may support different, specific reference JIO interface drop-down box (for example: 0 - 5V; 1 - 3.3V; 2-off)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JIOSetVio

Set the JIOVIO level

Syntax

```
ErrorType JIOSetVio (
    void* DevHandle
    uint8_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` VIO option enumeration value. Different models may support different values. For details, please refer to the JIO interface drop-down box (for example: 0 - 3.3V; 1 - 1.8V).

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

JIOSetID

Set the ID of the JIO device.

Syntax

```
ErrorType JIOSetID (
    void* DevHandle
    uint16_t val
    );
```

Parameters

`[in] DevHandle` device Handle
`[in] val` ID value (0~65535)

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

JIOIntoBoot

Restart JIO and enter bootloader

Syntax

```
ErrorType JIOIntoBoot (
    void* DevHandle
    );
```

Parameters

`[in] DevHandle` device Handle

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

-------- JI2C interface--------

I2CScan

Scan the I2C slave address

Syntax

```
ErrorType I2CScan (
    void* DevHandle
    uint8_t* cnt
    uint8_t* result
    );
```

Parameters

`[in] DevHandle` device Handle

`[out] cnt` returns the number of scanned slave addresses

`[out] result` returns the scanned slave address (be sure to pass in 128 arrays of uint8_t type, and the scanned address will be stored in this array)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

I2CWrite

I2C write data

Syntax

```
ErrorType I2CWrite (
    void* DevHandle
    uint8_t slave_addr,
    REGADDR_TYPE reg_type,
    uint32_t reg_addr,
    uint16_t len
    uint8_t* data
    );
```

Parameters

`[in] DevHandle` device Handle
`[in] slave_addr` slave address (8-bit mode)
`[in] reg_type` register address type, refer to REGADDR_TYPE enumeration value
`[in] reg_addr` register Address
`[in] len` write Length
`[in] data` array of data written

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

I2CRead

I2C read data

Syntax

```
ErrorType I2CRead (
    void* DevHandle
    uint8_t slave_addr,
    REGADDR_TYPE reg_type,
    uint32_t reg_addr,
    uint16_t len
    uint8_t* buf
    );
```

Parameters

`[in] DevHandle` device Handle
`[in] slave_addr` slave address (8-bit mode)
`[in] reg_type` register address type, refer to REGADDR_TYPE enumeration value
`[in] reg_addr` register Address
`[in] len` read Length
`[out] buf` read the array in which the data is stored (make sure the array is at least = = len)

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

I2CReadWithDelay

I2C read data (with delay)

Syntax

```
ErrorType I2CReadWithDelay (
    void* DevHandle
    uint8_t slave_addr,
    REGADDR_TYPE reg_type,
    uint32_t reg_addr,
    uint16_t len
    uint8_t*  buf
    uint8_t sr_delay,
    uint8_t raddr_delay
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] slave_addr` slave address (8-bit mode)

`[in] reg_type` register address type, refer to REGADDR_TYPE enumeration value

`[in] reg_addr` register Address

`[in] len` read Length

`[out] buf` read the array in which the data is stored (make sure the array is at least = = len)

`[in] sr_delay` read plus delay wait (before Sr repeat start condition)

`[in] raddr_delay` read plus delay wait (after sending the read address)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

EEWrite

EEPROM write data

Syntax

```
ErrorType EEWrite (
    void* DevHandle
    uint8_t base_slave_addr,
    REGADDR_TYPE reg_type,
    uint16_t page_size,
    uint32_t reg_addr,
    uint32_t len
    uint8_t* data
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] base_slave_addr` base slave address (EEPROM with multiple blocks has multiple slave addresses, please use the first one)(8-bit mode)

`[in] reg_type` register address type, refer to REGADDR_TYPE enumeration value (EEPROM can only be 8 bits or 16 bits, refer to the specific model)

`[in] page_size` page size of EEPROM (make sure this parameter is correct for cross-page processing)

`[in] reg_addr` register Address

`[in] len` write Length

`[in] data` array of data written

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

EERead

EEPROM read data

Syntax

```
ErrorType EERead (
    void* DevHandle
    uint8_t base_slave_addr,
    REGADDR_TYPE reg_type,
    uint32_t reg_addr,
    uint32_t len
    uint8_t* buf
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] base_slave_addr` base slave address (EEPROM with multiple blocks has multiple slave addresses, please use the first one)(8-bit mode)

`[in] reg_type` register address type, refer to REGADDR_TYPE enumeration value (EEPROM can only be 8 bits or 16 bits, refer to the specific model)

`[in] reg_addr` register Address

`[in] len` read Length

`[out] buf` read the array in which the data is stored (make sure the array is at least = = len)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

I2CRegisterIntCallback

Register interrupt callback function for I2C INT pin

Syntax

```
ErrorType I2CRegisterIntCallback (
    void* DevHandle
    INT_TYPE inttype
    I2CIntCallbackFun callback)
```

Parameters

`[in] DevHandle` device Handle

`[in] inttype` interrupt trigger type, refer to INT_TYPE enumeration value

`[in] callback` interrupt callback function pointer, custom interrupt function (parameter

and return value are void), bring the function name into this parameter, this function will be executed when the interrupt is triggered

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

I2CCloseIntCallback

Interrupt callback function to close I2C INT pin

Syntax

```
ErrorType I2CCloseIntCallback(void* DevHandle)
```

Parameters

`[in] DevHandle` device Handle

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

JI2CReboot

Restart the JI2C device

Syntax

```
ErrorType JI2CReboot (
    void* DevHandle
    );
```

Parameters

`[in] DevHandle` device Handle

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

JI2CSetVcc

Setting the JI2CVCC output voltage

Syntax

```
ErrorType JI2CSetVcc (
    void* DevHandle
    uint8_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` VCC option enumeration value, different models may support different, specific reference JI2C interface drop-down box (for example: 0 - 5V; 1 - "= VIO" ; 2-off)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JI2CSetVio

Set the JI2CVIO level

Syntax

```
ErrorType JI2CSetVio (
    void* DevHandle
    uint8_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` VIO option enumeration value, different models may support different, specific reference JI2C interface drop-down box (for example: 0 - 3.3V; 1 - 1.8V)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JI2CSetSpeed

Set JI2C communication rate

Syntax

```
ErrorType JI2CSetSpeed (
    void* DevHandle
    uint8_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` I2C rate option enumeration value, different models may support different, specific reference JI2C interface drop-down box (for example: 0 - 10K; 1 - 50K)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JI2CSetID

Set the ID of the JI2C device.

Syntax

```
ErrorType JI2CSetID (
    void* DevHandle
    uint16_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` ID value (0~65535)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JI2CIntoBoot

Restart JI2C and enter bootloader

Syntax

```
ErrorType JI2CIntoBoot (
    void* DevHandle
    );
```

Parameters

`[in] DevHandle` device Handle

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

-------- JSPI interface--------

SPIWriteOnly

SPI write-only data

Syntax

```
ErrorType SPIWriteOnly (
    void* DevHandle
    SPICK_TYPE ck
    SPIFIRSTBIT_TYPE firstbit
    uint32_t len
    uint8_t* dataw)
```

Parameters

`[in] DevHandle` device Handle

`[in] ck` clock type, refer to SPICK_TYPE enumeration value

`[in] firstbit` bit order (MSB,LSB), reference SPIFIRSTBIT_TYPE enumeration value

`[in] len` write Length

`[in] dataw` array of data written

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

SPIReadOnly

SPI read data only

Syntax

```
ErrorType SPIReadOnly (
    void* DevHandle
    SPICK_TYPE ck
    SPIFIRSTBIT_TYPE firstbit
    uint32_t len
    uint8_t* bufr)
```

Parameters

`[in] DevHandle` device Handle

`[in] ck` clock type, refer to SPICK_TYPE enumeration value

`[in] firstbit` bit order (MSB,LSB), reference SPIFIRSTBIT_TYPE enumeration value

`[in] len` read Length

`[out] bufr` read the array in which the data is stored (make sure the array is at least = = len)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

SPIWriteRead

SPI write and read data at the same time

Syntax

```
ErrorType SPIWriteRead (
    void* DevHandle
    SPICK_TYPE ck
    SPIFIRSTBIT_TYPE firstbit
    uint32_t len
    uint8_t* dataw
    uint8_t* bufr)
```

Parameters

`[in] DevHandle` device Handle

`[in] ck` clock type, refer to SPICK_TYPE enumeration value

`[in] firstbit` bit order (MSB,LSB), reference SPIFIRSTBIT_TYPE enumeration value

`[in] len` read and write length

`[in] dataw` array of data written

`[out] bufr` read the array in which the data is stored (make sure the array is at least = = len)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

QSPIWriteOnly

QSPI write-only data

Syntax

```
ErrorType QSPIWriteOnly (
    void* DevHandle
    SPICK_TYPE ck
    SPIFIRSTBIT_TYPE firstbit
    uint32_t len
    uint8_t* dataw)
```

Parameters

`[in] DevHandle` device Handle

`[in] ck` clock type, refer to SPICK_TYPE enumeration value

`[in] firstbit` bit order (MSB,LSB), reference SPIFIRSTBIT_TYPE enumeration value

`[in] len` write Length

`[in] dataw` array of data written

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

QSPIReadOnly

QSPI read data only

Syntax

```
ErrorType QSPIReadOnly (
    void* DevHandle
    SPICK_TYPE ck
    SPIFIRSTBIT_TYPE firstbit
    uint32_t len
    uint8_t* bufr)
```

Parameters

`[in] DevHandle` device Handle

`[in] ck` clock type, refer to SPICK_TYPE enumeration value

`[in] firstbit` bit order (MSB,LSB), reference SPIFIRSTBIT_TYPE enumeration value

`[in] len` read Length

`[out] bufr` read the array in which the data is stored (make sure the array is at least = = len)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

SPIWriteWithCMD

SPI writes data with instructions (CMD, ADDR, ALT, DUMMY)

Syntax

```
ErrorType SPIWriteWithCMD (
        void* DevHandle
        SPICK_TYPE ck
        SPIFIRSTBIT_TYPE firstbit
        QSPI_TYPE qspitype
        FIELDLEN_TYPE cmdtype
        uint32_t  cmd
        FIELDLEN_TYPE addrtype
        uint32_t addr
        FIELDLEN_TYPE alttype
        uint32_t  alt
        FIELDLEN_TYPE dummytype
        uint32_t len
        uint8_t* dataw)
```

Parameters

`[in] DevHandle` device Handle
`[in] ck` clock type, refer to SPICK_TYPE enumeration value
`[in] firstbit` bit order (MSB,LSB), reference SPIFIRSTBIT_TYPE enumeration value
`[in] qspitype` spi/qspi combination type, reference QSPI_TYPE enumeration value
`[in] cmdtype` cmd bytes, reference FIELDLEN_TYPE enumeration value
`[in] cmd` cmd
`[in] addrtype` addr Bytes, reference FIELDLEN_TYPE enumeration value
`[in] addr` addr
`[in] alttype` alt bytes, reference FIELDLEN_TYPE enumeration value
`[in] alt` alt
`[in] dummytype` number of dummy bytes, reference FIELDLEN_TYPE enumeration value
`[in] len` write Length
`[in] dataw` array of data written

Return Value

If the operation succeeds, returns ErrNone
If the operation fails, return an error code reference ErrorType enumeration value

SPIReadWithCMD

SPI read data with instructions (CMD, ADDR, ALT, DUMMY)

Syntax

```
ErrorType SPIReadWithCMD (
        void* DevHandle
        SPICK_TYPE ck
        SPIFIRSTBIT_TYPE firstbit
        QSPI_TYPE qspitype
        FIELDLEN_TYPE cmdtype
        uint32_t  cmd
        FIELDLEN_TYPE addrtype
        uint32_t addr
        FIELDLEN_TYPE alttype
        uint32_t  alt
        FIELDLEN_TYPE dummytype
        uint32_t len
        uint8_t* bufr)
```

Parameters

`[in] DevHandle` device Handle

`[in] ck` clock type, refer to SPICK_TYPE enumeration value

`[in] firstbit` bit order (MSB,LSB), reference SPIFIRSTBIT_TYPE enumeration value

`[in] qspitype` spi/qspi combination type, reference QSPI_TYPE enumeration value

`[in] cmdtype` cmd bytes, reference FIELDLEN_TYPE enumeration value

`[in] cmd` cmd

`[in] addrtype` addr Bytes, reference FIELDLEN_TYPE enumeration value

`[in] addr` addr

`[in] alttype` alt bytes, reference FIELDLEN_TYPE enumeration value

`[in] alt` alt

`[in] dummytype` number of dummy bytes, reference FIELDLEN_TYPE enumeration value

`[in] len` read Length

`[out] bufr` read the array in which the data is stored (make sure the array is at least = = len)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

SPIRegisterIntCallback

Register the interrupt callback function for the SPI INT pin

Syntax

```
ErrorType SPIRegisterIntCallback (
    void* DevHandle
    INT_TYPE inttype
    SPIIntCallbackFun callback)
```

Parameters

`[in] DevHandle` device Handle

`[in] inttype` interrupt trigger type, refer to INT_TYPE enumeration value

`[in] callback` interrupt callback function pointer, custom interrupt function (parameter and return value are void), bring the function name into this parameter, this function will be executed when the interrupt is triggered

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

SPICloseIntCallback

Interrupt callback function to close SPI INT pin

Syntax

```
ErrorType SPICloseIntCallback(void* DevHandle)
```

Parameters

`[in] DevHandle` device Handle

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JSPIReboot

Restart the JSPI device

Syntax

```
ErrorType JSPIReboot (
    void* DevHandle
    );
```

Parameters

`[in] DevHandle` device Handle

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JSPISetVcc

Setting the JSPIVCC output voltage

Syntax

```
ErrorType JSPISetVcc (
    void* DevHandle
    uint8_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` VCC option enumeration value, different models may support different, specific reference JSPI interface drop-down box (for example: 0 - 5V; 1 - "= VIO" ; 2-off)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JSPISetVio

Setting the JSPIVIO level

Syntax

```
ErrorType JSPISetVio (
    void* DevHandle
    uint8_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` VIO option enumeration value, different models may support different, specific reference JSPI interface drop-down box (for example: 0 - 3.3V; 1 - 1.8V)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JSPISetSpeed

Set the JSPI communication rate

Syntax

```
ErrorType JSPISetSpeed (
    void* DevHandle
    uint8_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` I2C rate option enumeration value, different models may support different, specific reference JSPI interface drop-down box (for example: 0 - 468.75K; 1 - 937.5K)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JSPISetID

Set the JSPI device ID.

Syntax

```
ErrorType JSPISetID (
    void* DevHandle
    uint16_t val
    );
```

Parameters

`[in] DevHandle` device Handle

`[in] val` ID value (0~65535)

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value

JSPIIntoBoot

Restart JSPI and enter bootloader

Syntax

```
ErrorType JSPIIntoBoot (
    void* DevHandle
    );
```

Parameters

`[in] DevHandle` device Handle

Return Value

If the operation succeeds, returns ErrNone

If the operation fails, return an error code reference ErrorType enumeration value