Developer's Guide for Moxa RTU Controllers

Second Edition, December 2011

www.moxa.com/product

Developer's Guide for Moxa RTU Controllers

Copyright Notice

Trademarks

Disclaimer

Information in this document is subject to change without notice and does not represent a commitment on the part of Moxa.

Moxa provides this document as is, without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Moxa reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Moxa assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

Technical Support Contact Information

www.moxa.com/support

Moxa Americas
Toll-free: 1-888-669-2872
Tel: +1-714-528-6777
Fax: +1-714-528-6778

Moxa China (Shanghai office)
Toll-free: 800-820-5036
Tel: +86-21-5258-9955
Fax: +86-21-5258-5505

Moxa Europe
Tel: +49-89-3 70 03 99-0
Fax: +49-89-3 70 03 99-99

Moxa Asia-Pacific
Tel: +886-2-8919-1230
Fax: +886-2-8919-1231

Table of Contents

1

Development Environment Setup

In this chapter we describe how to install a tool chain on the host computer to develop user applications. In addition, the process for performing cross-platform development and debugging is also introduced. For clarity, the MOXA RTU Controller is called a target system.

The following functions are covered in this chapter:

- **Linux Tool Chain**
  - Installing the Linux Tool Chain
  - Compiling Applications
  - On-Line Debugging with GDB

Linux Tool Chain

The Linux tool chain contains a suite of cross compilers and other tools, as well as the libraries and header files that are necessary to compile your applications. These tool chain components must be installed on a Linux-based host computer (PC). The following Linux distributions can be used to install the tool chain.

- Fedora Core 6 (on x86)
- Mandrake 8.1 (on x86)
- Red Hat 7.3, 8.0, 9.0 (on x86)
- SuSE 7.3 (on x86)
- YellowDog 2.1 (on PowerPC)
- Solaris 7 and 8 (on Sparc)
- Debian 3.1, 4.0 (on x86)
- Ubuntu 9.04. (see note)

NOTE Ubuntu users will need to prepare their system by entering the following commands:

```
apt-get install libncurse5-dev
mkdir /mnt/ramdisk
```

Disregard the "[==: unexpected operator" warning when installing the tool chain.

Installing the Linux Tool Chain

The tool chain requires approximately 1 GB of hard disk space. To install the tool chain, follow the steps.

1. Insert the Documentation and Software CD into your PC, and then enter the following commands:

   #mount /dev/cdrom /mnt/cdrom
   #sh /mnt/cdrom/Software/toolchain/arm-linux_3.3.2_V1.X_BuildXXXXXXXX.sh

2. Wait for the installation process to complete. This should take a few minutes.
3. Add the directory **/usr/local/arm-linux/bin** to your path. You can do this in the current login by issuing the following commands:

   #export PATH="/usr/local/arm-linux/bin:$PATH"

Alternatively, the same commands can be added to **$HOME/.bash_profile** to make it effective for all login sessions.

Compiling Applications

To compile a simple C application, use the cross compiler instead of the regular compiler:

```
#arm-linux-gcc –o example –Wall –g –O2 example.c
#arm-linux-strip –s example
#arm-linux-gcc -ggdb –o example-debug example.c
```

Most of the cross compiler tools are the same as their native compiler counterparts, only with an additional prefix that specifies the target system. The prefix is "i386-linux-" for x86 environments and "arm-linux-" for MOXA RTU controllers. For example, "gcc" is the native C compiler, whereas "arm-linux-gcc" is the cross C compiler for the ARM-based ioLogik W5348-C/ioPAC 8020-C series.

Moxa provides cross compiler tools for the following native compilers. Simply add the “arm-linux-” prefix.

ar	Manages archives (static libraries)
as	Assembler
c++, g++	C++ compiler
cpp	C preprocessor
gcc	C compiler
gdb	Debugger
ld	Linker
nm	Lists symbols from object files
objcopy	Copies and translates object files
objdump	Displays information about object files
ranlib	Generates indexes to archives(static libraries)
readelf	Displays information about ELF files
size	Lists object file section sizes
strings	Prints strings of printable characters from files (usually object files)
strip	Removes symbols and sections from object files (usually debugging information)

On-Line Debugging with GDB

The tool chain also provides an on-line debugging mechanism to help you develop your program. Before starting a debugging session, add the option -**ggdb** when you compile the program. A debugging session runs on a client-server architecture on which the server **gdbserver** is installed on the target system and the client **ddd** is installed on the host computer. In the following instructions, we assume that you have uploaded a program named **hello-debug** to the target system and wish to debug this program.

1. Log on to the target system and run the debugging server program.

   #gdbserver 192.168.4.142:2000 hello-debug
   Process hello-debug created; pid=38

   This tells the debugging server to listen for connections on network port 2000 of the network interface 192.168.4.142 of the target system. The name of the program to be debugged is indicated after the network port. Additional arguments can be added after the program name as needed.
2. In the host computer, switch to the directory that contains the program source.

   cd /my_work_directory/myfilesystem/testprograms

3. Execute the client program.

   #ddd --debugger arm-linux-gdb hello-debug &

4. Enter the following command at the GDB, ddd command prompt.

   >> target remote 192.168.4.142:2000

   The command produces a line of output on the target system console, similar to the following.
   Remote debugging using 192.168.4.99:2000

   192.168.4.99 is the host PC's IP address, and 2000 is the port number. You can now begin debugging in the host environment using the interface provided by ddd.

5. Set a break point in the main function by double clicking or entering **b main** on the command line.
6. Click the **cont** button.

2

Programmer's Guide

This chapter includes important information for programmers.

The following topics are covered in this chapter:

- **Flash Memory**
- **C Library**
  - I/O API
  - I/O Alarm API
  - Cellular API
  - SMS API
  - Serial API
  - Modbus Master API
  - Modbus/RTU Master
  - Modbus/TCP Master
  - Modbus/TCP Slave
  - RAS API
  - Misc API

Flash Memory

Partition sizes are hard coded into the kernel binary. The flash memory map is shown in the following table.

	Total Size	Contents	Location	Access
System Space	20 MB	Boot Loader	0x80000000 to 0x80080000	Read Only
		Linux Kernel	0x80080000 to 0x80400000	
		Root File System (JFFS2)	0x80400000 to 0x81400000	
User Space	12 MB	User directory (JFFS2)	0x81400000 to 0x82000000	Read/Write

If the user file system is incorrect, the kernel will change the root file system to the kernel and use the default Moxa file system. To finish the boot process, run the init program.

NOTE

1. The user file system is a complete file system. Users can create and delete directories and files (including source code and executable files) as needed.
2. Users can create the user file system on the host PC or the target platform and copy it to the ioLogik W5348-C series or the ioPAC 8020-C series.
3. Continuously writing data to the flash is not recommended, since doing so will decrease the flash's life.

C Library

The ioLogik W5348-C series and ioPAC 8020-C series both support control devices with Moxa APIs. Users will need to include **libmoxa_pgm.h** to use the following Moxa APIs.

I/O API

The device node is located at **/dev/mxio**. Users must include **libmoxa_pgm.h**. Return values of I/O API functions are shown below; they can also be found in **libmoxa_pgm.h**. In addition, note that **/dev/mxio** cannot be opened again while it is in use.

```
#define IO_SUCCESS                0
#define IO_ERR_DEVICE             0x0001
#define IO_ERR_CMD                0x0002
#define IO_ERR_TYPE               0x0003
#define IO_ERR_ITEM               0x0004
#define IO_ERR_COPY               0x0005
#define IO_ERR_SLOT               0x0006
#define IO_ERR_ARGUMENT           0x0007
#define IO_ERR_RW                 0x0008
#define IO_ERR_ACTION             0x0009
#define IO_ERR_8000_AI_0_10V      0x1000
#define IO_ERR_8000_AI_4_20mA     0x1001
```

Note: "-1" indicates an illegal action.

Digital Input

Status query and mode settings can be done with the following functions. Note that if a DIO channel is configured as a DO, all DI "Set" and "Get" operations will be ignored, and will not generate a response or return.

MX_DI_Mode_Get

UINT32 MX_DI_Mode_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Reads a given DI channel mode. The channel mode will be written to the buf parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
Applies only to the ioLogik W5348-C series.	

MX_DI_Value_Get

UINT32 MX_DI_Value_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Read a given DI channel value. The channel value will be written to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	

MX_DI_Filter_Get

UINT32 MX_DI_Filter_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Reads a given DI channel filter. The channel filter will be written to the *buf* parameter. The default filter is 1 (per 10 milliseconds).	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_CntStart_Get

UINT32 MX_DI_CntStart_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Read counter start flags of appointed DI channels. These counter start flags will be written to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_CntTrigger_Get

UINT32 MX_DI_CntTrigger_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Read counter triggers of appointed DI channels. These counter triggers will be written to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_CntValue_Get

UINT32 MX_DI_CntValue_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Reads counter values of appointed DI channels. These counter values will be written to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_CntOverflow_Get

UINT32 MX_DI_CntOverflow_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Read counter overflow flags of appointed DI channels. These counter overflow flags will be written to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_Mode_Set

UINT32 MX_DI_Mode_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Input Parameters	
slot	
start	
count	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_Filter_Set

UINT32 MX_DI_Filter_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Write a given DI channel filter. The channel filter will be changed according to the *buf* parameter. The default filter is 1 (per 10 milliseconds).	
Input Parameters	
slot	
start	
count	
buf	the unit is milliseconds
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_CntStart_Set

UINT32 MX_DI_CntStart_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Write counter start flags to appointed DI channels. These counter overflow flags will change specific DI channel statuses according to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_CntTrigger_Set

UINT32 MX_DI_CntTrigger_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Write counter triggers to appointed DI channels. These counter triggers will change the status of specific DI channels according to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_CntValue_Set

UINT32 MX_DI_CntValue_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Write counter values to appointed DI channels. These counter values will change the status of specific DI channels according to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DI_CntOverflow_Reset

UINT32 MX_DI_CntOverflow_Reset(UINT32 slot, UINT32 start, UINT32 count);	
Clear counter overflow flags of appointed DI channels.	
Input Parameters	
slot	
start	
count	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

Digital Output

Status and mode query/control can be done with the following functions. Note that if a DIO channel is configured as a DI, all DO “Set” and “Get” operations will be ignored without generating a response or return.

MX_DO_Mode_Get

UINT32 MX_DO_Mode_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Read a given DO channel mode. The channel mode will be written to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_Value_Get

UINT32 MX_DO_Value_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Read a given DO channel value. The channel value will be written to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	

MX_DO_SigW_L_Get

UINT32 MX_DO_SigW_L_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Gets the length of the low edge of the target DO channels. These signal lengths will be written to the *buf* parameter. The default length is 1 unit, 10 ms for DO, and 1500 ms for Relay.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_SigW_H_Get

UINT32 MX_DO_SigW_H_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Gets the length of the high edge of the target DO channels. These signal lengths will be written to the *buf* parameter. The default length is 1 unit, 10 ms for DO, and 1500 ms for Relay.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_PWM_Start_Get

UINT32 MX_DO_PWM_Start_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Reads pulse start flags of appointed DO channels. These pulse start flags will be written to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_PWM_Count_Get

UINT32 MX_DO_PWM_Count_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Read pulse counts of appointed DO channels. These pulse counts will be written to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_Relay_TotalCount_Get

UINT32 MX_DO_Relay_TotalCount_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Read total counts of appointed DO channels. These total counts will be written to the *buf* parameter. Total Counts of Relay cannot be modified after starting.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_Relay_CurrentCount_Get

UINT32 MX_DO_Relay_CurrentCount_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Reads current counts of appointed DO channels. These current counts will be written to the *buf* parameter. Current Counts of Relay can be modified by users at run-time.	
Input Parameters	
slot	
start	
count	
Output Parameters	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_Mode_Set

UINT32 MX_DO_Mode_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Writes a given DO channel mode. The channel mode will be changed according to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_Value_Set

UINT32 MX_DO_Value_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Writes a given DO channel value. The channel value will be changed according to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
buf	
Return Value	
Result of the call	

MX_DO_SigW_L_Set

UINT32 MX_DO_SigW_L_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Sets the length of the low edge to the target DO channels. These signal lengths will change the status of specific DO channels according to the *buf* parameter. the default length is 1 unit, 10 ms for DO, and 1500 ms for Relay.	
Input Parameters	
slot	
start	
count	
buf	The minimum value is 1 (per 10 milliseconds) for DO channels and 1500 milliseconds for Relay channels.
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_SigW_H_Set

UINT32 MX_DO_SigW_H_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Sets the length of the high edge to the target DO channels. These signal lengths will change specific DO channel statuses according to the *buf* parameter. the default length is 1 unit, 10 ms for DO, and 1500 ms for Relay.	
Input Parameters	
slot	
start	
count	
buf	The minimum value is 1 (per 10 milliseconds) for DO channels and 1500 milliseconds for Relay channels.
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_PWM_Start_Set

UINT32 MX_DO_PWM_Start_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Writes pulse start flags to appointed DO channels. These pulse start flags will change specific DO channel statuses according to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_PWM_Count_Set

UINT32 MX_DO_PWM_Count_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Writes pulse counts to appointed DO channels. These pulse counts will change specific DO channel statuses according to the *buf* parameter.	
Input Parameters	
slot	
start	
count	
buf	0 means continuous output
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_DO_Relay_CurrentCount_Set

UINT32 MX_DO_Relay_CurrentCount_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Writes current counts to appointed DO channels. These current counts will change specific DO channel statuses according to the *buf* parameter. Current Counts of Relay can be modified by the user at run-time, but Total Counts of Relay cannot be changed after starting.	
Input Parameters	
slot	
start	
count	
buf	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

Analog Input

MX_AI_Value_RAW_Get

UINT32 MX_AI_Value_RAW_Get(UINT32 slot, UINT32 start, UINT32 count, INT32 *buf);	
Reads a given AI channel raw value. The channel raw value will be written to the *buf* parameter.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
Output Parameters	
buf	
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h.	

MX_AI_Value_ENG_Get

UINT32 MX_AI_Value_ENG_Get(UINT32 slot, UINT32 start, UINT32 count, float *buf);	
Reads a given AI channel engineering value. The channel engineering value will be written to the *buf* parameter.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
Output Parameters	
buf	
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h.	

MX_AI_Range_Setting_Get

UINT32 MX_AI_Range_Setting_Get(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Reads a given AI channel range. The channel range will be written to the *buf* parameter.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
Output Parameters	
buf	
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h	

MX_AI_Min_RAW_Get

UINT32 MX_AI_Min_RAW_Get(UINT32 slot, UINT32 start, UINT32 count, INT32 *buf);	
Reads min raw values of appointed AI channels. These min raw values will be written to the *buf* parameter.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
Output Parameters	
buf	
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h	

MX_AI_Max_RAW_Get

UINT32 MX_AI_Max_RAW_Get(UINT32 slot, UINT32 start, UINT32 count, INT32 *buf);	
Reads max raw values of appointed AI channels. These min raw values will be written to the *buf* parameter.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
Output Parameters	
buf	
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h	

MX_AI_Min_ENG_Get

UINT32 MX_AI_Min_ENG_Get(UINT32 slot, UINT32 start, UINT32 count, float *buf);	
Reads min engineering values of appointed AI channels. These min engineering values will be written to the *buf* parameter.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
Output Parameters	
buf	
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h	

MX_AI_Max_ENG_Get

UINT32 MX_AI_Max_ENG_Get(UINT32 slot, UINT32 start, UINT32 count, float *buf);	
Reads max engineering values of appointed AI channels. These max engineering values will be written to the *buf* parameter.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
Output Parameters	
buf	
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h	

MX_AI_Range_Setting_Set

UINT32 MX_AI_Range_Setting_Set(UINT32 slot, UINT32 start, UINT32 count, UINT32 *buf);	
Write a given AI channel range. The channel range will be changed according to the *buf* parameter.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
buf	150 mV, 500 mV, 0 to 150 mV, 0 to 500 mV, and 0 to 5 V are not supported
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h	

MX_AI_Min_Reset

UINT32 MX_AI_Min_Reset(UINT32 slot, UINT32 start, UINT32 count);	
Sets min raw and engineering values to be current values for appointed AI channels.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h	

MX_AI_Max_Reset

UINT32 MX_AI_Max_Reset(UINT32 slot, UINT32 start, UINT32 count);	
Sets max raw and engineering values to be current values for appointed AI channels.	
Input Parameters	
slot	Slot index of the AI module (zero based). For ioLogik W5348, the slot index is always “0”
start	The first queried AI channel (zero based)
count	The number of queried AI channels
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h	

MX_AI_Range_Signed

UINT32 MX_AI_Range_Signed(UINT32 range);	
Checks if the readings of an AI channel is a positive value or contains positive and negative values.	
Input Parameters	
range	
Return Value	
TRUE (positive and negative) or FALSE (positive only).	
Notes	
ioLogik W5348-C series only	

MX_AI_BurnOut_Value_Set

UINT32 MX_AI_BurnOut_Value_Set(UINT32 slot, UINT32 start, UINT32 count, float *buf);	
To set the current burn-out value of the AI channels (this function is only available in the module RM-3802-T, and only within an AI range of 4-20mA)	
Input Parameters	
slot	Slot index of the AI module (zero based)
start	
count	
buf	
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h.	
Notes	
This function can only be used for the ioPAC RM-3802-T AI module. The burn-out value is only applied while the AI channel range is set to 4-20mA. The valid burn out value is 0.0-4.0 mA If the AI channel range is set to 0-20mA, the burn-out value will not be applied.	

MX_AI_BurnOut_Value_Get

UINT32 MX_AI_BurnOut_Value_Get(UINT32 slot, UINT32 start, UINT32 count, float *buf);	
To get the current burn-out value of the AI channels (this function is only available in the module RM-3802-T, and only within an AI range of 4-20mA)	
Input Parameters	
slot	Slot index of the AI module(zero based)
start	
count	
buf	
Return Value	
To get the complete IO_ERR code list, please refer to the header file libmoxa_pgm.h.	
Notes	
This function can only be used for the ioPAC RM-3802-T AI module. The burn-out value is only applied while the AI channel range is set to 4-20mA. The valid burn out value is 0.0-4.0 mA If the AI channel range is set to 0-20mA, the burn-out value will not be applied.	

Misc I/O

MX_DIO_Map_Get

UINT32 MX_DIO_Map_Get(UINT32 slot, UINT32 *num);	
Gets the current status of DIO channels.	
Input Parameters	
slot	0 = IOLOGIK W5348-C SERIES n = Expansion Devices
Output Parameters	
num	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_Total_Slots_Get

UINT32 MX_Total_Slots_Get(UINT32 *num);	
Gets the number of slots.	
Output Parameters	
num	
Return Value	
Result of the call	
Notes	
ioPAC 8020-C series only	

MX_Slot_Inserted_Get

UINT32 MX_Slot_Inserted_Get(UINT32 *state);	
Gets the current status of inserted slots.	
Output Parameters	
num	
Return Value	
Result of the call	
Notes	
ioPAC 8020-C series only	

MX_Slot_Info_Get

UINT32 MX_Slot_Info_Get (UINT32 slot, UINT32 *moduleType);	
Gets the inserted module type of the slot. If the slot is empty, moduleType returns with value IO_MODULE_EMPTY.	
Input Parameters	
Slot	the slot index, starting from 0
Output Parameters	
moduleType	module type
Return Value	
Result of the call	
Notes	
ioPAC 8020-C series only	

MX_DIO_Map_Set

UINT32 MX_DIO_Map_Set(UINT32 slot, UINT32 num);	
Sets DIO channels to be DI or DO.	
Input Parameters	
slot	0 = IOLOGIK W5348-C SERIES n = Expansion Devices
num	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

I/O Alarm API

The device node is located at **/dev/mxio**. Users must include **libmoxa_pgm.h**. Return values of I/O Alarm API functions are listed below; they can also be found at **libmoxa_pgm.h**. In addition, **/dev/mxio** cannot be opened if it was opened and not closed yet.

```
#define   IO_ALARM_OVERFLOW              -100
#define   IO_ALARM_EXCEPTION             -101
#define   IO_ALARM_INVALID_ACTION        -102
#define   IO_ALARM_INVALID_PARAMETER     -103
#define   IO_ALARM_QUEUE_EMPTY           -104
```

Note: “0” (zero) indicates success, and “-1” indicates an illegal action.

Digital Input Alarm

MX_DI_Alarm_Register

int MX_DI_Alarm_Register(int slot, int channel, int len, int trigger);	
Registers a DI channel for monitoring. If the condition is matched, the event will be recorded with status and time. In addition, each channel can only be registered once.	
Input Parameters	
slot	
channel	
len	Buffer Size: 5 to 20 records
trigger	
Return Value	
Result of the call or a handle.	

MX_DI_Alarm_Unregister

int MX_DI_Alarm_Unregister(int handle);	
Unregisters a DI channel for monitoring and clears the buffer.	
Input Parameters	
handle	
Return Value	
Result of the call	

MX_DI_Alarm_Get

int MX_DI_Alarm_Get(int handle, struct IO_Alarm_Data *data);	
Reads alarm data of the DI channel. This alarm data will be written to the *data* parameter.	
Input Parameters	
handle	
Output Parameters	
data	
Return Value	
Result of the call	

MX_DI_Alarm_Set_Trigger

int MX_DI_Alarm_Set_Trigger(int handle, int trigger);	
Set the condition for the DI channel.	
Input Parameters	
handle	
trigger	
Return Value	
Result of the call	

MX_DI_Alarm_Get_Trigger

int MX_DI_Alarm_Get_Trigger(int handle, int *trigger);	
Gets the condition of the DI channel. The condition will be written to the *trigger* parameter.	
Input Parameters	
handle	
Output Parameters	
trigger	
Return Value	
Result of the call	

MX_DI_Alarm_Count

int MX_DI_Alarm_Count(int handle);	
Gets the number of records of the alarm data in the registered buffer for the DI channel.	
Input Parameters	
handle	
Return Value	
The number of records of the alarm data.	

MX_DI_Alarm_Clear

int MX_DI_Alarm_Clear(int handle);	
Clears the buffer of the DI channel.	
Input Parameters	
handle	
Return Value	
Result of the call	

MX_DI_Alarm_Reset

int MX_DI_Alarm_Reset();
Unregisters all DI channels.
Return Value
Result of the call

Analog Input Alarm

MX_AI_Alarm_Register

int MX_AI_Alarm_Register(int slot, int channel, int len, float value, int condition);	
Register an AI channel for monitoring. If the condition is matched, the event will be recorded with status and time. In addition, each channel can only be registered once.	
Input Parameters	
slot	
channel	
len	Buffer Size: 5 to 20 records
value	
condition	
Return Value	
Result of the call or a handle.	
Notes	
ioLogik W5348-C series only	

MX_AI_Alarm_Unregister

int MX_AI_Alarm_Unregister(int handle);	
Unregisters an AI channel for monitoring and clears the buffer.	
Input Parameters	
handle	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_AI_Alarm_Get

int MX_AI_Alarm_Get(int handle, struct IO_Alarm_Data *data);	
Reads alarm data of the AI channel. This alarm data will be written to the *data* parameter.	
Input Parameters	
handle	
Output Parameters	
data	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_AI_Alarm_Set_Point

int MX_AI_Alarm_Set_Point(int handle, float value, int condition);	
Set conditions for the AI channel.	
Input Parameters	
handle	
value	
condition	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_AI_Alarm_Get_Point

int MX_AI_Alarm_Get_Point(int handle, float *value, int *condition);	
Get conditions of the AI channel. These conditions will be written to the *value* and *condition* parameters.	
Input Parameters	
handle	
Output Parameters	
value	
condition	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_AI_Alarm_Count

int MX_AI_Alarm_Count(int handle);	
Gets the number of records of the alarm data in the registered buffer for the AI channel.	
Input Parameters	
handle	
Return Value	
The number of records of the alarm data.	
Notes	
ioLogik W5348-C series only	

MX_AI_Alarm_Clear

int MX_AI_Alarm_Clear(int handle);	
Clears the buffer of the AI channel.	
Input Parameters	
handle	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

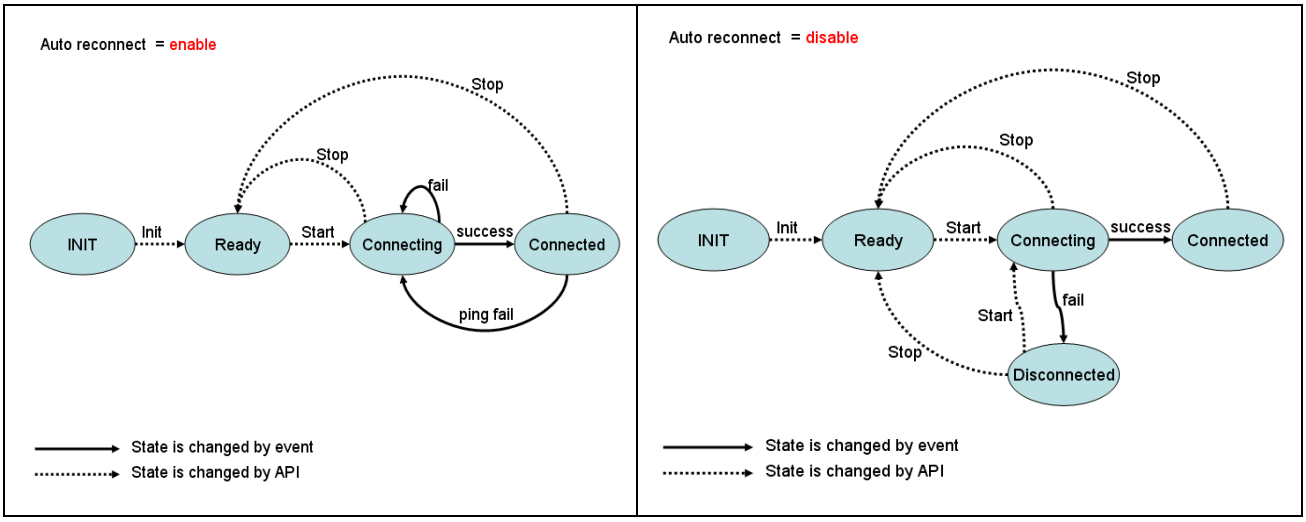
MX_AI_Alarm_Reset

int MX_AI_Alarm_Reset();
Unregisters all AI channels.
Return Value
Result of the call
Notes
ioLogik W5348-C series only

Cellular API

Cellular API functions communicate with the internal cellular modem via two device nodes located at **/dev/ttyACM** and **/dev/ttyUSB0**. Users must include **libmoxa_pgm.h**. Cellular API functions work in two operating modes: Auto Reconnect as enabled or disabled. If Auto Reconnect is enabled, it automatically detects the cellular connection failure via a ping command to a user-specified remote host periodically. Once a cellular connection failure is detected, it will try to re-establish the cellular network without any interaction with the user's program. If auto Reconnect is disabled, the user's program has total control of cellular connection and disconnection; on the other hand, the program must detect the cellular network failure by itself. To apply these two modes, cellular API functions are implemented as a state machine. The states of these two modes are described in figures below.

```
#define MODEM_STATE_INIT          0
#define MODEM_STATE_READY         1
#define MODEM_STATE_CONNECTING    2
#define MODEM_STATE_CONNECTED     3
#define MODEM_STATE_DISCONNECT    4
```



Return values of cellular API functions are shown below; they can also be found at **libmoxa_pgm.h**.

```
#define MODEM_ERR_OK                0//successful
#define MODEM_ERR_INIT              1//not init
#define MODEM_ERR_PARAM             2//invalid parameter
#define MODEM_ERR_OPEN              3//open device error
#define MODEM_ERR_SIM               4//SIM card error
#define MODEM_ERR_PIN               5//PIN code error
#define MODEM_ERR_BAND              6//Set band error
#define MODEM_ERR_ECHO_OFF          7//turn off echo error
#define MODEM_ERR_CELLULAR_DENIED   8//register to cellular network is denied
#define MODEM_ERR_RESET             9//reset modem error
#define MODEM_ERR_THREAD            10//create thread error
#define MODEM_ERR_APN               11//set APN error
#define MODEM_ERR_CREDENTIAL        12//generate PPP credential file error
#define MODEM_ERR_ATTACH            13//attach to 3G network error
#define MODEM_ERR_IF                14//ppp interface is not established
#define MODEM_ERR_CMD               15//err command (wrong state, or previous cmd is executing)
#define MODEM_ERR_TIMEOUT           16//connect to 3G timeout (PPP)
#define MODEM_ERR_STATE             17//invalid cmd state
#define MODEM_ERR_BREAK             18//user break
#define MODEM_ERR_SEM               19//create named semaphore for modem failed
```

MX_Cellular_Modem_Init

INT32 MX_Cellular_Modem_Init(INT8 *pin, UINT32 band);	
Init cellular library; users should invoke this initial function before calling any other cellular API function. On success, the API state will be changed to READY from INIT.	
Input Parameters	
Pin	PIN code of the SIM card (a 4-byte string). If the PIN is empty, use NULL.
Band	Enable/disable the frequency band of the cellular modem. #define MODEM_BAND_HC25_GSM900 1 #define MODEM_BAND_HC25_GSM1800 2 #define MODEM_BAND_HC25_GSM850 4 #define MODEM_BAND_HC25_GSM1900 8 #define MODEM_BAND_HC25_WCDMA2100 16 #define MODEM_BAND_HC25_WCDMA1900 32 #define MODEM_BAND_HC25_WCDMA8500 64 #define MODEM_BAND_HC25_AUTO 127
Return Value	
Returns MODEM_ERR_OK on success. Refer to the returned code table for return values when an error occurs.	
Notes	
ioLogik W5348-C series only	

MX_Cellular_Net_Start

INT32 MX_Cellular_Net_Start(INT8 *apn, INT8 *userName, INT8 *password, CheckInfo* autoCheck);	
Once the initialization is done and the state is MODEM_STATE_READY, users can use this function to establish a cellular connection with Auto Reconnect in Enabled mode or Disabled mode. If Auto Reconnect is enabled, the cellular API will check the cellular connection status periodically and re-establish the cellular connection when the cellular connection is down. If Auto Reconnect is disabled, users should define and check the cellular status manually with their program. On success, the API state will be changed from READY to CONNECTING. Once a cellular connection is established, the state will be changed from CONNECTING to CONNECTED. If any errors occur while establishing the cellular connection, the state will be changed to DISCONNECTED. If Auto Reconnect is disabled, users should monitor the state with a program and call this function to re-establish the cellular connection.	
Input Parameters	
APN	Access point name; should follow the data plan/package
UserName	The user's name; should follow the data plan/package
Password	The password; should follow the data plan/package
AutoCheck	autoCheck: autoCheck → autoCheckEnable: 0:disabled, 1:enable Auto Reconnect mode autoCheck → pingHostname: The IP or the host name of a public server, if autoCheckEnable==1, the cellular API will ping to that server periodically to check the status of the cellular connection. autoCheck → pingIntervalS: the interval time in seconds for detecting the mote host or a public server. autoCheck → pingMaxFail: If the ping timeout occurs successively more times than this settings, the cellular API will re-establish the cellular connection.
Return Value	
Returns MODEM_ERR_OK on success. Refer to the returned code table for return values when an error occurs.	
Notes	
ioLogik W5348-C series only	

MX_Cellular_Net_Stop

INT32 MX_Cellular_Net_Stop(void);	
MX_Cellular_Net_Stop stops the cellular network. On success, the state will be changed to READY.	
Input Parameters	
None.	
Return Value	
Returns MODEM_ERR_OK on success. Refer to the returned code table for return values when an error occurs.	
Notes	
ioLogik W5348-C series only	

MX_Cellular_Net_State

INT32 MX_Cellular_Net_State(void);	
Returns the API state.	
Input Parameters	
None.	
Return Value	
The value of the API state is listed in the header file of the library.	
Notes	
ioLogik W5348-C series only	

MX_Cellular_Modem_RSSI

INT32 MX_Cellular_Modem_RSSI(UINT8 *rssi);	
The user's program can enable this function to retrieve the RSSI. The RSSI value ranges from 0 to 31.	
Input Parameters	
Rssi	Users should provide a UINT8 buffer for retrieving the RSSI value.
Return Value	
Returns MODEM_ERR_OK on success. Refer to the returned code table for return values when an error occurs.	
Notes	
ioLogik W5348-C series only	

MX_Cellular_Modem_Reset

INT32 MX_Cellular_Modem_Reset(void);	
Resets the cellular modem. The user's program should use this function very carefully. The recommended way to use this function is to enable it if Auto Reconnect is disabled, and the API state is in READY or DISCONNECTED. ***Never*** use this function when the API state is in CONNECTING or CONNECTED.	
Input Parameters	
None.	
Return Value	
Returns MODEM_ERR_OK on success. Refer to the returned code table for return values when an error occurs.	
Notes	
ioLogik W5348-C series only	

MX_Cellular_Set_Debug

void MX_Cellular_Set_Debug(UINT8 debug);	
This function is for debugging only. Invoke this function to enable/disable debug messages for the cellular API.	
Input Parameters	
Debug	0: disable 1: enable.
Return Value	
None.	
Notes	
ioLogik W5348-C series only	

SMS API

Mx_SMS_Send_Ascii

INT32 Mx_SMS_Send_Ascii(INT8 *phone, INT8 *pin, INT8 *asciiMsg, INT32 msgLength);	
Sends an SMS message with content in ASCII format.	
Input Parameters	
phone	Phone number in C string format; e.g, “0939008056”
pin	SIM PIN in C string format; string length must be 4 bytes; e.g., “1234”. If the SIM does not have a PIN code, this parameter is ignored.
asciiMsg	User defined SMS ASCII content.
msgLength	The length of asciiMsg in bytes; the maximum length is 160.
Output Parameters	
None	
Notes	
ioLogik W5348-C series only	

Mx_SMS_Send_Ucs2

INT32 Mx_SMS_Send_Ucs2(INT8 *phone, INT8 *pin, INT8 *ucs2Msg, INT8 msgLength);	
Sends an SMS message with unicode content in UCS2 format.	
Input Parameters	
phone	Phone number in C string format; e.g., “0939008056”
pin	SIM PIN in C string format, string length must be 4 bytes; e.g., “1234”. If the SIM does not have a PIN code, this parameter is ignored.
ucs2Msg	User defined SMS content in UCS2 format.
msgLength	The length of ucs2Msg in bytes; the maximum length is 140 bytes.
Output Parameters	
None	
Return Value	
SMS_ERR_OK: Send SMS successful SMS_ERR_LENGTH: Invalid length SMS_ERR_OPEN: Open modem device err SMS_ERR_MODEM: Cannot communicate with the modem SMS_ERR_PIN: No SIM card or PIN error SMS_ERR_PDU: Sets SMS to PDU mode error SMS_ERR_ERR: Sends SMS timeout error	
Notes	
ioLogik W5348-C series only	

Serial API

Device nodes are located at **/dev/ttyMx**. Users must include **libmoxa_pgm.h**. Return values of Serial API functions are shown below; they can also be found at **libmoxa_pgm.h**.

```
#define SERIAL_OK                    0
#define SERIAL_ERROR_FD              -1
#define SERIAL_ERROR_OPEN            -2
#define SERIAL_PARAMETER_ERROR       -3
```

MX_SerialOpen

int MX_SerialOpen(int port);	
Opens a serial port.	
Input Parameters	
port	
Return Value	
Result of the call or a file descriptor.	
Port Mapping	
ioLogik W5348-C series: Port 1 → ttyM0 (COM1) Port 2 → ttyM1 (COM2) ioPAC 8020-C series: Port 1 → ttyM0 (COM)	

MX_SerialWrite

int MX_SerialWrite(int port, char* str, int len);	
Block Write.	
Input Parameters	
port	
str	
len	
Return Value	
Result of the call or number of writing bytes.	

MX_SerialNonBlockRead

int MX_SerialNonBlockRead(int port, char* buf, int len);	
Non-Block Read. Data will be written to the *buf* parameter.	
Input Parameters	
port	
len	
Output Parameters	
buf	
Return Value	
Result of the call or number of bytes that were read.	

MX_SerialBlockRead

int MX_SerialBlockRead(int port, char* buf, int len);	
Block Read. Data will be written to the *buf* parameter.	
Input Parameters	
port	
len	
Output Parameters	
buf	
Return Value	
Result of the call or number of bytes that were read.	

MX_SerialClose

int MX_SerialClose(int port);	
Closes a serial port.	
Input Parameters	
port	
Return Value	
Result of the call	

MX_SerialFlowControl

int MX_SerialFlowControl(int port, int control);	
Sets the Flow Control for the serial port.	
Input Parameters	
port	
control	
Return Value	
Result of the call	

MX_SerialSetSpeed

int MX_SerialSetSpeed(int port, unsigned int speed);	
Set the Baud Rate for the serial port.	
Input Parameters	
port	
speed	
Return Value	
Result of the call	

MX_SerialSetMode

int MX_SerialSetMode(int port, unsigned int mode);	
Set the serial interface (RS232, RS422, or RS485) for the serial port.	
Input Parameters	
port	
mode	
Return Value	
Result of the call	

MX_SerialGetMode

int MX_SerialGetMode(int port);	
Gets the current serial interface for the serial port.	
Input Parameters	
port	
Return Value	
Result of the call or RS232/422/485.	

MX_SerialSetParam

int MX_SerialSetParam(int port, int parity, int databits, int stopbit);	
Sets the parity, data bits, and stop bits for the serial port.	
Input Parameters	
port	
parity	
databits	
stopbit	
Return Value	
Result of the call	

MX_SerialDataInInputQueue

int MX_SerialDataInInputQueue(int port);	
Gets the length of the data in the input queue for the serial port.	
Input Parameters	
port	
Return Value	
Result of the call or the length of the data.	

MX_SerialDataInOutputQueue

int MX_SerialDataInOutputQueue(int port);	
Gets the length of the data in the output queue for the serial port.	
Input Parameters	
port	
Return Value	
Result of the call or the length of the data.	

MX_FindFD

int MX_FindFD(int port);	
Gets the File Descriptor for the serial port.	
Input Parameters	
port	
Return Value	
Result of the call or a file descriptor.	

Modbus Master API

MX_Modbus_Master_Init(void);

INT32 MX_Modbus_Master_Init(void);	
This function, which must be called before you call any other Modbus master API functions, allocates the resources needed for the Modbus master library; to release the allocated resources, call MX_Modbus_Master_Uninit().	
Input Parameters	
None	
Output Parameters	
None	
Return Value	
MODBUS_ERR_OK: Library initialization is done. MODBUS_ERR_LIB_INIT: Library has been initialized.	

MX_Modbus_Master_Uninit(void);

void MX_Modbus_Master_Uninit(void);	
Call this function to release the resources allocated by MX_Modbus_Master_Init().	
Input Parameters	
None	
Output Parameters	
None	
Return Value	
None	

Modbus/RTU Master

MX_Modbus_Rtu_Master_Open

INT32 MX_Modbus_Rtu_Master_Open(UINT32 port, TTY_PARAM *param);	
This function is used to open and configure the external UART port.	
Input Parameters	
port	UART port number. ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
param	Fill this structure parameter to configure baudrate, parity, data-bits, stop-bit, mode, and flow-control for the specific UART port.
Output Parameters	
None	
Return Value	
MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_UART_OPEN MODBUS_ERR_OPENED MODBUS_ERR_UART_MODE MODBUS_ERR_UART_BAUDRATE MODBUS_ERR_UART_FORMAT MODBUS_ERR_UART_FLOW	

MX_Modbus_Rtu_Master_Close (UINT32 port);

INT32 MX_Modbus_Rtu_Master_Close (UINT32 port);	
This function is used to close the opened external UART port.	
Input Parameters	
port	UART port number. ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
Output Parameters	
None	
Return Value	
MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED	

MX_Modbus_Rtu_Master_Read_Coils

INT32 MX_Modbus_Rtu_Master_Read_Coils(UINT32 port, UINT8 unitId, UINT16 startAddr , UINT16 coilCount, UINT8 byteCoils[], UINT32 timeoutMs);	
This function performs Modbus function code 0x1; it reads multiple coils and puts the output.	
Input Parameters	
port	UART port number ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
unitId	Modbus device unit identifier
startAddr	Modbus address
coilCount	The number of read coils
timeoutMs	Timeout value in milliseconds
Output Parameters	
byteCoils	This parameter holds the read coils value; users should provide enough buffer space. For example, if coildCount=9, the buffer size of byteCoils should be at least 2 bytes.
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_UART_WRITE MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP MODBUS_ERR_CRC	

MX_Modbus_Rtu_Master_Write_Coils

INT32 MX_Modbus_Rtu_Master_Write_Coils (UINT32 port, UINT8 unitId, UINT16 startAddr, UINT16 coilCount, UINT8 byteCoils[], UINT32 timeoutMs);	
This function performs Modbus function code 0xF. It writes multiple coils, and the coils values.	
Input Parameters	
port	UART port number ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
unitId	Modbus device unit identifier
startAddr.	Modbus address
coilCount	The numbers of coils
byteCoils	The user should put the coils value in bit-map format
timeoutMs	Timeout value in milliseconds
Output Parameters	
None	
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_UART_WRITE MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP MODBUS_ERR_CRC	

MX_Modbus_Rtu_Master_Write_Coil

INT32 MX_Modbus_Rtu_Master_Write_Coil(UINT32 port, UINT8 unitId, UINT16 addr, UINT8 coil, UINT32 timeOutMs);	
This function performs Modbus function code 0x5; it writes a single coil.	
Input Parameters	
port	UART port number ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
unitId	Modbus device unit identifier
addr	Modbus address
coil	coil value: 0 or 1
timeoutMs	Timeout value in milliseconds
Output Parameters	
None	
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_UART_WRITE MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP	

MX_Modbus_Rtu_Master_Read_Discrete_Inputs

INT32 MX_Modbus_Rtu_Master_Read_Discrete_Inputs(UINT32 port, UINT8 unitId, UINT16 startAddr , UINT16 coilCount, UINT8 byteCoils[], UINT32 timeoutMs);	
This function performs Modbus function code 0x2; it reads multiple discrete inputs.	
Input Parameters	
port	UART port number ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
unitId	Modbus device unit identifier
startAddr	Modbus address
coilCount	The numbers of input coilds
timeoutMs	Timeout value in millisecond
Output Parameters	
byteCoils	This parameter holds the read discrete inputs; users should provide enough buffer space. For example, if coildCount=9, the buffer size of byteCoils should be at least 2 bytes.
Return Value	
MODBUS standard error codes are mapped to values 16 to 24. Refer to header file libmoxa_pgm.h MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_UART_WRITE MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP MODBUS_ERR_CRC	

MX_Modbus_Rtu_Master_Read_Input_Regs

INT32 MX_Modbus_Rtu_Master_Read_Input_Regs(UINT32 port, UINT8 unitId, UINT16 startAddr , UINT16 regCount, UINT16 regs[], UINT32 timeoutMs);	
This function performs Modbus function code 0x4; it reads multiple input registers.	
Input Parameters	
port	UART port number ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
unitId	Modbus device unit identifier
startAddr	Modbus address
regCount	The number of input registers
timeoutMs	Timeout value in milliseconds
Output Parameters	
regs	This parameter holds the read input registers; users should provide enough buffer space. For example, if regCount =4, the buffer size of regs should be at least 8 bytes.
Return Value	
MODBUS standard error codes are mapped to values 16 to 24. Refer to header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_UART_WRITE MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP MODBUS_ERR_CRC	

MX_Modbus_Rtu_Master_Read_Holding_Regs

INT32 MX_Modbus_Rtu_Master_Read_Holding_Regs(UINT32 port, UINT8 unitId, UINT16 startAddr , UINT16 regCount, UINT16 regs[], UINT32 timeoutMs);	
This function performs Modbus function code 0x3; it reads multiple holding registers.	
Input Parameters	
port	UART port number ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
unitId	Modbus device unit identifier
startAddr	Modbus address
regCount	The number of holding registers
timeoutMs	Timeout value in milliseconds
Output Parameters	
regs	This parameter holds the read holding registers; users should provide enough buffer space. For example, if regCount=4, the buffer size of regs should be at least 8 bytes.
Return Value	
MODBUS standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_UART_WRITE MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP MODBUS_ERR_CRC	

MX_Modbus_Rtu_Master_Write_Holding_Regs

INT32 MX_Modbus_Rtu_Master_Write_Holding_Regs (UINT32 port, UINT8 unitId, UINT16 startAddr , UINT16 regCount, UINT16 regs[], UINT32 timeoutMs);	
This function performs Modbus function code 0x10; it writes multiple holding registers.	
Input Parameters	
port	UART port number. ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
unitId	Modbus device unit identifier
startAddr	Modbus address
regCount	The numbers of holding registers.
Regs	This parameter holds the holding registers value; each register occupies 2 bytes of buffer space.
timeoutMs	Timeout value in milliseconds
Output Parameters	
None	
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_UART_WRITE MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP MODBUS_ERR_CRC	

MX_Modbus_Rtu_Master_Write_Holding_Reg

INT32 MX_Modbus_Rtu_Master_Write_Holding_Reg(UINT32 port, UINT8 unitId, UINT16 startAddr , UINT16 reg, UINT32 timeoutMs);	
This function performs Modbus function code 0x06; it writes single holding register.	
Input Parameters	
port	UART port number ioLogik W5348-C series: the port numbers are 0 and 1 ioPAC 8020-C series: the port number is 0
unitId	Modbus device unit identifier
startAddr	Modbus address
Reg	This parameter holds the value of the holding register
timeoutMs	Timeout value in milliseconds
Output Parameters	
None	
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_UART_WRITE MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP	

Modbus/TCP Master

MX_Modbus_Tcp_Master_Open

INT32 MX_Modbus_Tcp_Master_Open(UINT8 ipAddress[], UINT16 tcpPort, UINT32 cTimeoutMs, UINT32 *sHandle);	
This function establishes a TCP connection to the target IP address. If the returned code is MODBUS_ERR_OK, the connection handle sHandle should be closed by the function MX_Modbus_Tcp_Master_Close.	
Input Parameters	
ipAddress	The IP address of the Modbus slave
tcpPort	The TCP listened port of the Modbus master
cTimeoutMs	Modbus address
Output Parameters	
sHandle	Returned connection handle.
Return Value	
MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_SOCKET: Socket error MODBUS_ERR_TIMEOUT: Connect timeout MODBUS_ERR_CONNECT: Connect to target error MODBUS_ERR_HANDLE: No more handles	

MX_Modbus_Tcp_Master_Close

INT32 MX_Modbus_Tcp_Master_Close(UINT32 sHandle);	
This function closes the TCP connection established by MX_Modbus_Tcp_Master_Open.	
Input Parameters	
sHandle	TCP connection handle
Output Parameters	
None	
Return Value	
MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED	

MX_Modbus_Tcp_Master_Ioctl

INT32 MX_Modbus_Tcp_Master_Ioctl(UINT32 sHandle, UINT8 unitId, UINT32 rwTimeoutMs);	
This function configures the unit identifier and timeout values of the Modbus/TCP connection.	
Input Parameters	
sHandle	TCP connection handle
unitId	Unit identifier
rwTimeoutMs	timeout value in milliseconds
Output Parameters	
None	
Return Value	
MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED	

MX_Modbus_Tcp_Master_Read_Coils

INT32 MX_Modbus_Tcp_Master_Read_Coils(UINT32 sHandle, UINT16 startAddr , UINT16 coilCount, UINT8 byteCoils[]);	
This function performs Modbus function code 0x1; it reads multiple coils and puts the output in parameter byeCoils.	
Input Parameters	
sHandle	TCP connection handle
startAddr	Modbus address
coilCount	The numbers of read coils
Output Parameters	
byteCoils	This parameter holds the read coils value; users should provide enough buffer space. For example, if coildCount=9, the buffer size of byteCoils should be at least 2 bytes.
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED MODBUS_ERR_RESP MODBUS_ERR_TIMEOUT MODBUS_ERR_SOCKET	

MX_Modbus_Tcp_Master_Write_Coils

INT32 MX_Modbus_Tcp_Master_Write_Coils(UINT32 sHandle, UINT16 startAddr , UINT16 coilCount, UINT8 byteCoils[]);	
This function performs Modbus function code 0xF; it writes multiple coils, and the coils value should be put in parameter byteCoils in bit-map format	
Input Parameters	
sHandle	TCP connection handle
startAddr	MODBUS address
coilCount	The numbers of coils.
byteCoils	User should put coils value in bit-map format.
Output Parameters	
None	
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED MODBUS_ERR_SOCKET MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP	

MX_Modbus_Tcp_Master_Write_Coil

INT32 MX_Modbus_Tcp_Master_Write_Coil(UINT32 sHandle, UINT16 addr , UINT8 coil);	
This function performs Modbus function code 0x5; it writes a single coil.	
Input Parameters	
sHandle	TCP connection handle
addr	Modbus address
coil	coil value: 0 or 1
Output Parameters	
None	
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED MODBUS_ERR_SOCKET MODBUS_ERR_TIMEOUTMODBUS_ERR_RESP	

MX_Modbus_Tcp_Master_Read_Discrete_Inputs

INT32 MX_Modbus_Tcp_Master_Read_Discrete_Inputs(UINT32 sHandle, UINT16 startAddr , UINT16 coilCount, UINT8 byteCoils[]);	
This function performs Modbus function code 0x2; it reads multiple discrete inputs.	
Input Parameters	
sHandle	TCP connection handle
startAddr	Modbus address
coilCount	The numbers of discrete inputs.
Output Parameters	
byteCoils	This parameter holds the read discrete inputs; users should provide enough buffer space. For example, if coildCount=9, the buffer size of byteCoils should be at least 2 bytes.
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED MODBUS_ERR_SOCKET MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP	

MX_Modbus_Tcp_Master_Read_Input_Regs

INT32 MX_Modbus_Tcp_Master_Read_Input_Regs(UINT32 sHandle, UINT16 startAddr , UINT16 regCount, UINT16 regs[]);	
This function performs Modbus function code 0x4; it reads multiple input registers.	
Input Parameters	
sHandle	TCP connection handle
startAddr	MODBUS address
regCount	The numbers of input registers
timeoutMs	Timeout value in milliseconds
Output Parameters	
regs	This parameter holds the read input registers; users should provide enough buffer space. For example, if regCount =4, the buffer size of regs should be at least 8 bytes.
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED MODBUS_ERR_SOCKET MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP	

MX_Modbus_Tcp_Master_Read_Holding_Regs

INT32 MX_Modbus_Tcp_Master_Read_Holding_Regs(UINT32 sHandle, UINT16 startAddr , UINT16 regCount, UINT16 regs[]);	
This function performs Modbus function code 0x3; it reads multiple holding registers.	
Input Parameters	
sHandle	TCP connection handle
startAddr	Modbus address
regCount	The numbers of holding registers
Output Parameters	
regs	This parameter holds the read holding registers; users should provide enough buffer space. For example, if regCount=4, the buffer size of regs should be at least 8 bytes.
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED MODBUS_ERR_SOCKET MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP	

MX_Modbus_Tcp_Master_Write_Holding_Regs

INT32 MX_Modbus_Tcp_Master_Write_Holding_Regs(UINT32 sHandle, UINT16 startAddr , UINT16 regCount, UINT16 regs[]);	
This function performs Modbus function code 0x10; it writes multiple holding registers.	
Input Parameters	
sHande	TCP connection handle
startAddr	Modbus address
regCount	The numbers of holding registers
Regs	This parameter holds the holding registers value; each register occupies 2 bytes of buffer space.
Output Parameters	
None	
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED MODBUS_ERR_SOCKET MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP	

MX_Modbus_Tcp_Master_Write_Holding_Reg

INT32 MX_Modbus_Tcp_Master_Write_Holding_Reg(UINT32 sHandle, UINT16 addr , UINT16 regValue);	
This function performs Modbus function code 0x06; it writes a single holding register.	
Input Parameters	
sHandle	TCP connection handle
Addr	Modbus address of the holding register
regValue	This parameter holds the value of the holding register
Output Parameters	
None	
Return Value	
Modbus standard error codes are mapped to values 16 to 24. Refer to the header file libmoxa_pgm.h. MODBUS_ERR_OK MODBUS_ERR_LIB_INIT MODBUS_ERR_PARAM MODBUS_ERR_NOT_OPENED MODBUS_ERR_SOCKET MODBUS_ERR_TIMEOUT MODBUS_ERR_RESP	

Modbus/TCP Slave

Users must include **libmoxa_pgm.h**. Return values of Modbus/TCP Slave API functions are shown below; they can also be found at **libmoxa_pgm.h**.

```
#define MODBUS_ERR_OK                    0
#define MODBUS_ERR_ADDRESS               -1
#define MODBUS_ERR_ADDRESS_COLLISION     -2
#define MODBUS_ERR_BIND                  -3
#define MODBUS_ERR_CREATE_SOCKET         -4
#define MODBUS_ERR_CREATE_THREAD         -5
#define MODBUS_ERR_EXCEPTION             -6
#define MODBUS_ERR_EXCEED_MAP_SIZE       -7
#define MODBUS_ERR_FUNCTION              -8
#define MODBUS_ERR_ILLEGAL_HANDLE        -9
#define MODBUS_ERR_IDLE_TIMEOUT          -10
#define MODBUS_ERR_ILLEGAL_ACTION        -11
#define MODBUS_ERR_LISTEN                -12
#define MODBUS_ERR_LISTEN_PORT_OVERFLOW  -13
#define MODBUS_ERR_MAP_TYPE              -14
#define MODBUS_ERR_MAP_EMPTY             -15
#define MODBUS_ERR_MEMORY_LEAK           -16
#define MODBUS_ERR_NO_START              -17
#define MODBUS_ERR_NO_REGISTER           -18
#define MODBUS_ERR_PORT_LISTENING        -19
#define MODBUS_ERR_REGISTERED_PORT       -20
#define MODBUS_ERR_SIZE                  -21
#define MODBUS_ERR_SYSTEM_TIMEOUT        -22
#define MODBUS_ERR_SET_SOCKET_MODE       -23
#define MODBUS_ERR_SET_SOCKET_OPTION     -24
```

MX_Modbus_Tcp_Slave_Init

int MX_Modbus_Tcp_Slave_Init();	
Inits the Modbus Slave Framework.	
Return Value	
Result of the call	

MX_Modbus_Tcp_Slave_Exit

int MX_Modbus_Tcp_Slave_Exit();	
Exits Modbus Slave Framework.	
Return Value	
Result of the call	

MX_Modbus_Tcp_Slave_Register

int MX_Modbus_Tcp_Slave_Register(UINT16 port, UINT16 map_size, UINT32 idle_timeout_second, UINT32 *sHandle);	
Registers a Modbus listen port.	
Input Parameters	
port	
map_size	the maximum is 500 and the minimum is 1
idle_timeout_second	the minimum is 10 seconds
Output Parameters	
sHandle	
Return Value	
Result of the call	

MX_Modbus_Tcp_Slave_Unregister

int MX_Modbus_Tcp_Slave_Unregister(UINT32 sHandle);	
Unregisters the Modbus listen port.	
Input Parameters	
sHandle	
Return Value	
Result of the call	

MX_Modbus_Tcp_Slave_Start

int MX_Modbus_Tcp_Slave_Start(UINT32 sHandle);	
Starts the Modbus server; the listen port is the registered port.	
Input Parameters	
sHandle	
Return Value	
Result of the call	

MX_Modbus_Tcp_Slave_Stop

int MX_Modbus_Tcp_Slave_Stop(UINT32 sHandle);	
Stops the Modbus server.	
Input Parameters	
sHandle	
Return Value	
Result of the call	

MX_Modbus_Tcp_Slave_Add_Entry

int MX_Modbus_Tcp_Slave_Add_Entry(UINT32 sHandle, UINT8 Map_Type, UINT16 Address, void *pUserData, pfnModbusRead, pfnModbusWrite);	
To add a Modbus entry.	
Input Parameters	
sHandle	
Map_Type	
Address	
pUserData	pass the user data to the function callback
pfnModbusRead	function callback for Modbus read
pfnModbusWrite	function callback for modbus write
Return Value	
Result of the call	

MX_Modbus_Tcp_Slave_Delete_Entry

int MX_Modbus_Tcp_Slave_Delete_Entry(UINT32 sHandle, UINT8 Map_Type, UINT16 Address);	
Deletes a Modbus entry.	
Input Parameters	
sHandle	
Map_Type	
Address	
Return Value	
Result of the call	

MX_Modbus_Tcp_Slave_Map_Count

int MX_Modbus_Tcp_Slave_Map_Count(UINT32 sHandle, UINT16 *count);	
Gets the number of entries for the registered port.	
Input Parameters	
sHandle	
Output Parameters	
count	
Return Value	
Result of the call	

MX_Modbus_Tcp_Slave_Map_Dump

int MX_Modbus_Tcp_Slave_Map_Dump(UINT32 sHandle);	
Dumps total entries for the registered port.	
Input Parameters	
sHandle	
Return Value	
Result of the call	

RAS API

The following RAS API functions are used to connect the target system's I/O status to the Moxa Active OPC Server so that a remote SCADA system can collect and control the real-time I/O status from the Moxa RTU Controllers via Active OPC Server.

MX_Ras_Init

INT32 MX_Ras_Init(void);	
This function, which must be called before invoking any other RAS API function, will allocate resources needed for the RAS library. To release the resources, use the function MX_Ras_Uninit.	
Input Parameters	
None	
Output Parameters	
None	
Return Value	
RAS_ERR_OK RAS_ERR_LIB_INIT	

MX_Ras_Uninit

void MX_Ras_Uninit(void);	
Use this function to release the RAS library resources allocated by MX_Ras_Init.	
Input Parameters	
None	
Output Parameters	
None	
Return Value	
None	

MX_Ras_Connect

INT32 MX_Ras_Connect(UINT8 *serverName, UINT32 heartBeatS, UINT8 *ipAddress, UINT16 port, UINT32 timeoutMs, UINT32 *sHandle);	
This function establishes a TCP connection to the Active OPC (RAS) server and retrieves the connection handle from the parameter sHandle. The handle is used by RAS API functions.	
Input Parameters	
serverName	A C string with NULL terminated. Active OPC Server combines this string and the MAC address as a primary key to identify the TCP connection. Connections with the same primary key are classified as the same device by the Active OPC Server. The maximum length is 256 bytes, including one NULL-terminated byte.
hearBeatS	Heart-beat interval in seconds (0: disable heartbeat)
ipAddress	Active OPC Server IP address
port	Active OPC server listened port (9900 is recommended)
timeoutMs	Timeout interval in milliseconds
Output Parameters	
sHandle	The returned TCP connection handle.
Return Value	
RAS_ERR_OK: If this code is returned, sHandle should be released by MX_Ras_Close(). RAS_ERR_LIB_INIT RAS_ERR_PARAM RAS_ERR_CREATE_THREAD RAS_ERR_SOCKET RAS_ERR_TIMEOUT RAS_ERR_CONNECT RAS_ERR_NO_MEMORY RAS_ERR_NO_QUEUE	

MX_Ras_Disconnect

INT32 MX_Ras_Disconnect(UINT32 sHandle);	
This function closes the connection and clears all registered tags. Use MX_Ras_Reconnect instead to reconnect to the Active OPC Server and keep all registered tags.	
Input Parameters	
sHandle	The opened connection handle.
Output Parameters	
None	
Return Value	
RAS_ERR_OK RAS_ERR_LIB_INIT RAS_ERR_PARAM	

MX_Ras_Reconnect

INT32 MX_Ras_Reconnect(UINT32 sHandle, UINT32 timeoutMs);	
This function closes the opened connection and establishes a new connection to the Active OPC Server. Every registered tag will be reserved. User can use this function while RAS APIs return an error code RAS_ERR_SOCKET.	
Input Parameters	
sHandle	The opened connection handle.
timeoutMs	The timeout value in millisecond.
Output Parameters	
None	
Return Value	
RAS_ERR_OK RAS_ERR_LIB_INIT RAS_ERR_PARAM RAS_ERR_NO_MEMORY RAS_ERR_NO_QUEUE	

MX_Ras_AddTag

INT32 MX_Ras_AddTag(UINT32 sHandle, TAG_INFO *tagInfo, UINT32 timeoutMs);	
This function registers a new user-defined tag to the Active OPC (RAS) Server.	
Input Parameters	
sHandle	The opened connection handle.
tagInfo	The C-structure TAG_INFO contains the configuration information of a tag. TAG_INFO.tagValue: The initial value of the tag; the user should provide enough memory for the specific type of value. TAG_INFO.strTagName: A null-terminated C string; it must be unique. TAG_INFO.strTagDescription: A null-terminated C string. TAG_INFO.strTagUnit: A null-terminated C string. TAG_INFO.tagAccessRight: 0: read-only; 2: read/write TAG_INFO.tagValueType: TAG_TYPE_BOOL: 1 byte, 0/1 TAG_TYPE_WORD: 2 bytes TAG_TYPE_INT: 4 byte signed integer TAG_TYPE_DWORD: 4 byte unsigned integer TAG_TYPE_FLOAT: 4 bytes, floating point TAG_INFO.tagQuality: 0x0000-0x7FFF: Good; 0x8000-0xFFFF: Bad TAG_INFO.tagCallBack: User-defined tag-write callback function. This function will be invoked when the Active OPC Server issues a write command to the registered tags. The callback function should return RAS_ERR_OK to acknowledge success.
timeoutMs	The timeout value in milliseconds.
Output Parameters	
None	
Return Value	
RAS_ERR_OK RAS_ERR_LIB_INIT RAS_ERR_PARAM RAS_ERR_EXIST_TAG RAS_ERR_NO_MEMORY RAS_ERR_NO_QUEUE RAS_ERR_SERVER_ERROR	

MX_Ras_DelTag

INT32 MX_Ras_DelTag(UINT32 sHandle, UINT8 *tagName, UINT32 timeoutMs);	
This function deletes a registered tag from the Active OPC (RAS) Server.	
Input Parameters	
sHandle	The opened connection handle
tagName	The name of the tag (null-terminated C string)
timeoutMs	The timeout value in milliseconds
Output Parameters	
None	
Return Value	
RAS_ERR_OK	
RAS_ERR_LIB_INIT	
RAS_ERR_PARAM	
RAS_ERR_NO_MEMORY	
RAS_ERR_NO_QUEUE	
RAS_ERR_SERVER_ERROR	

MX_Ras_DelAllTag

INT32 MX_Ras_DelAllTag(UINT32 sHandle, UINT32 timeoutMs);	
This function deletes all registered tags from the Active OPC (RAS) Server.	
Input Parameters	
sHandle	The opened connection handle
timeoutMs	The timeout value in milliseconds
Output Parameters	
None	
Return Value	
RAS_ERR_OK	
RAS_ERR_LIB_INIT	
RAS_ERR_PARAM	
RAS_ERR_NO_MEMORY	
RAS_ERR_NO_QUEUE	
RAS_ERR_SERVER_ERROR	

MX_Ras_UpdateTag

INT32 MX_Ras_UpdateTag(UINT32 sHandle, TAG_INFO *tagInfo, UINT32 timeoutMs);	
This function updates the configuration info of a tag to the Active OPC (RAS) server.	
Input Parameters	
sHandle	The opened connection handle
tagInfo	Refer to the parameter description of function MX_Ras_AddTag()
timeoutMs	The timeout value in milliseconds
Output Parameters	
None	
Return Value	
RAS_ERR_OK	
RAS_ERR_LIB_INIT	
RAS_ERR_PARAM	
RAS_ERR_NO_TAG	
RAS_ERR_NO_MEMORY	
RAS_ERR_NO_QUEUE	
RAS_ERR_SERVER_ERROR	

MX_Ras_UpdateValue

INT32 MX_Ras_UpdateValue(UINT32 sHandle, UINT8 *tagName, void *tagValue, UINT32 timeoutMs);	
This function updates the tag value to the Active OPC (RAS) Server.	
Input Parameters	
sHandle	The opened connection handle.
tagName	Refer to the parameter description of function MX_Ras_AddTag()
tagValue	Refer to the parameter description of function MX_Ras_AddTag()
timeoutMs	The timeout value in milliseconds
Output Parameters	
None	
Return Value	
RAS_ERR_OK	
RAS_ERR_LIB_INIT	
RAS_ERR_PARAM	
RAS_ERR_NO_TAG	
RAS_ERR_NO_MEMORY	
RAS_ERR_NO_QUEUE	
RAS_ERR_SERVER_ERROR	

MX_Ras_UpdateHeartbeat

INT32 MX_Ras_UpdateHeartbeat(UINT32 sHandle, UINT32 heartbeatS, UINT32 timeoutMs);	
This function updates the heartbeat interval to the Active OPC (RAS) server.	
Input Parameters	
sHandle	The opened connection handle
heartbeatS	Heartbeat interval in seconds (0: disabled)
timeoutMs	The timeout value in milliseconds
Output Parameters	
None	
Return Value	
RAS_ERR_OK RAS_ERR_LIB_INIT RAS_ERR_PARAM RAS_ERR_NO_TAG RAS_ERR_NO_MEMORY RAS_ERR_NO_QUEUE RAS_ERR_SERVER_ERROR	

Misc API

Device nodes are located at **/dev/mxmisc**, **/dev/rtc**, and **/dev/swtd**. Users must include **libmoxa_pgm.h**. Returned values of MISC API are shown below; they can also be found at **libmoxa_pgm.h**. Note that **/dev/mxmisc** and **/dev/rtc** cannot both be opened again if they are in use.

```
#define MISC_SUCCESS          0
#define MISC_ERR_DEVICE       -1
#define MISC_ERR_ARGUMENT     -2
#define MISC_ERR_RW           -3
#define MISC_ERR_ACTION       -4
```

MX_Signal_LED_Get

UINT32 MX_Signal_LED_Get(UINT32 num, UINT32 *state);	
Gets the signal LED status. The state will be written to the *state* parameter.	
Input Parameters	
num	0: Signal 0 1: Signal 1 2: Signal 2
Output Parameters	
state	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_Ready_LED_Get

UINT32 MX_Ready_LED_Get(UINT32 *state);	
Gets the ready LED status. The state will be written to the *state* parameter.	
Output Parameters	
state	0: Off 1: Green 2: Red (for the ioPAC 8020-C series)
Return Value	
Result of the call	

MX_Fault_LED_Get

UINT32 MX_Fault_LED_Get(UINT32 *state);	
Gets the fault LED status. The state will be written to the *state* parameter.	
Output Parameters	
state	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_Link_LED_Get

UINT32 MX_Link_LED_Get(UINT32 *state);	
Gets the link LED status. The state will be written to the *state* parameter.	
Output Parameters	
state	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_IO_LED_Get

UINT32 MX_IO_LED_Get(UINT32 *state);	
Gets the I/O LED status. The state will be written to the *state* argument.	
Output Parameters	
state	0: Off 1: Green 2: Red
Return Value	
Result of the call	
Notes	
ioPAC 8020-C series only	

MX_Signal_LED_Set

UINT32 MX_Signal_LED_Set(UINT32 num, UINT32 state);	
Sets the signal LED status according to num and state.	
Input Parameters	
num	0: Signal 0 1: Signal 1 2: Signal 2
state	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_Ready_LED_Set

UINT32 MX_Ready_LED_Set(UINT32 state);	
Sets the ready LED status according to the state.	
Input Parameters	
state	0: Off 1: Green 2: Red (for the ioPAC 8020-C series)
Return Value	
Result of the call	

MX_Fault_LED_Set

UINT32 MX_Fault_LED_Set(UINT32 state);	
Sets the fault LED status according to the *state* parameter.	
Input Parameters	
state	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_Link_LED_Set

UINT32 MX_Link_LED_Set(UINT32 state);	
Sets the link LED status according to the *state* parameter.	
Input Parameters	
state	
Return Value	
Result of the call	
Notes	
ioLogik W5348-C series only	

MX_IO_LED_Set

UINT32 MX_IO_LED_Set(UINT32 state);	
Sets the IO LED status according to the *state* parameter.	
Input Parameters	
state	0: Off 1: Green 2: Red
Return Value	
Result of the call	
Notes	
ioPAC 8020-C series only	

MX_RTC_Get

UINT32 MX_RTC_Get(struct rtc_time *rtc);	
To get the RTC. The RTC will be written on the argument rtc.	
Output Parameters	
rtc	
Return Value	
Result of the call	

MX_RTC_Set

UINT32 MX_RTC_Set(struct rtc_time *rtc);	
Sets the Real Time Clock (RTC).	
Input Parameters	
rtc	
Return Value	
Result of the call	

MX_SWTD_Enable

UINT32 MX_SWTD_Enable(UINT32 swtdtime);	
Enables the Software Watchdog. When the Software Watchdog is enabled, it will not trigger the Watchdog.	
Input Parameters	
swtdtime	the unit is milliseconds
Return Value	
Result of the call	

MX_SWTD_Disable

UINT32 MX_SWTD_Disable();
Disables the Software Watchdog. When the Software Watchdog is disabled, the kernel will handle the acknowledgement of the Watchdog function.
Return Value
Result of the call

MX_SWTD_Ack

UINT32 MX_SWTD_Ack();	
Kicks the Watchdog.	
Return Value	
Result of the call	

MX_IO_Scan_Rate_Get

int MX_IO_Scan_Rate_Get(UINT32 *units);	
Gets the I/O Scan Rate.	
Output Parameters	
units	1 unit = 10 ms for the ioLogik W5348-C series 1 unit = 10 ms for the ioPAC 8020-C series
Return Value	
Result of the call	

MX_IO_Scan_Rate_Set

int MX_IO_Scan_Rate_Set(UINT32 units);	
Sets the I/O Scan Rate.	
Input Parameters	
units	1 unit = 10 ms for the ioLogik W5348-C series 1 unit = 10 ms for the ioPAC 8020-C series
Return Value	
Result of the call	

MX_API_Version_Get

UINT32 MX_API_Version_Get(void);
Gets the API version.
Return Value
API version

MX_API_BuildDate_Get

UINT32 MX_API_BuildDate_Get(void);
Gets the API Build Date.
Return Value
API Build Date

MX_System_Version_Get

int MX_System_Version_Get(void);
Gets the System version.
Return Value
System version

MX_System_BuildDate_Get

int MX_System_BuildDate_Get(void);
Gets the System Build Date.
Return Value
System Build Date

3

Sample Code

This chapter includes important information for programmers. API sample code is placed in the **mxtest** folder on the Document and Software CD.

The following topics are covered in this chapter:

- **I/O**
- **I/O Alarm**
- **Cellular (ioLogik W5348-C series only)**
- **SMS (ioLogik W5348-C series only)**
- **Serial**
- **Modbus Master (TCP/RTU)**
- **Modbus TCP Slave**
- **RAS (Active Tag Service)**
- **Misc**
- **SD**

I/O

C file: CD://Software/example/mxtest/io_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: In the C main function, this example demonstrates how to use the I/O API to get/set the value of the digital I/O and analog I/O. It also shows how to configure an I/O channel.

I/O Alarm

C file: CD://Software/example/mxtest/io_alarm_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: This example shows how to use the I/O alarm API to configure the trigger criteria for alarm-based I/O. The target system will detect the alarms automatically.

Cellular (ioLogik W5348-C series only)

C file: CD://Software/example/mxtest/cellular_net_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: In the C main function, this example shows how to use the cellular API to control a cellular modem's connection to a 3G network.

SMS (ioLogik W5348-C series only)

C file: CD://Software/example/mxtest/sms_send_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: This example shows how to use the SMS API to send a short message. The message content can be ASCII or UCS2 format. For ASCII messages, the maximum length is 160 bytes. For UCS2 messages, the maximum length is 140 bytes (70 UCS2 characters).

Serial

C file: CD://Software/example/mxtest/serial_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: This example shows how to configure and communicate to a target system's serial ports. For the ioLogik W5348-C series, the external serial port numbers are 0 and 1. For the ioPAC-8020-C series, the external serial port number is 0.

Modbus Master (TCP/RTU)

C files: CD://Software/example/mxtest/modbus_master_tcp_test.c
CD://Software/example/mxtest/modbus_master_rtu_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: The Modbus Master API includes support for both TCP and RTU protocols. This example shows how to communicate to a Modbus slave device.

Modbus TCP Slave

C file: CD://Software/example/mxtest/modbus_slave_tcp_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: The Modbus TCP slave API includes support for the TCP protocol. This example shows how to set up a Modbus/TCP server.

RAS (Active Tag Service)

C file: CD://Software/example/mxtest/ras_client_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: The RAS API provides a way to link local I/O status to MOXA Active OPC Server. I/O data can be actively pushed to the server and to the SCADA system or updated from the server and SCADA system. The sample code shows how to establish the connection and Add/Del/Upload a tag to the Active OPC Server.

Misc

C file: CD://Software/example/mxtest/misc_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: This example shows how to operate GPIO, RTC, and Watchdog.

SD

C file: CD://Software/example/mxtest/sd_test.c

Header file: libmoxa_pgm.h

API Library file: libmoxa_pgm.a

Description: This example shows how to operate an SD card.