Contents

Version Changes

Date/Rev	Description
18-Apr-25/Rev 1	Initial release

Modbus Master Controller

The device allows multiple Modbus slave units connected into the communicate bus to be managed by backend server though internet using IoT concept.

Users require to configure all register properties that require to be polled by the Modbus Master Controller (Either using auto polling, menu polling or menu setting slave's register). Each poll command provided must be included with a unique Reference ID ("ref", any 16bit integer determine by user) for the command. This Reference ID will be used throughout all the communication between the backend server and the Modbus Master Controller.

Once the polling register properties are properly configured into the device memory, it will start auto polling the Modbus Slave Device according to the polling period provided in the configuration. But if the polling period is set to 0. User require to manually request the polling (using command r.mbmctrl.poll). To set the registers in the Modbus Slave's register, users can use command r.memctrl.set.

Users can change the Modbus Communication Baud-rate by using command s.mbmctrl.setcom.

The default configuration is 19200baud, 8 bits, No Parity and 1 Stop Bit (8-n-1)

Peripheral

The peripheral names are as below,

On JSON Key under d.sys.perip array,

Key	Value Type	Value
Keycmd	String	mbmctrl
feature	String	NA

Attribute

Share Attribute

Key	Value Type	Description
s.mbmctrl.setcom	object	Change the Modbus Serial port setting.

Json object is as below,

```
{
  “baudrate”: 19200,
  “parity”:0,
  “stopbit”: 0
}
```

The value pair detail is as below,

- baudrate: Communication port baud rate, support 1200baud to 115200baud.
- parity: Parity bit,
  - 0 = No Parity
  - 1 = Even Parity
  - 2 = Odd Parity
- stopbit: No. of stop bit.
  - 0 = 1 Stop bit
  - 1 = 1.5 Stop bit
  - 2 = 2 Stop bit

Telematics Data

Key	Value Type	Description
t.mbmctrl.poll	Array of object	**Return the result of last polling.**

Each object in the Json array consists of value pair as listed below,

```
{
  "ref": 100,
  "iserror":0,
  "isbit": 1,
  "param": [ 1,0, …]
}
```

The value pair detail is as below,

- ref: Modbus Polling command reference number.
- iserror: Value 0(false) / 1(true). It is set to 1 if encounter error during last polling.
- isbit: Value 0(false) / 1(true). If set to true, the "param" is a bit value result (register type Coil or Discrete). This value pair only available if "iserror" = 0.
- param: The polling result in array format. The array is according to the polling command "regsize" provided. This value pair only available if "iserror" = 0.
    - When isBit = 0 (Input/Holding register), each element in param is 16bit word
    - When isBit = 1 (Coil/Discrete register), each element in param is Boolean bit with value 0 or 1.

Key	Value Type	Description
t.mbmctrl.set	Array of object	**Return the set request result of r.mbmctrl.set.**

Each object in the Json array consists of value pair as listed below,

```
{
  "ref": 0,
  "succ":0
}
```

The value pair detail is as below,

- ref: Modbus Polling command reference number.
- succ: Value 0(false) / 1(true). It is set to 1 if set request is successful.

RPC Call and Response

Key	Value Type	Description
r.mbmctrl.addcmd	Array of object	**Add multiple Modbus polling commands.**

Each object in the Json array consists of value pair as listed below,

```
{
  “ref”: 0,
  “period”:10,
  “type”: 0,
  “slvaddr”: 1,
  “regaddr”: 10,
  “regsize”: 4
}
```

- ref: Unsigned integer. Modbus Polling command reference number. Should be unique. It will be use as the command reference ID through out the communication with the backend server.
- period: Unsigned integer. Command polling period. Each count is 0.1 second. (E.g. Period = 10 will cause the command polling in every 1 second). If the period is set to 0, it will not do the auto polling the command.
- type. Unsigned integer. Register type to poll,
  - 0 = Holding Register (16bit Word Parameter, Read/Write)
  - 1 = Input Register (16bit Word Parameter, Read Only)
  - 2 = Coil bit (1bit Boolean Parameter, Read Only)
  - 3 = Discrete bit (1bit Boolean Parameter, Read/Write)

  Other value is not valid.
- regaddr. Unsigned integer. Modbus register Address (zero based address).
- regsize. Unsigned integer. Total number of registers to poll. The maximum number of “regsize” is 64.

Please take note that, under Modbus protocol, Holding/Input Register is a 16bit word per register address, and Coil/Discrete Register is 1bit result per register address.

If the provided “ref” has already exists in the memory setting, it will replace it will the newly provided setting.

If the “ref” does not exist, it will create and store the setting into the memory.

The reply format of adding command either success or failure for each object element will be in Json array of object and the value pair in each array object are as listed below,

```
{
  “ref”: 0,
  “succ”: 1
}
```

If adding is success, it will return succ:1.

Key	Value Type	Description
r.mbmctrl.delcmd	Array of object	**Delete multiple Modbus Command from the setting memory.**

Each element in the Json array object will consists of value pair as listed below,

```
{
   “ref”: 0
}
```

The reply format of delete command either success or failure for each object element will be in Json array of object and the value pair in each array object are as listed below,

```
{
   “ref”: 0,
   “succ”: 1
}
```

If delete is success, it will return succ:1

Key	Value Type	Description
r.mbmctrl.delall	Bool	**Set to true will DELETE ALL the setting in the memory.**

E.g.

r.mbmctrl.delall: true

If the delete process is successful, it will return succ:1 in RPC response topic

```
{
   “succ”: 1
}
```

Key	Value Type	Description
r.mbmctrl.listcmd	Bool	**Set to true. It will return all the currently configured Modbus polling command.**

The reply is in array format. Each object in the Json array consists of value pair as listed below,

```
{
  “ref”: 0,
  “period”:10,
  “type”: 0,
  “slvaddr”: 1,
  “regaddr”: 10,
  “regsize”: 4
}
```

Regarding the detailed information for the value pair, please refer to r.mbmctrl.addcmd command.

Key	Value Type	Description
r.mbmctrl.set	Array of object	**Set multiple parameters according to the Modbus cmd set in the device. The set status result will be return in telematic topic under command “t.mbmctrl.set”**

Each object in the Json array consists of value pair as listed below,

```
{
  “ref”: 0,
  “param”: {1,…}
}
```

Each element in “param” array is the value for the 1 register. The total number of element in the “param” must be same as “regsize” setting during “r.mbmctrl.addcmd” setup.

- If the register type is Input/Holding Register, each element is 16bit word.
- If the register type is Coil/Discrete Register, each element is Boolean with value 0 or 1.

If the accepted to process, it will return succ:1 in RPC response topic

```
{
  “succ”: 1
}
```

Key	Value Type	Description
r.mbmctrl.poll	Object	**Request to poll the list of Ref. provided. The Ref. must be already registered using command r.mbmctrl.addcmd.**

.
E.g.

{ “r.mbmctrl.poll”: [{ “ref”:20 } , { “ref”:30 }] }

It will send out all the latest parameter values cache in the memory in telematic topic (using command “t.mbmctrl.poll”) if the request is accepted. Please refer to command “t.mbmctrl.poll” for the value return format.

If the accepted to process, it will return succ:1 in RPC response topic

```
{
   [ {  “ref”:20, “succ”: 1 }],  [ {  “ref”:30, “succ”: 1 }],
}
```

Key	Value Type	Description
r.mbmctrl.pollall	Bool	**Set to true. It will return all the parameter value cache in the memory in Telematics Topic**

.
E.g.

{ “r.mbmctrl.pollall”: true }

It will send out all the latest parameter values cache in the memory in telematic topic (using command “t.mbmctrl.poll”) if the request is accepted. Please refer to command “t.mbmctrl.poll” for the value return format.

If the accepted to process, it will return succ:1 in RPC response topic

```
{
  “succ”: 1
}
```

Command Example

RPC Req.: Add Command

- Ref 10 with Holding(R/W), addr 4, size 4 and
- Ref 20 with Coil (R/W), addr 4, size 4

```
{
 "r.mbmctrl.addcmd": [
  {
   "ref": 10,
   "period": 10,
   "type": 0,
   "slvaddr": 1,
   "regaddr": 4,
   "regsize": 4
  },
  {
   "ref": 20,
   "period": 10,
   "type": 2,
   "slvaddr": 1,
   "regaddr": 4,
   "regsize": 4
  }
 ]
}
```

RPC Req.: Delete All Command

```
{
 "r.mbmctrl.delall":true
}
```

RPC Req.: List All parameter

```
{
 "r.mbmctrl. pollall":true
}
```

RPC Req.: Set param

```
{
 "r.mbmctrl.set": [
  { "ref": 20, "param": [1, 0, 0, 1] }
 ]
}
```

RPC Req.: Set param

```
{
 "r.mbmctrl.set": [
  { "ref": 20, "param": [  1,   1, 1, 1] },
  { "ref": 10, "param": [100, 200, 1, 2] }
 ]
}
```

4-Wire I/O Extension Bus

The I/O Extension Bus allow the device to add multiple I/O Extension with each I/O Extension unit having 8 Input port and 8 Output port.

Each unit of I/O Extension is identified by a device number 0 to 7. After the I/O Extension unit is connected to the bus, the controller requires restart to allow the controller to scan for any connected I/O Extension unit.

Peripheral

The peripheral names are as below,

On JSON Key under d.sys.perip array,

Key	**Value Type**	**Value**
Keycmd	String	ioextension
feature	String	16ch

Attribute

Device Only Attribute

Key	**Value Type**	**Description**
d.ioextension.dev	Array of object	List the currently detected device Is either connected or disconnected.

Each array element will contain value as below,

```
{
  “id”: 0,
  “conn”: true
}
```

Where,

- id = the Device ID.
- conn = is “true” if the device connected and “false” if the device is disconnected.

It only will include the device that was detected during initial power up bus scanning.

Share Attribute

Key	Value Type	Description
s. ioextension.setup	Array of object	Define the function mode of each Device ID and I/O Channel

Each array element will contain value as below,

```
{
  "id": 0,
  "config": [
    {
      "ch": 0,
      "mode": 1,
      "fperiodon": 3,
      "fperiodoff": 1,
      "pulsecnt: 1,
    }
  ]
}
```

- id, Device ID in the bus. (0 to 7)
- ch, I/O Channel index (0 to 15)
- mode, I/O channel mode. For more details, please refer them to the I/O Port Mode Table below.
- fperiodon, on pulse width period in 0.1 sec/cnt during Pulse Output Mode Only (Mode 1,3).
- fperiodoff, off pulse width period in 0.1 sec/cnt during Pulse Output Mode Only (Mode 1,3).
- pulsecnt, number of on/off cycle during Pulse Output Mode Only (Mode 1,3).

The total period of each cycle will be,

Cycle Period = "periodon" + "periodoff"

I/O Port Mode Table

Mode	Bit 2	Bit 1	Bit0	Description
0	0	0	0	Output Port, Non-Inverse, Standard Mode
1	0	0	1	Output Port, Non-Inverse, Pulse Output Mode
2	0	1	0	Output Port, Inverse, Standard Mode
3	0	1	1	Output Port, Inverse, Pulse Output Mode
4	1	0	0	Input Port, Non-Inverse, Standard Mode
5	1	0	1	Input Port, Non-Inverse, Push Button Input Mode
6	1	1	0	Input Port, Inverse, Standard Mode
7	1	1	1	Input Port, Inverse, Button Input Mode

Input Port, Standard Mode

During this mode, any input state changes (On -> Off or Off -> On) message will be sent back to the server immediately through Telematics Channel.

During Off -> On, the counter will increase by one.

Input Port, Push Button Mode (Feature Mode)

During this mode, the Input Channel signal will go through an internal de-bounce management unit to prevent multiple triggers during button press/On. The input will be able to detect single press, double press, triple press and long press (press more than 4 second) condition.

Once the user releases the button during the final press action, the system will determine the type of action that has been performed by the user and send back the result to the server.

On each user action, the counter will increase by 1.

Output Port Standard Mode

During this mode, when the server/host sends ON or OFF request, the device will immediately send out the signal to the output port.

The On/Off condition of the output port will be notified in the Telematics Channel.

Output Port Pulsed Output (Feature Mode)

During this mode, when the server/host sends ON request, the device will immediately send out the pulses on the output port. The ON period and OFF period can be configured through fperiodon and fperiodoff register respectively. The number of pulses cycle to be sent out can be configure through register pulsecnt.

Once the number of pulse cycles is completed, the output will stop, and the off message will be sent back to the server.

During output of the pulses, if the server/host sends another ON request, the pulse counter will reset without interrupting the pulse cycle timing.

If the server/host sends OFF request while output port is busy sending out the pulse, the output will immediately turn off and stop the output pulse.

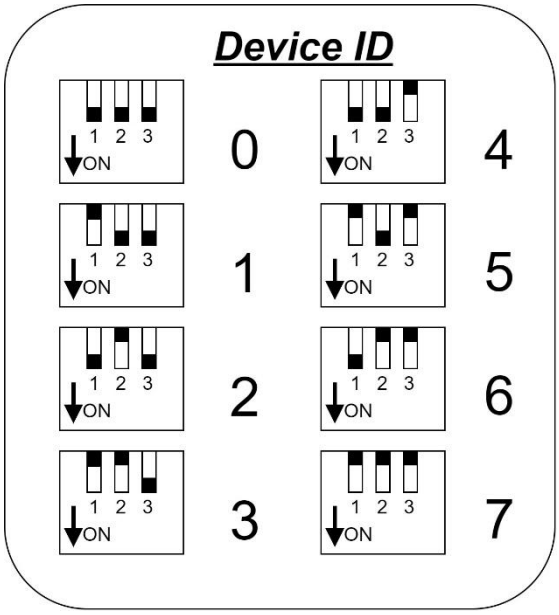
The start/stop of the pulse output will be notified in the Telematics Channel as on/off.

Inverse and Non-Inverse Mode

During Input mode, if the inverse mode is set, LOW signal will be notified to the server as ON. The example usage of the inverse mode is as below,

During Push Button input mode, the input push button is connected to GND when the Push Button is pressed. By enabling the inverse mode on the channel, when a user presses the push button, the system will detect the user action as ON.

During Output mode, if the inverse mode is disable, when the server/host is request for ON output, the output port will turn on the NPN transistor causing the connected device to turn on.



Configuration of the I/O port as Input port or Output port

The I/O Extension Unit hardware will contain a total of 16 I/O channels indicated as CH0 to 15. The Remote I/O Main Controller will not be able to know which port is allocated as Input port or as Output port on the I/O Extension Unit hardware.

It is up to the user to determine the configuration of the hardware and configure the main controller accordingly. The setting will be kept in the non-volatile memory of the device once configuration is done.

Each I/O Extension Unit will have a DIP switch to configure the Device ID of the device. Please make sure there is no Device ID duplication. The system can connect to a total of 16 Channel x 8 devices.

Please take note that if the Remote I/O Main Controller unit is also having 16 Channel I/O port unit, the default Device ID for the built-in device is 0.

During power up of the Remote I/O Main Controller unit and the I/O Extension Unit, the system will scan the Bus for all the connected devices. No new device should be added after the system is powered up. If required to add a new device, the system must cycle the power to allow the system to re-scan the bus for the new device.

Telematics Data

Key	Value Type	Description
t. ioextension.param	Array of object	Return the status of each port in the array if the status of the array changes.

Each element of array will consist of value listed below if it is set as input port,

```
{
        "id": 0,
        "state": [
            {
                "ch": 0,
                "mode": 6,
                "ison": true,
                "ts": 34245324,
                "count": 554
            },
            {
                "ch": 1,
                "mode": 6,
                "ison": false,
                "ts": 34245324,
                "count": 554
            },
            ...
        ]
}
```

if it is set as Push Button input port,

```
{
        "id": 0,
        "state" : [
            {
                "ch": 1,
                "mode": 6,
                "btnpress": 2,
                "ts": 34245323,
                "count": 554
            },
            ...
        ]
}
```

- id = the Device ID.
- ch = Input Channel Index.
- mode = the current mode setting of the Channel.
- ts = System timestamp in millisecond.
- ison = true if it is ON and false when it is OFF.
- btnpress can be either
    - 1 for normal press,
    - 2 for double click the button,
    - 3 for triple click the button,
    - 4 for long press for more than 4 second.

- count = number of on/off cycle in Standard mode and number of buttons press cycle count in Push Button mode.

If it is set as Output Port,

```
{
    "id": 0,
    "state": [
        {
            "ch": 3,
            "mode": 1,
            "ison": true,
            "ts": 342352
        },
        …
    ]
}
```

Where,

- id = the Device ID.
- ch, is the port channel index.
- mode = the current mode setting of the Channel.
- ison, set to true if it is ON and false when it is OFF in Standard Mode. Will set to true when start the pulse output and set to false when pulse output ended in Pulse Mode.
- *ts, system timestamp in millisecond*

Key	**Value Type**	**Description**
t. ioextension.dev	Array of object	Return the device connected and disconnected message.

The message formats are,

```
{
    "id": 0,
    "conn": true,
    "ts": 34245323,
}
```

Where,

- id = the Device ID.
- conn = is "true" if the device just connected and "false" if the device is just disconnected.
- ts, system timestamp in millisecond

RPC Call and Response

Key	Value Type	Description
r. ioextension.param	Array of Object	Set the output port.

Each element in the array will be consist of,

```
{
   “id”: 0,
   “rpc”: [
      {
         “ch”: 0,
         “ison”: true
      }
   ]
}
```

Where,

- id, is the device id.
- ch, is the channel index.
- ison, set to true of false, for further detail, please refer below for Standard Mode and Pulse Mode.

During Standard mode, set “ison” to true will turn on the output, set it to false will turn it off.

During Pulse mode, set “ison” to true will start the pulse output.

During the output port is sending out the output pulse,

- If “ison” is set to true again, it will reset and restart the pulse counting internal counter and continue with the pulse output.
- If “ison” is set to false, it will immediately stop the pulse output and set the output port to off.

Return result in RPC Call Response
r.outputport.result = true if the command process successfully.

Command Example

RPC Req.: set Dev#0 All Output ON

```
{
 "r.ioextension.param": [
  {
   "id": 0,
   "rpc": [
    {"ch": 0, "ison": true},
    {"ch": 1, "ison": true},
    {"ch": 2, "ison": true},
    {"ch": 3, "ison": true},
    {"ch": 4, "ison": true},
    {"ch": 5, "ison": true},
    {"ch": 6, "ison": true},
    {"ch": 7, "ison": true}
   ]
  }
 ]
}
```

RPC Req.: set Dev#0 All Output OFF

```
{
 "r.ioextension.param": [
  {
   "id": 0,
   "rpc": [
    {"ch": 0, "ison": false},
    {"ch": 1, "ison": false},
    {"ch": 2, "ison": false},
    {"ch": 3, "ison": false},
    {"ch": 4, "ison": false},
    {"ch": 5, "ison": false},
    {"ch": 6, "ison": false},
    {"ch": 7, "ison": false}
   ]
  }
 ]
}
```

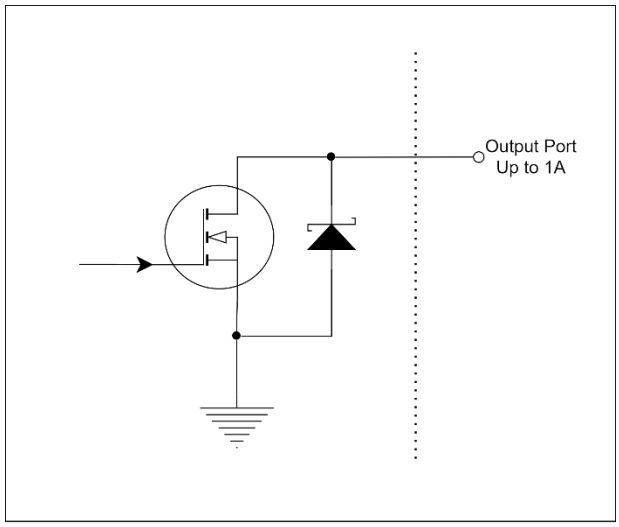
Attb. Setting: Dev0/CH0 to Output Pulse Mode with each cycle Off 0.2s, On 0.5s, and Total 3 Cycle.

```
{
 "s.ioextension.setup": [
  {
   "id": 0,
   "config": [
    {"ch": 0, "mode": 1, "fperiodon": 2,
"fperiodoff": 5, "pulsecnt": 3}
   ]
  }
 ]
}
```

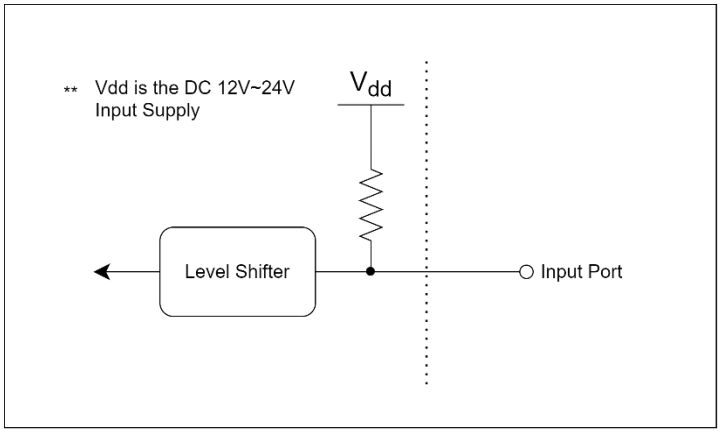
Attb. Setting: Dev0/CH0, 1, 2 and 3 to Standard Output Mode

```
{
 "s.ioextension.setup": [
  {
   "id": 0,
   "config": [
    {"ch": 0, "mode": 0},
    {"ch": 1, "mode": 0},
    {"ch": 2, "mode": 0},
    {"ch": 3, "mode": 0}
   ]
  }
 ]
}
```

Typical Input and Output Channel Design



Typical Output Port Design



Typical Input Port Design