Modbus Protocol
for BPM

Albércorp.

990 South Rogers Circle, Suite 11
Boca Raton, FL 33487
Tel: 561-997-2299 Fax: 561-997-5588
www.alber.com

Information in this document is subject to change without notice.

Modbus Protocol for BPM, Book Revision 1.0

BPM Modbus Communications Protocol

BPM user parameters are stored in a structure variable **user_parameters**. A Modbus controller can modify the operation parameters stored in the BPM memory and collect the cycle and alarms information.
(ASCII FRAMING MODE)

1. The User Structure Variables

There are four structures in the BPM, one each for the four batteries that it monitors.

```
struct user_parameters {        /* user parameters/ battery */
 unsigned char
          char_spare[6]
 unsigned int
                  if_lo,        /* user input float current-Lo val */
                  if_hi,        /* user input float current-Hi value */
                  ov_lo,        /* user input Volts-Lo value */
                  ov_hi,        /* user input Volts-Hi value */
                 sensor,        /* IF sensor selection settings */
                  shunt,        /* IL sensor  */
              amp_rating,       /* battery IL   amps rating */
               mV_rating,       /* battery IL mVolts rating */
                  cycles,       /* number of cycles recorded */
                  alarms,       /* number of alarm records recorded */
              int_spare1,       /* spare */
              int_spare2;

 float
                  rating,       /* load current rating factor */
             shunt_ratio,       /* IL shunt rating */
                 s_volts,       /* channel scaling factor */
                s_hiamps,       /* channel scaling factor */
                s_loamps,       /* channel scaling factor */
            float_spare1,
            float_spare2;
};
```

NOTE: char = 1 byte
int = 2 bytes
float = 4 bytes (Conforms to IEEE-754 standard.)
TRUE = non-zero value;
FALSE = Zero value;

1.1. User_parameters Structure to Holding Registers Mapping

Modbus commands [03 03h]-read holding registers, [06 06h]-preset single register, [16 10h]-preset multiple registers can be used to read/modify the parameters. (See Meaning, Type, and Valid Values for user_parameters section for meaning and valid data.)

Battery	Reference Offset	Holding Registers (4xxxx)	Address(dec)	Address(hex)
1	40001	1 - 29	0000-0028	0000-001C
2	40030	30 - 58	0029-0057	001D-0039
3	40056	59 - 87	0058-0086	003A-0056
4	40088	88 - 116	0087-0115	0057-0073

BATTERY 1

PARAMETER	OFFSET(bytes)	HOLDING REG(S)	ADDRESS
unsigned char			
char_spare[6]	0000(0000H)	40001,40002 40003	0000 (0000H)
unsigned int			
if_lo	0006(0006H)	40004	0003 (0003H)
if_hi	0008(0008H)	40005	0004 (0004H)
ov_lo	0010(000AH)	40006	0005 (0005H)
ov_hi	0012(000CH)	40007	0006 (0006H)
sensor	0014(000EH)	40008	0007 (0007H)
shunt	0016(0010H)	40009	0008 (0008H)
amp_rating	0018(0012H)	40010	0009 (0009H)
mV_rating	0020(0014H)	40011	0010 (000AH)
cycles	0022(0016H)	40012	0011 (000BH)
alarms	0024(0018H)	40013	0012 (000CH)
int_spare1	0026(001AH)	40014	0013 (000DH)
int_spare2	0028(001CH)	40015	0014 (000EH)
float			
rating	0030(001EH)	40016,4001	0015 (000FH)
shunt_ratio	0034(0022H)	40018,40019	0017 (0011H)
s_volts	0038(0026H)	40020,40021	0019 (0013H)
s_hiamps	0042(002AH)	40022,40023	0021 (0015H)
s_loamps	0046(002EH)	40024,40025	0023 (0017H)
float_spare1	0050(0032H)	40026,40027	0025 (0019H)
float_spare2	0054(0036H)	40028,40029	0027 (001BH)

BATTERY 2

PARAMETER	OFFSET(bytes)	HOLDING REG(S)	ADDRESS
unsigned char			
char_spare[6]	0000(0000H)	40030,40031 40032	0029 (001DH)
unsigned int			
if_lo	0006(0006H)	40033	0032 (0020H)
if_hi	0008(0008H)	40034	0033 (0021H)
ov_lo	0010(000AH)	40035	0034 (0022H)
ov_hi	0012(000CH)	40036	0035 (0023H)
sensor	0014(000EH)	40037	0036 (0024H)
shunt	0016(0010H)	40038	0037 (0025H)
amp_rating	0018(0012H)	40039	0038 (0026H)
mV_rating	0020(0014H)	40040	0039 (0027H)
cycles	0022(0016H)	40041	0040 (0028H)
alarms	0024(0018H)	40042	0041 (0029H)
int_spare1	0026(001AH)	40043	0042 (002AH)
int_spare2	0028(001CH)	40044	0043 (002BH)
float			
rating	0030(001EH)	40045,40046	0044 (002CH)
shunt_ratio	0034(0022H)	40047,40048	0046 (002EH)
s_volts	0038(0026H)	40049,40050	0048 (0030H)
s_hiamps	0042(002AH)	40051,40052	0050 (0032H)
s_loamps	0046(002EH)	40053,40054	0052 (0034H)
float_spare1	0050(0032H)	40055,40056	0054 (0036H)
float_spare2	0054(0036H)	40057,40058	0056 (0038H)

BATTERY 3

PARAMETER	OFFSET(bytes)	HOLDING REG(S)	ADDRESS
unsigned char			
char_spare[6]	0000(0000H)	40059,40060 40061	0058 (003AH)
unsigned int			
if_lo	0006(0006H)	40062	0061 (003DH)
if_hi	0008(0008H)	40063	0062 (003EH)
ov_lo	0010(000AH)	40064	0063 (003FH)
ov_hi	0012(000CH)	40065	0064 (0040H)
sensor	0014(000EH)	40066	0065 (0041H)
shunt	0016(0010H)	40067	0066 (0042H)
amp_rating	0018(0012H)	40068	0067 (0043H)
mV_rating	0020(0014H)	40069	0068 (0044H)
cycles	0022(0016H)	40070	0069 (0045H)
alarms	0024(0018H)	40071	0070 (0046H)
int_spare1	0026(001AH)	40072	0071 (0047H)
int_spare2	0028(001CH)	40073	0072 (0048H)
float			
rating	0030(001EH)	40074,40075	0073 (0049H)
shunt_ratio	0034(0022H)	40076,40077	0075 (004BH)
s_volts	0038(0026H)	40078,40079	0077 (004DH)
s_hiamps	0042(002AH)	40080,40081	0079 (004FH)
s_loamps	0046(002EH)	40082,40083	0081 (0051H)
float_spare1	0050(0032H)	40084,40085	0083 (0053H)
float_spare2	0054(0036H)	40086,40087	0085 (0055H)

BATTERY 4

PARAMETER	OFFSET(bytes)	HOLDING REG(S)	ADDRESS
unsigned char			
char_spare[6]	0000(0000H)	40088,40089 40090	0087 (0057H)
unsigned int			
if_lo	0006(0006H)	40091	0090 (005AH)
if_hi	0008(0008H)	40092	0091 (005BH)
ov_lo	0010(000AH)	40093	0092 (005CH)
ov_hi	0012(000CH)	40094	0093 (005DH)
sensor	0014(000EH)	40095	0094 (005EH)
shunt	0016(0010H)	40096	0095 (005FH)
amp_rating	0018(0012H)	40097	0096 (0060H)
mV_rating	0020(0014H)	40098	0097 (0061H)
cycles	0022(0016H)	40099	0098 (0062H)
alarms	0024(0018H)	40100	0099 (0063H)
int_spare1	0026(001AH)	40101	0100 (0064H)
int_spare2	0028(001CH)	40102	0101 (0065H)
float			
rating	0030(001EH)	40103,40104	0102 (0066H)
shunt_ratio	0034(0022H)	40105,40106	0104 (0068H)
s_volts	0038(0026H)	40107,40108	0106 (006AH)
s_hiamps	0042(002AH)	40109,40110	0108 (006CH)
s_loamps	0046(002EH)	40111,40112	0110 (006EH)
float_spare1	0050(0032H)	40113,40114	0112 (0070H)
float_spare2	0054(0036H)	40115,40116	0114 (0072H)

1.2. Meaning, Type, and Valid Values for user_parameters

struct user_parameters
User parameters/battery structure

unsigned char
char_spare[6]
Spare char space for future parameters.

unsigned int

if_lo
User input float current-Lo value (0-999).

if_hi
User input float current-Hi value (0-999).

ov_lo
User input Volts-Lo value (0-600).

ov_hi
User input Volts-Hi value (0-600).

sensor
IF sensor.
Set to TRUE if sensor for this battery is turned on. Else FALSE to disable it.

Shunt
IL sensor selection settings.
This is a **read only register**. Writing to this register will **not** generate an **error** or the value to be changed. This value is set by the BPM when there is a valid **amp&mV** rating.

amp_rating
Battery load amps rating per mV (0-999).

mV_rating
Battery load mVolts rating (0-999).
When changing amp and mV rating, the amp rating must be set first. When writing to the mV rating register, the BPM will validate the values and **calculate** the **Shunt** value accordingly.

cycles
Number of cycles recorded (hits).
Read this value to find out number of cycles recorded. Set to 0 to clear cycles.

Alarms
Number of alarm records recorded.
Read this value to find out number of alarms recorded. Set to 0 to clear alarms.

int_spare1
int_spare2
For future use.

float

rating

This is **read only register**. No errors will be generated by writing to this address or the value to be changed. This value is the shunt_ratio*1000. The BPM calculates this value. This value is used to calculate the load current.

shunt_ratio

This is **read only register**. No errors will be generated by writing to this address or the value to be changed. This value is calculated by the BPM and is the amp_rating / mV_rating.

s_volts

Volts channel scaling factor.
To display the current voltage or to convert values in ad_counts. (See Alarms and Cycle Data section and Current AD Counts section.)
Volts= Ad_count_value * s_volts;

Calibrating the voltage is a matter of calculating the scaling factor;
S_volts = known voltage / Ad_count_value

s_hiamps

Load amps channel scaling factor.
To display the load current or to convert values in ad_counts. (See Alarms and Cycle Data section and Current AD Counts section.)
Amps = Ad_count_value * s_hiamps * rating;

Calibrating the load current is a matter of calculating the scaling factor;
S_hiamps =
known current / Ad_count_value * rating

s_loamps

Float channel scaling factor.
To display the float current or to convert values that are in ad_counts. (See Alarms and Cycle Data section and Current AD Counts section.)
mA = Ad_count_value * s_loamps;

Calibrating the float current is a matter of calculating the scaling factor;
mA = known float / Ad_count_value;

float_spare1
float_spare2

For future use.

NOTE on Calibration. To calibrate a voltage of a battery, take the known voltage and divide it by the current ad_count_value (see the Current AD Counts section) and set the scaling value to it.

2. Temperature Parameters

```
         PARAMETER       HOLDING REG(S)     ADDRESS

Unsigned char
         data_temp     40117             0116(0074H)
                  Flag for display format of temperature.
                  TRUE=fahrenheit,
                  FALSE=celsius.
                  Value is in upper byte of register;
                 data_temp = (reg_val >> 8) & 0xff;

unsigned int
              t_lo     40118             0117(0075H)
              t_hi     40119             0118(0076H)

                  Valid values for high and low temp alarms levels:
                  (0-200)f,( 0-93) c.
int
      temp_alarms           40120             0119(0077H)
                 Number of temp. alarms. Set to zero to clear out
alarms.

float
           s_temp            40121,40122       0120(0078H)
                 Temp. channel scaling factor.
```

To display the current temperature or to convert values that are in ad_counts. (See the Alarms and Cycle Data section and the Current AD Counts section.
Fahrenheit Temp = **(((ad_count_value * s_temp)- .098)/(0.0003773))*1.8 + 32**;
Celsius Temp = **((ad_count_value * s_temp)- .098)/(0.0003773);**

Calibrating the temperature:
s_temp (F) = **((known_temp –32)*.0003773/(1.8)+.098)/ ad_count_value;**
s_temp (C) = **((known_temp*.0003773)+.098) / ad_count_value;**

3. Alarm Flags

PARAMETER	HOLDING REG(S)	ADDRESS
unsigned char		
ov_alarm1	40123	0122(007AH)
ov_alarm2	40124	0123(007BH)
ov_alarm3	40125	0124(007CH)
ov_alarm4	40126	0125(007DH)
TRUE indicates battery overall voltage is out of range.		
If_float1	40127	0126(007EH)
If_float2	40128	0127(007FH)
If_float3	40129	0128(0080H)
If_float4	40130	0129(0081H)
TRUE indicates battery float alarm.		
h_alarm1	40131	0130(0082H)
h_alarm2	40132	0131(0083H)
h_alarm3	40133	0132(0084H)
h_alarm4	40134	0133(0085H)
TRUE indicates battery in hit mode.(discharge).		
t_alarm	40135	0134(0086H)
TRUE indicates temp alarm.		

NOTE: Alarm flags are read only registers. Writing to these registers will **not** generate an **error** or the value to be changed.
NOTE: Value is in upper byte.

flag = (reg_val >> 8) & 0xff;

4. Alarms and Cycle Data

Each battery has an alarm and cycle counter. These counters keep track of the number of alarms and cycles that occurred for each battery. To access the actual alarm or cycle data, the BPM must be polled for the information. The cycle and alarm data is accessed using Modbus command (20 14h)-**Read general reference.**
Cycle data is stored in file numbers 1- 4. For batteries 1 - 4.
Alarm data is stored in file numbers 5-8. For batteries 1 - 4.
Temp alarms are stored in file number 9. (See application notes for usage.)

The structure of the alarm/cycle:

```
struct data_record
unsigned char
      recrd_type,  /* record_types: OVH, OVL, THI, TLO, IFH, IFL */
           aux1;  /* for future use */
                Value is in upper byte of register.
                recrd_type= (regval >> 8)& 0xff;

int
     recrd_hcnts,  /* actual alarm level high prog. by user */
     recrd_lcnts,  /* actual alarm level low prog by user ) */
      recrd_cnts,  /* actual "ad_reading" in counts         */
  highest_dchg_i,  /* actual "ad_reading" for IL or IF */
     _30s_dchg_I;  /* actual "ad_reading" @30sec for IL or IF */

                These values are in ad_count_values to convert them to
                V,A,mA or Temp see scaling factors in section 3&4.
WHERE:
unsigned char
         month,       Value is upper byte of register.
                month = (regval >> 8)& 0xff;

          date,  Value is lower byte of register.
                date = (regval & 0xff);

          year,  Value is upper byte of register.
                year = (regval >> 8)& 0xff;

         hours,  Value is lower byte of register.
                hours = (regval & 0xff);

       minutes,  Value is upper byte of register.
                minutes = (regval >> 8)& 0xff;

       seconds,  Value is lower byte of register.
                seconds = (regval & 0xff);

unsigned long
recrd_duration;   record duration seconds 32 bit number
};
```

Alarms and cycle data. (cont)

Each **record** is **22 bytes** long or **11 registers** (Modbus).To access the data, first read the number of alarms/cycles for a given battery or temperature Then choose the file number associated with the alarm/cycle. Alarms/cycles # 1 starts at address 0000 or register 1.
To access the n'th alarm/cycle (n-1*11) =starting address of alarm/cycle.

Reg Offset	Variable(s);
0	recrd_type
1	recrd_hcnts
2	recrd_lcnts
3	recrd_cnts
4	highest_dchg_I
5	_30_dchg_I
6	month, date
7	year, hour
8	minutes, seconds
9, 10	duration

a) **Alarm Types**
1=voltage alarm.
2=float sensor alarm.
3=temp. alarm.
4=battery cycle.
5=AC ripple.

All Alarms store the date, time and duration.

1) **Cycle Alarms** store the following information:
recrd_hcnts = ov @ 30 seconds.
recrd_lcnts = lowest ov duration cycle.
Recrd_cnts = ov @ start of cycle
Highest_dchg_I = max load current during cycle.
_30_dchg_I = load current @ 30 seconds.

2) All Other alarms(**OV,IF,TEMP,AC**) store the following information:
Recrd_cnts = alarm level @ start of alarm.(***value is in ad_counts**).
Recrd_hcnt = alarm level high program by user(***value is in ad_counts**).
Recrd_lcnt = alarm level low programmed by user(***value is in ad_counts**).

5. Current AD Counts

Associated with each channel is a count value that has the current analog reading value for a given channel. This value is the current **12 bit** AD value that is read from a given channel.(e.g., OV, IF, IL). This value must be converted to an actual voltage, current or temperature by multiplying it by the scaling factor. See scaling factors in the Meaning, Type and Valid Values for user_parameter section and the Temperature Parameters section for usage. (**NOTE**: Values can be positive or negative.)

PARAMETER	HOLDING REG(S)	ADDRESS int
OV1_count	40136	0135(0087H)
HI1_count	40137	0136(0088H)
LI1_count	40138	0137(0089H)
OV2_count	40139	0138(008AH)
HI2_count	40140	0139(008BH)
LI2_count	40141	0140(008CH)
O31_count	40142	0141(008DH)
H31_count	40143	0142(008EH)
LI3_count	40144	0143(008FH)
OV4_count	40145	0144(0090H)
HI4_count	40146	0145(0091H)
LI4_count	40147	0146(0092H)
T_count	40148	0147(0093H)

OV = Operation Voltage for Batteries 1 to 4.
HI = Load Current for Batteries 1 to 4.
LI = Float Current for Batteries 1 to 4.
T = Temperature.

The count values are updated about once a second, depending on the number of batteries and sensors.

6. References

This document may be found on the Albércorp Web site in the Technical Library/Modbus Register Maps section under BPM. Access Albércorp on the Web at: http://www.alber.com

Consult the following source for assistance with the Modbus protocol:

Modicon Modbus Protocol Reference Guide.
Modicon, Inc., Industrial Automation Systems, 1 High Street, North Andover, MA 01845
Modicon, Inc. is on the Web at: http://www.modicon.com

Consult the following source for third party test software to access all of the BPM Modbus registers:

Wintech Software, P.O. Box 907, Lewisburg, WV 24901
Wintech is on the Web at http://www.win-tech.com
The test software is Modscan.