EZ Protocol

Communication Protocol for EZPLC

For use of EZAutomation and AVG Customers with EZPLC Products

EZAutomation
www.ezautomation.net
1-877-774-EASY

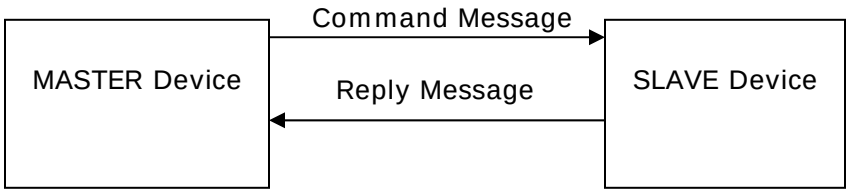
EZ Protocol

Communication Protocol for Use with EZPLC family of Products

1. Intoduction

Many of EZAutomation products, such as EZPLC, use EZProtocol for serial communications. This document describes a subset of EZProtocol that would allow reading and writing data from EZPLC.

The EZProtocol communication is a master-slave protocol. Master Device sends command messages and Slave Devices responds to the commands by sending a reply message. The EZPLC in this communication acts as a slave, and responds to communications commands sent by a Master. This is illustrated in the diagram below:



Master Device initiates all communications.
Slave devices (EZPLC) checks communication packet *only* for check sum errors. The devices do not validate packet contents, i.e. EZPLC would not check if a Master has requested data from an invalid address. ***It is Master devices responsibility to ensure that the message packet fields have correct values***.

2. Device Address

The Protocol allows for one master to communicate with multiple slaves using EZProtocol. However, Master can talk to only one slave at one time. Since there is only one master, there is no address associated with the Master Device. Only Slave Devices have address.

EZ Protocol requires that each slave device must have a unique address. The address is divided in two parts:

- a 4-bit group number and
- a 12-bit unit number.

(*Currently the Group and Unit number on EZPLC are fixed as 1 and 1 respectively and cannot be changed by user*).
The commands are addressed to a specific unit by using its group and unit number. The protocol allows 4 bits for the group number and 12 bits for the unit number. Though the protocol allows up to 16 groups and up to 4096 unit numbers, the actual numbers will depend up on the product. In some product these are preset and user does not change them; however the protocol requires that the commands use these preset group and unit numbers (even though they are fixed).

These commands can be used over RS232, RS422 or Ethernet Medium.

3. Command and Reply Message Formats

This section describes the general format of the command and reply messages. Next section describes a subset of commands that would allow you to read/write data from EZPLC.

Each command can either write-to or read-from the Slave Device. In response to a command, slave device would send a reply.

3.1 Reply to Write Commands

For **Write commands**, the reply acknowledges the command and includes an error code if there were any errors in executing the command. This type of reply (to acknowledge a command and provide possible error code) has a common format, and is used for all write commands as well as for read commands in case of errors. This type of reply is called "**Standard Reply**" in EZ Protocol.
(The EZPLC checks only for the checksum errors in commands, and does not check for the validity of other fields in the command message packet. For write commands, EZPLC also checks to make sure that the PLC is in Prog Mode before performing writes.)

3.2 Reply to Read Commands

For **read commands**, the reply would normally contain the requested data. The reply format depends upon the command and is described along with the respective command code description later in this document. If there is some problem in the command (say checksum error), the slave device returns "**Standard Reply**" (instead of the normal reply) possibly with an error code. (The EZPLC checks only for the checksum errors in commands, and does not check for the validity of other fields in the command message packet.)

3.3 Command Messages and Read-Replies Format:

(Used for Command Messages and, in case no check sum errors in Command Messages received by Slave, for replies to Read commands)

Byte Offset	No of Bytes	Value	Comment
0	1	0xAA	Start character
1	1	Command Length	Length in Bytes of the message packet - 1
2	1	(Group<<4) | (Unit&0x0fff >>8)	Group# in Bits b7-b4 Upper 4 bits of Unit# in bits b3-b0 (The Group and Unit numbers are for the unit the message is sent to)
3	1	Unit & 0x00ff	Lower 8 bits for the Unit Number in this byte
4	1	COMMAND CODE	See allowed commands
5	n	Data bytes	Depends up on the commands
5+n	2	Checksum (LSB, MSB order)	CRC checksum of all bytes of the message, except the Start byte (0xAA)

3.4 Standard Reply Format:

(Used for replies to ALL write commands and, in case of errors, for Read commands)

Byte Offset	No of Bytes	Value	Comment
0	1	0xAA	Start character
1	1	8 (Message Length)	Length in Bytes of the message packet - 1
2	1	(Group<<4) | (Unit&0x0fff >>8)	Group# in Bits b7-b4 Upper 4 bits of Unit# in bits b3-b0 (The Group and Unit numbers are for the unit the message is sent to)
3	1	Unit & 0x00ff	Lower 8 bits for the Unit Number in this byte
4	1	0x00	Reply code for Standard Reply
5	1	COMMAND Code Echo	Same as Received in Command Message
6	1	ERROR Code	
7	2	Checksum (LSB, MSB order)	CRC checksum of all bytes of the message, except the Start byte (0xAA)

3.5 Message Field Definitions

The fields in the message packets are described below:

Start Character:
EZProtocol uses 0xAA as a Start character for all message packets.

Length:
This byte represents Length of message packet in bytes. The start character is not included in the Length. Thus:

```
Length = total number of bytes in the message - 1
```

Group and Unit Numbers
(For EZPLC family of products the Group and Unit Numbers are fixed to:
Group Number = 1
Unit Number = 1)
Each Slave has a Group and Unit Number. Two bytes are used to encode group and unit numbers. First of the two bytes has group number in upper 4 bits, and upper 4 bits of the unit number in the lower 4 bits. The second byte contains lower 8 bits of the Unit number.

Following pseudo code illustrates this:

```
int group, unit;
unsigned char message[255];
.
.
message[2]= ((group&0x000f)<<4) | ((unit & 0x0fff)>>4);
message[3]= unit & 0x00ff;
.
.
```

COMMAND Code
The EZ Protocol allows several commands. Following codes are described later in this document:
Please note that these codes can be used ONLY for the EZPLC (as well as for the EZPLC integrated in EZText PLC and EZTouchPLC)

Command Code	Description
18	Get Device Status and ID
19	Read a Block of Discrete
20	Read a Block of Registers
21	Read multiple elements
25	Write multiple elements

Data Bytes

Most commands require some data. For example to read data, command would require address. Data Bytes contain all the data required by command. The format of the data is described in the description of each command later in this document.

ERROR Code (*applicable to Standard Reply only*)

Slave devices check the incoming message for Checksum. If checksum is incorrect, the Slave Devices send the standard reply with error code set to 1. For write commands, the EZPLC also ensures that the PLC is in Prog (or Edit) mode. EZPLC does not check or validate any field. ***It is master devices responsibility to ensure that the various fields in the communication packet to the Slave device are validated.*** Below is the list of error codes returned by the Slave Devices.

ERROR Code	Description
0	No Error
1	Check Sum Error
21	PLC not in Prog mode (Generated when Master is trying to write values with PLC not in Prog Mode)

CHECKSUM

The Message packets are appended by a checksum. The checksum is a two byte value. The checksum of a message is calculated using the bytes that represent length, unit number, message code, and data. Please note that Start character is not included in this computation.

EZ Protocol uses CRC for checksum computations. The algorithm for the CRC computation is described later in this document.

4. Message Details

This section describes command messages and corresponding replies. Hex numbers are used with 0x prefix. All other numbers used are in decimal format.

4.1 Get STATUS/ID Command (Command code 18)

Description

This command returns Status and several other pieces of information about the unit. These include hardware & firmware (Boot as well as Exec) revisions, various memory sizes and run state of the device.

Command Message

Byte Offset	Value	Description
0	0xAA	Start character
1	06	Length
2	0x10	Group/unit number (2 bytes) of the Salve Device the command is being sent to. Group = 1, and Unit = 1 for EZPLC
3	0x01	
4	18	Command code
5	0xb8	Checksum LSB
6	0x65	Checksum MSB

Reply Message for Command Code 18

Byte Offset	Value	Description
0	0xAA	0xAA Start flag
1	37	Length
2	0x10	Group/unit number (2 bytes) fixed at 1, 1
3	0x01	
4	18	Command code echo
5	0x60	Device type code:
6		Hardware revision (Number 0, 1, ..)
7		Boot software Major revision (ASCII character code, e.g. A, B, etc)
8		Boot software revision, minor number (ASCII code of number, e.g. “0”, “1”, etc)
9		Exec software major revision (ASCII character code, e.g. A, B, etc) (‘?’ = no exec)
10		Exec software revision minor number (ASCII code of number, e.g. “0”, “1”, etc) (‘?’ = no exec)
11		Exec flash size (0=PROM) (4 bytes) (MSB to LSB)
15		Size of user memory in bytes (4 bytes, MSB to LSB)
19		Amount of free memory in Bytes (4 bytes, MSB to LSB)
23		User program size in Bytes (4 bytes, MSB to LSB)
27		System memory size (4 bytes) (MSB to LSB)
31		Executing boot /exec: 0 = boot, 1 = exec
32		Run status 0 = STOPPED, 1 = RUNNING
36		Checksum LSB
37		Checksum MSB

4.2 BLOCK READ DISCRETES (19)

Description: Reads a block of discretes from the PLC. The data is bit-packed, with b0 as the first discrete value. The discrete address MUST be a multiple of 16.

NOTE: *The multi-byte values within messages are sent MSB-LSB.*

Command Message

Byte Offset	Value	Description
0	0xAA	0xAA Start flag
1	10	10 Length
2	0x10	Group/unit number (2 bytes)
3	0x01	
4	19	Command code
5		Discrete type (from the table below)
6		MSB of Discrete address
7		LSB of Discrete address
8	N	Number of discrete to read (Must be multiple of 16)
9		Checksum (LSB)
10		Checksum (MSB)

Reply Message for Command Code 19

Byte Offset	Value	Description
0	0xAA	Start flag
1	10+(N+7) /8	Length (N= Number of Discrete read)
2	0x10	Group/unit number (2 bytes)
3	0x01	
4	19	19 Command code echo
5		Discrete type (see below)
6		Discrete address (MSB)
7		Discrete address (LSB)
8		N = Number of discretes read (is Multiple of 16)
9		Bit-packed discrete values Bit B0 for discrete address 1, bit B1 for address 2, and so on.
9+(N+7) /8		Checksum (2 bytes)

Discrete Types	
0	I
1	O
2	S
6	SD

4.3 BLOCK READ REGISTERS (20)

Description: Reads a block of Registers from the EZPLC. The Block of registers starts at "Register Address."

NOTE: *The multi-byte values within messages are sent MSB-LSB.*

Command Message

Offset	Value	Description
0	0xAA	Start flag
1	10	Length
2	0x10	Group/unit number (2 bytes)
3	0x01	
4	20	Command code
5		Register type (from table below)
6		Register address (MSB)
7		Register address (LSB)
8	N	Number of registers to read
9		Checksum LSB
10		Checksum MSB

Reply Message for command code 20

Offset	Value	Description
0	0xAA	Start flag
1	10+N* 2	Length
2	0x10	Group/unit number (2 bytes)
3	0x01	
4	20	Command code echo
5		Register type (See Below)
6		Register address (MSB)
7		Register address (LSB)
7	N	N = Number of registers read
9		Register values (N* 2 bytes)
9+N*2		Checksum (LSB)
9+N*2+1		Checksum (MSB)

Register Types	
3	IR
4	OR
5	R
7	SR
17	XR
18	#R
19	XS
20	#S

4.4 READ MULTIPLE ELEMENTS (21)

Description: Reads a list of elements from the PLC. There is a maximum of 78 elements per message. The elements can be a combination of discretes and registers. The addresses are not included in the reply; only values are included. The values in the reply are in the same order as the address list in the message. Each discrete value uses one byte (B0 = value of discrete and each register value uses two bytes.

NOTE: *The multi-byte values within messages are sent MSB-LSB.*

Command Message

Offset	Value	Description
0	0xAA	Start flag
1	7+N*3	Length
2	0x10	Group/unit number (2 bytes)
	0x01	
4	21	Command code
5	N	N = Number of elements to read
		Elements types and addresses (3 bytes per element) of N elements
6		Element Type of first element
7		MS Byte of element Address
8		LS Byte of element address
9		Element Type of second element
.		.
.		.
6+N*3		Checksum (LSB)
6+N*3+1		Checksum (MSB)

Reply Message for Command Code 21

Offset	Value	Description
0	0xAA	Start flag
1	10+L	Length
2	0x10	Group/unit number (2 bytes)
3	0x01	
4	21	21 Command code echo
5	N	N = Number of elements read
9		Element values (L bytes) One byte for a Discrete type, and two bytes for a Register type
9+L		Checksum (LSB)
9+L+1		Checksum (MSB)

Element types:

Discrete Types	
0	I
1	O
2	S
6	SD
Register Types	
3	IR
4	OR
5	R
7	SR
17	XR
18	#R
19	XS
20	#S

4.5 WRITE MULTIPLE ELEMENTS (26)

(Write values to multiple Registers/discrete elements) (25)

Description: Write the values to the registers and discrete memory locations. The type (1 byte), address (2 bytes), and value (2 bytes) is passed for each register. Two bytes for values are used even for Discrete types. For discrete types, if B0 = 0 (of LSB of 2 value bytes) the discrete is turned OFF and if B0 = 1 (of LSB) the discrete is turned ON.

NOTE: *The multi-byte values within messages are sent MSB-LSB.*

Command Message

Byte Offset	Value	Description
0	0xAA	Start flag
1	7+N*5	Length
2	0x10	Group/unit number (2 bytes)
3	0x01	
4	25	Command code
5	N	N = Number of registers to write
6		Register or Discrete type for first element (see table below)
7		Address MSB
8		Address LSB
9		Value MSB
10		Value LSB
11	.	.
12	.	.
6+N*5		Checksum (LSB)
6+N*5+1		Checksum (MSB)

Reply Message for Command Code 25

Standard Reply

Element Types:

Discrete Types	
0	I
1	O
2	S
6	SD
Register Types	
3	IR
4	OR
5	R
7	SR
17	XR
18	#R
19	XS
20	#S

5. CRC Checksum Algorithm

The Cyclic Redundancy Check (CRC) uses the polynomial

$$X^{16} + X^{12} + X^5 + 1$$

The algorithm used to compute the polynomial is:

Use 3 working registers: <high>, <low> and <temp>

Initialize <high> and <low> to 0

For each character, starting with first character, perform the following steps:

(1) <temp> = <character> exclusive-or <high>
(2) <temp> = (<temp> shifted right 4 bits) exclusive-or <temp>
(3) <high> = (<temp> shifted left 4 bits) exclusive-or <low>
(4) <high> = (<temp> shifted right 3 bits exclusive-or <high>
(5) <low> = (<temp> shifted left 5 bits) exclusive-or <temp>

After the last character has been incorporated the CRC high byte is in <high> and the low byte is in <low>.