Bottom communication protocol function of servo**(Steer_protocol.h)**

If you want to understand the underlying details of the servo communication, read on, and if you just want to use the servo, refer to the application layer in Chapter 1 to solve most of the problems.

If you have special reasons to know the details of servo control, it is best to cooperate with the 7bot mechanical arm servo consultation, as well as the source program Steer_protocol.c file and Steer_protocol.h file to read, the source file gives a more complete comments. You can see our operation of the underlying register, this document is just for ease of reference to the servo function and basic usage。

2.0 Macro definition of instructions and registers in the underlying communication protocol of the servo

Table20. All instruction descriptions of **Steer_protocol.h** are given.

instruct	macro definition	describe
0X01	INSTRUCTION_PING	Query instruction
0X02	INSTRUCTION_READ_DATA	read instruction
0X03	INSTRUCTION_WRITE_DATA	Write instruction
0X04	INSTRUCTION_REG_WRITE	Asynchronous write instruction
0X05	INSTRUCTION_ACTION	Trigger asynchronous write instruction
0X06	INSTRUCTION_RESET	reset instruction

Table21. The register address description of **Steer_protocol.h** is given.

Register address	macro definition	describe
0X05	ID_REG	Address of ID: by modifying the value of this position, you can modify the address of the servo (address: 0~253)
0X09	MIN_ANGLE_LIMIT_H	high-byte storage location limited by Minimum angle
0X0A	MIN_ANGLE_LIMIT_L	Low byte storage location limited by minimum angle
0X0B	MAX_ANGLE_LIMIT_H	High byte high byte storage location limited by maximum angle
0X0C	MAX_ANGLE_LIMIT_L	Low byte storage location limited by maximum angle
0X10	MAX_TORQUE_H	Maximum torque high byte storage location
0X11	MAX_TORQUE_L	Maximum torque low byte storage location
0X12	DEFAULT_SPEED	Adjusting velocity position
0X14	MIDDLE_POSITION_H	Median adjustment of high byte storage location
0X15	MIDDLE_POSITION_L	Middle bit adjusts low byte storage location
0X28	TORQUE_SWITCH	Torque switch (1 open and 0 turn)
0X2A	TARGET_POSITION_H	Target location High byte storage location
0X2B	TARGET_POSITION_L	Target location low byte storage location
0X38	CURRENT_POSITION_H	Current position high byte storage location
0X39	CURRENT_POSITION_L	当前位置低字节存储位置
0X41	TARGET_SPEED	速度调整

0XFE	BROADCAST_ADDR	broadcast address

2.1 Variables in the **Steer_protocol** class of the communication protocol layer at the bottom of the servo

This class is defined in Steer_protocol.h, see this file for details.

Table22. gives a list of public variables for Steer_protocol

Public variable name(public)	describe
no	no

Table23. gives a list of private variables for Steer_protocol

Private variable name (private)	describe
svSer	Serial selection: for example, &Serial1

Table24. gives a list of the public functions of **Steer**

function name	describe
Steer_protocol	servo communication protocol constructor: initialize serial communication port
Set_Serial_init	Initialize communication serial port
begin	Initialize in communication with the computer, using the user only when testing the servo
Check_Sum	Checksum function
ping	Working status query function
read	Read the state function of the servo
reset	servo reset function
write	Write function
sync_write	Synchronous write function (broadcast)

2.1.0 **Steer_protocol()=default;** (note: this function is overloaded, and the next table describes its overloaded function)

Table25. Describes the function **Steer_protocol Table25**.

function name	Steer_protocol
function prototype	Steer_protocol() = default;
functional description	Default constructor
input parameter	no
returned value	no
prerequisite	Initialize a **Steer_protocol** object
Called function	no

Example:

/**Create a **Steer_protocol** object and initialize it by default**/
Steer_protocol steer_protocol();

2.1.1 函数 Steer_protocol;（注意:该函数重载，下个表格介绍其重载函数）

Table26.描述了函数 Steer_protocol Table26.

function name	Steer_protocol
function prototype	Steer_protocol(HardwareSerial *serial, long timeout)
functional description	servo communication protocol constructor: initialize serial communication
Input parameter 1	Serial: Serial port selection
Input parameter 2	Timeout：Serial port timeout
returned value	no
prerequisite	Initialize a **Steer_protocol** object
Called function	no

example:

/**Create a **Steer_protocol** object and set its serial port to **serial1**, and timeout to 10 milliseconds**/

Steer_protocol   steer_protocol(&Serial1, 10);

2.1.2 function **Set_Serial_init**

Table27.

function name	Set_Serial_init
function prototype	void Set_Serial_init(HardwareSerial *serial)
functional description	To provide an initialization interface for its parent class Steer
input parameter	Serial: Serial port selection
returned value	no
prerequisite	Initialize a **Steer_protocol** object
Called function	no

example:

/**Create a **Steer_protocol** object and initialize it**/

Steer_protocol   steer_protocol;

steer_protocol.Set_Serial_init(&Serial1);

2.2.2 function **begin**

Table28.

function name	begin
function prototype	void begin(HardwareSerial *serial, long timeout)
functional description	The user uses the servo only when testing it, communicating with the computer
Input parameter 1	Serial: Serial port selection
Input parameter 2	Timeout：Serial port timeout
returned value	no
prerequisite	Initialize a **Steer_protocol** object
Called function	no

例:

/**Create a **Steer_protocol** object and set its serial communication port to **serial1**, timeout 10 milliseconds, and initialize the communication between the board and the computer**/

Steer_protocol   steer_protocol(&Serial1, 10);

steer_protocol,begin(&Serial1, 10);

2.2.3 function **Check_Sum**

Description: checksum formula: check Sum = ~ (ID Length Instruction Parameter1. Parameter N); specific please see the 7bot robot arm servo Communication Protocol

Table29.

function name	Check_Sum
function prototype	byte Check_Sum(byte *buf, byte len);
functional description	checksummat
Input parameter 1	buf: Array address to be checked
Input parameter 2	len： Array length
returned value	Data type byte: checksum
prerequisite	Initialize a **Steer_protocol** object
Called function	no

Example:

/** creates a Steer_protocol object and outputs the last number of the the data buf as a checksum, set: byte buf[] = {1,2,3,4,5,6} so: len = 5 **/

```
Steer_protocol     steer_protocol(&Serial1, 10);
steer_protocol.Check_Sum (buf, 5);
```

2.2.4 function **ping**

Table30.

function name	ping
function prototype	boolean ping(byte id, byte *data)
functional description	Working status query function to test whether the servo answers
Input parameter 1	id: servo ID
Input parameter 2	data： Input a pointer of type byte to get the servo working state
returned value	True: the servo has a response and communication is normal False: no response from the servo, abnormal communication
prerequisite	Initialize a **Steer_protocol** object
Called function	no

Example:

/**Create a Steer_protocol object and set: byte *dat; Query the working status of the servo with ID 1**/

```
Steer_protocol     steer_protocol(&Serial1, 10);
steer_protocol. ping(0x01 , dat);
```

2.2.5 function **read**

Table31.

function name	read

function prototype	Boolean read(byte id, byte regStartAddr, byte *data, byte readlen)
functional description	Read the state function of the servo
Input parameter 1	id: servo ID
Input parameter 2	regStartAddr： Memory start address to read information
Input parameter 3	data： Input pointer of type byte to store read information
Input parameter 4	readlen： Length of data read
returned value	true: 读取成功 false: 读取失败
prerequisite	Initialize a **Steer_protocol** object
Called function	no

Example:

/**Create a Steer_protocol object and set the servo ID = 1; read two bytes from the address 0X38 in the control table, Steer_protocol Steer1; byte dat[2]; **/
Steer_protocol steer_protocol(&Serial1,10);

steer_protocol.read(0x01,0x38,dat,0x02);

2.2.6 function **reset**

2.2.7 Table32.

function name	reset
function prototype	void reset(byte id)
functional description	servo reset function: restores the servo to factory setting
input parameter 1	id: servo ID
returned value	Void
prerequisite	Initialize a Steer_protocol object
Called function	write

Example:

/**Create a Steer_protocol object and set the servo ID = 1 to restore the factory settings**/
Steer_protocol steer_protocol(&Serial1,10);

steer_protocol. reset(0x01);

2.2.8 function **write**

Table33.

function name	write
function prototype	void write(byte id, byte regStartAddr, byte *buf, byte bufLen)
functional description	Write function
Input parameter 1	id: servo ID
Input parameter 2	regStartAddr： Memory start address for information to be written
Input parameter 3	buf: Enter an array pointer of type byte to store

	information that needs to be written
Input parameter 4	bufLen:Length of data written
returned value	void
prerequisite	Initialize a Steer_protocol object
Called function	no

Example:

/**Create a Steer_protocol object and set up the steering engine ID = 1; write two bytes from the address 0X2A in the control table as the target location, byte dat[2] = { 0x00 , 0xff }**/
Steer_protocol steer_protocol(&Serial1, 10);

steer_protocol. write(0x01 , 0X2A , dat , 0x02);

2.2.9 function **sync_write**

Table34.

function name	sync_write
function prototype	void sync_write(byte regStartAddr, byte *buf, byte svNum, byte perDataLen)
functional description	Synchronous write function: you can write several data to one or more servo at the same time, commonly used to write simultaneously the target position and the running time to form the speed control
Input parameter 1	regStartAddr： Memory start address for information to be written
Input parameter 2	buf: Enter an array pointer of type byte to store information that needs to be written
Input parameter 3	svNum： Number of servo to be written synchronously
Input parameter 4	perDataLen:Length of data to be written to each servo
returned value	void
prerequisite	Initialize a **Steer_protocol** object
Called function	no

Example:

/**Create a Steer_protocol object and write two bytes from the address 0X2A in the control table for the target position, bytedat[]={0x01,0x00,0xff........0x06,0x00,0xff }; **/
Steer_protocol steer_protocol(&Serial1,10);

steer_protocol.sync_write(0X2A,buf,6,3);