KST Servo Drone CAN Interface Description

Document modifications			
RN	**Date**	**Author**	**Modifications**
V1.01	2025-02-19	Lanmz	Creation;

The KST Drone CAN interface V1.01.

1．Drone CAN interface support：

1.1 Node status report

Interface ID : UAVCAN_PROTOCOL_NODESTATUS_ID

Interface Object : uavcan_protocol_NodeStatus

1.2 Run status reprot

Interface ID:UAVCAN_EQUIPMENT_ACTUATOR_STATUS_ID

Interface Object :uavcan_equipment_actuator_Status

1.3ESC status report

Interface ID:UAVCAN_EQUIPMENT_ESC_STATUS_ID

Interface Object :uavcan_equipment_esc_Status

1.4 Read Servo Information

Interface ID:UAVCAN_PROTOCOL_GETNODEINFO_ID

Interface Object :uavcan_protocol_GetNodeInfoResponse

1.5 Read and Write parameters

Interface ID:UAVCAN_PROTOCOL_PARAM_GETSET_ID

Interface Object : parameter list

Index	Parameter	Range	Default	
0	Baud	0~8	8	0~8:10k,20k,50k,100k, 125k,250k,500k,800k,1M
1	channel	0~15	0	
2	Control parameter	0、1	0	0 : Raw data 1 : array
3	Mid point Setting	1		Set the current position as the midpoint
4	Low limit	-1000~1000	-1000	
5	High Limit	-1000~1000	1000	
6	Postive Setting	0、1	1	0: clockwise; 1: Counterclockwise

1.6 Specific operation

Interface ID:UAVCAN_PROTOCOL_PARAM_EXECUTEOPCODE_ID

Interface Object :uavcan_protocol_param_ExecuteOpcodeRequest

1.7 DNA support

Provide a 12 byte unique ID and a 64 bit US timer to support automatic address allocation.

1.8Firmware updata support

Built in Bootloader supports firmware updata. The firmware updata uses CAN standard frames, while UAVCAN uses CAN extended frames, so the two will not conflict.

Updata software: (BUSUpdate) Run on systems with Win7 or higher versions.

2. Secondary development support

Users can also implement more UAVCAN interfaces as needed. KST provides secondary development interfaces for servos, which are provided in the form of KServoIntf.lib and KServoIntf.H/KServoIntf. c files.

2.1 Development interface

```
#define KSERVO_CANDATA_MAXSIZE          8     // CAN Up to 8 bytes
#define KSERVO_CONFIGPARAM_MAXSIZE 32   // Total 32 bytes of configuration space
#define KSERVO_CANFRAME_FLAG                0xFCA0  // Interface version flag

typedef struct kservo_can_frame_stu
{
    unsigned int        ExtID;          // CAN Extend ID，Only valid for the lower 29 bits
    unsigned short Flag;                // must be KSERVO_CANFRAME_FLAG
    unsigned char  IsExt            : 1;  // 0:not support; 1:extend frame
    unsigned char  IsRemote : 1;  // 0:data frame; 1:remote frame
    unsigned char               : 2;  // reserve
    unsigned char  DLC              : 4;  // 0~8, transmission data length
    unsigned char  Unused;          // When sending, it should be set to 0
    unsigned char  Data[KSERVO_CANDATA_MAXSIZE];
} KServoCANFrameStu;

typedef enum kservo_can_baud_type          // Support baud rate list, corresponding to 0~8
{
    ks10kbps,    ks20kbps,    ks50kbps,    ks100kbps,
    ks125kbps,   ks250kbps,   ks500kbps,   ks800kbps,   ks1Mbps
} KServoCANBaudType;

typedef enum kservo_param_type        // reserve
{
    ksParam1,    // reserve
    ksParam2,    // reserve
    ksParam3,    // reserve
                 //
} KServoParamType;

// Functions that must be implemented for up communication protocols
```

```
extern void KServo_TimerINTProc(void);    // mstimer interrupt
extern void KServo_CANINTProc(void);      // received CAN extension frame interrupt
  extern void KServo_DataInit(void);
/* Here, some data can be initialized. At this point, the configuration has been loaded and other interfaces have not been initialized and cannot be called.*/

extern void KServo_RunInit(void);
/*All internal interface initialization has been completed, and some protocol initialization work can be done*/

extern void KServo_RunSch(void);
/* Cycle time slice scheduling, preferably within 1ms of occupancy time*/


// Interface for calling up communication protocols
extern unsigned short KServo_GetusTick(void);   // 16 bit US counter since chip power on

extern unsigned __int64 KServo_GetusTickEx(void);  // 64 bit US counter since chip power on

extern _Bool KServo_GetConfigParam(void *pBuf, size_t Size);
// Used for reading communication protocol parameters
//Size <= KSERVO_CONFIGPARAM_MAXSIZE

extern _Bool KServo_SetConfigParam(const void *pBuf, size_t Size);
// Used for modifying communication protocol parameters
// Size <= KSERVO_CONFIGPARAM_MAXSIZE

extern _Bool KServo_SaveConfigParam(const char *Pass);
/* Used for storing communication protocol parameters, Pass will be provided later. After modifying the parameters, it is recommended to save them together.*/


extern unsigned char KServo_GetCANAddr(void);
/* Read back the Addr set by the KServo_SetCANAddr function, which defaults to a specific value after power on */

extern _Bool KServo_SetCANAddr(unsigned char Addr);
// This function must be called in KServo_dataInit after each power on

extern KServoCANBaudType KServo_GetCANBaud(void);
/* Read back the Baud set by the KServo_SetCANBaud function, which defaults to a specific value after power on*/
```

```
extern _Bool KServo_SetCANBaud(KServoCANBaudType Baud);
// This function must be called in KServo_dataInit after each power on

extern _Bool KServo_ResetConfig(const char *Pass);
/* Reset the configuration data to the factory state (Configurarm reset), and temporarily do not reset the internal parameters of the servo*/
/* Read and write midpoint, limit, control, etc., the settings are immediately effective and must be saved using KServo_SaveConfig Param*/

extern _Bool KServo_SetZero(void);    // Set the current position as the midpoint
extern _Bool KServo_SetNeu(short Neu);    // Set the midpoint
extern _Bool KServo_SetLimit(short Low, short High);      // -1000 ~ 1000
extern _Bool KServo_SetControl(unsigned short Ctrl);//

extern short KServo_GetNeu(void);      // -1000 ~ 1000
extern _Bool KServo_GetLimit(short *pLow, short *pHigh);
extern unsigned short KServo_GetControl(void);

extern void    KServo_Reset(const char *Pass);    // Reset Servo
extern _Bool KServo_SendPWM(short PWM);    // Set Range -1000~1000 => -100.0°  ~100.0°
extern short KServo_GetPWM(void);
/*Read back the PWM set by the KServo_SendPWM function, which defaults to a specific value after power on*/

extern short KServo_GetCurrPos(void);
/* read current position(-1000~1000 => -100.0°  ~100.0°  ) */

extern short KServo_GetCurrTemp(void);    // read servo temperature (-40~120)
extern short KServo_GetCurrCurrent(void);  // read servo currents, unit:10mA
extern short KServo_GetCurrVolt(void);              // Not support

extern unsigned int KServo_GetParam(KServoParamType Type);      // Not support
extern _Bool KServo_SetParam(KServoParamType Type, unsigned int Val);     // Not support
extern unsigned short KServo_GetSoftVer(void); // read software version

extern _Bool KServo_GetCANFrame(KServoCANFrameStu *pCANFrame);
// read received CAN extend frame,Can be called in CANINTProc or RunSchextern _Bool KServo_SendCANFrame(const KServoCANFrameStu *pCANFrame);
// Sending CAN extension frames, continuous sending of multiple frames will fail
```