X36−X35 NT Driver

Documentation for developers

CHAPTER I

THe following document is thought for people with knowlegde about C programming, even thought it could be interesting for those who want to know how Saitek X36 and X35 joysticks work.

The joysticks

The X36 and X35 joysticks can work combinated or seperated and there are two diferent versions ***F*** and ***T*** .
The X36 is a "normal" joystick with 2 axis and 13 buttons while the X35 has 2 axis (throttle and rudder), 14 buttons and two rotaries. ***F*** versions can be programmed and ***T*** versions can not, exception of ***X35−T*** that can be programmed combinated with the ***X36−F*** *(from here to the end we always refer to this combo).*

These joysticks have a double function :

− By one side they have an analog function that makes them work as a normal 4 axis joystick.
This function is made through the gameport and is completly indepent from the digital function, so to work you don´t need to have the keyboard conector plugged in it´s corresponding port.

− By the other side they have a digital function. With it buttons and axis can work as they were keyboard keys.
This function is made through the keyboard port and it depends on the game port.

Joystick initiatation

To use digital functions the keyboard port is used for input and for output. Input identifies the button pressed and output is user to program the joystick.

There are two way to initialize the joystick, one with the predefined configuration and one with a custom configuration. *With no initiation joysticks have a 4 axis, 4 button configuration.*

Predefined profile is loaded by pressing the ***Launch*** button after the operationg system is loaded and ready. With this profile the joysticks have a 4 xis, 6 buttons with 2 POV configuration.

To initialize joysticks in a custom way it´s necesary to send through the keyboard port exactly 14 bytes. For each byte sended a confimation byte is sended by the joystick except in last one where 9 bytes are received. Byte description is as following (decimal notation):

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14
110	16	11	Btn1	Btn2	Btn3	Btn4	Btn5	Btn6	POV1	POV2	T&R	8	16

- **Btn? :** Port buttons, indicating what joysticks buttons* are asigned as the 1−6 buttons in the analog function.
- **POV? :** With values from 0 to 4, they indicate what of the 4 hats is asigned as POV 1 and as POV 2 in the analog function (0 = not asigned).
- **T&R :** Throttle and Rudder configuration :
    - 0 = throttle and rudder in analog function.
    - 1 = throttle analog, rudder digital.
    - 2 = throttle digital, rudder analog.
    - 3 = throttle and rudder in digital function.

**Joystick button codes to use in Btn? byte appears in the table to the end of chapter II (0 = not asigned).*

Notes :

- Any configuration remains active until joysticks are manually reseted or the system is rebooted...or another profile is loaded.
- Each time the system is rebooted joysticks are reseted so any configuration is lost.
- By default, after a reset, the joysticks have the 4 axis, 4 button configuration.
- Once initiated, the predefined profile can not be loaded until the joysticks are reseted.

Digital function

After a custom initiation all buttons, axis and rotaries work. However, ***they can not be programmed***, this is, you can´t send any data to the joystick to tell it, "pressing button A send the Return key".

Nevertheless, once initiated each time you press/raise a button or you move an axis (working as digital) or rotary a 4−6 byte code is received through keyboard port, concretely :

Button codes

Byte 1	Byte 2	Byte 3	Byte 4
111	16	1	ID

Axis and rotaries codes

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
111	16	3	ID (33, 34, 35 ó 36)	Subposition (48−63)	Position (48−63)

Knowing this, joysticks can be programmed with an external program who replaces these codes by the keys we want. But, there is another problem...

As the codes are received in the keyboard port they are interpreted as keys, this is, 4 or 6 letters are written each time a button is pressed, so it´s necesary to delete the codes from the keyboard buffer before they were written and bypassing DirectX.

X36−X35 NT Driver

Documentation for developers

CHAPTER II

X36−X35 NT Driver

To initiate and "translate" joystick codes inside a Windows NT system the better way is to add a keyboard filter driver (**kbfiltr.sys**).
By this way we can bypass the NT kernel keyboard output data block and we can modify the keyboard buffer dato before it is procesed.

Respect to the file code, it is programmed to leave the default NT system interface intact, this is, once installed there is no difference from the default keyboard working model.
This means that, for example, it is not possible to send data through the keyboard port.

Driver control

The mechanism used to control initiation and code translation in the Driver is the Windows device control API. Concretely, the IO control call IOCTL_KEYBOARD_QUERY_INDICATOR_TRANSLATION definced by the API as :

```
#define
   IOCTL_KEYBOARD_QUERY_INDICATOR_TRANSLATION
     CTL_CODE(FILE_DEVICE_KEYBOARD, 0x0020, METHOD_BUFFERED, FILE_ANY_ACCESS)
```

To call this function it is necesary to used the device contro API function...

```
DeviceIOControl( device, call, &data, size, NULL, 0, &response, NULL);
```

where `device` is the keyboard port device address. It is a string defined in the windows registry in :

```
"HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\DeviceClasses\
\{884b96c3−56ef−11d1−bc8c−00a0c91405dd}\\##?#ACPI#PNP...\\#\\SymbolicLink"
```

According to the Driver programming depending on the data size of the call a different functions is interpreted. So one size is for a kind of configuration and other different is for another kind.
In the following pages you can see these functions and their asociated size.

Call of size = 11 bytes (*Initiation*)

This call is used to initiate the joysticks. *&data* is the 11 bytes data buffer adress to send to the driver.

Buffer

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11
Btn1	Btn2	Btn3	Btn4	Btn5	Btn6	POV1	POV2	T&R	Modes	Pinkie

- ***Modes*** is a boolean indicating when the mode switch is used as mode selector (*true (1)*) or as button (*false (0)*).
  When it is initiated to false, Mode 2 configuration is used.
- ***Pinkie*** is a boolean indicatinf if the pinkie function is used or does not.

Call of size = 8 bytes (*Combinable keys*)

This call is used to inform the driver what are the keyboard special key scancodes (CTRL, ALT, SHF and WIN).
&datos points to a 8 byte buffer with the scancodes, 4 for left keys and 4 for the right ones (the order is indifferent).

Call of size = 37 bytes (*Autorepeats*)

This is used to inform the driver what are the buttons with auto−repeat. The buffer has 37 bytes and it is necesary to make 6 calls (one per mode) to configure all the buttons.

- The first byte indicates the mode to configure (see next table).
- The 36 bytes left are booleans corresponding to the 36 joystick buttons (see table to the end of the chapter). There are **true** if the button will have auto−repeat.

Mode	0	1	2	3	4	5
Configuration	Mode 1	Mode 1 & Pinkie	Mode 2	Mode 2 & Pinkie	Mode 3	Mode 3 & Pinkie

Call of size = 24 bytes (*Keys*)

This call is used to send to the driver the kays asigned to each button, axis and rotary. As in the auto−repeat call it is necesary to make several call of this type to complete the configuration.

In this case the buffer has the following structure (24 bytes).

```
typedef struct _SAITEK_PACKET
{
UCHAR Mode;
UCHAR Position;
UINT64 Data;
UCHAR DataExt;
} SAITEK_PACKET, *PSAITEK_PACKET;
```

- ***Mode*** is the same as in the previous call.
- ***Position*** identifies the button, axis or rotary (see table to the end of the chapter).
- ***Data*** inidicates the key scancodes that will be sended pressing the button, axis or rotary. As it is an `UINT64` it can have 8 scancodes.
- ***DataExt*** indicates what of the 8 keys are extended keys. Each byte corresponds to one byte of *Data*, if the bit is 1 the key is extended and if it is 0 it is not.

Call of size = 25 bytes (*Rotaries*)

This call is user to decribe rotaries and axis working mode (when working is digital function).

In this case data buffer has 24 bytes defined as a matrix `UCHAR axis[4][6];`. The first matrix index indicates the rotary/axis and the second one the mode (see table below). The byte will be:

- ***0*** if the axis/rotary must be inactive.
- ***32*** if the axis/rotary will be incremental.
- ***N*** if the axis/rotary must have N position (*For axis N can be 6 as much and for rotaries 10 as much*).

To distinguish this call from the previous call data buffer must have an aditional byte. This byte can have any value as it is only used to distinguish the calls.

		0	1	2	3	4	5
		Mode 1	Mode 1 & Pinkie	Mode 2	Mode 2 & Pinkie	Mode 3	Mode 3 & Pinkie
0	Throttle	UCHAR	UCHAR	UCHAR	UCHAR	UCHAR	UCHAR
1	Rudder	UCHAR	UCHAR	UCHAR	UCHAR	UCHAR	UCHAR
2	Rotary 1	UCHAR	UCHAR	UCHAR	UCHAR	UCHAR	UCHAR
3	Rotary 2	UCHAR	UCHAR	UCHAR	UCHAR	UCHAR	UCHAR

Button positions and IDs

The following table indicates what button, axis or rotary is asigned to each position for the different calls. Also it´s indicated what is the button ID to use it on joystick initialitation.

Button, axis, rotary	Pos.	ID
Trigger	0	13
Launch	1	4
Button A	2	2
Button B	3	3
Button C	4	1
Pinkie	5	14
Hat 1 up	6	5
Hat 1 down	7	7
Hat 1 left	8	8
Hat 1 right	9	6
Hat 1 up−left	10	
Hat 1 up−right	11	
Hat 1 down−left	12	
Hat 1 down−right	13	
Hat 2 up	14	9
Hat 2 down	15	11
Hat 2 left	16	12
Hat 2 right	17	10
Hat 2 up−left	18	
Hat 2 up−right	19	
Hat 2 down−left	20	
Hat 2 down−right	21	
Button D	22	15
Mouse button	23	16
Mode left	24	
Mode right	25	
Aux left	26	
Aux right	27	
Mouse hat up	28	21
Mouse hat down	29	23
Mouse hat left	30	24
Mouse hat right	31	22
Hat 3 up	32	17
Hat 3 down	33	19
Hat 3 left	34	20
Hat 3 right	35	18

Button, axis, rotary	Pos.	ID
[1] Pulsations at button raise (button order is the same as is the table to the left)	36−71	
[2] Throttle	72−77	
[2] Rudder	78−83	
[2] Rotary 1	84−93	
[2] Rotary 2	94−103	

[1] *Example : position 36 is for "raised trigger", 37 "raised Launch"...*

[2] *For axis and rotaries, when they are in incremental mode the first position is for increment and the second for decrement. I.e., for throttle position 72 is increment and 73 decrement.*

X36−X35 NT Driver

Documentation for developers

CHAPTER III

.XMP file format

The actual format of the .XMP files used by the X36Map is this :

- ***WORD*** (2 bytes) − Number of commands in file.
- ***Commands***
    - ***STRING*** (32 bytes) − Name.
    - ***UINT64*** (8 bytes) − Scancodes (1 byte per scancode).
    - ***UCHAR*** (1 byte) − Extended keys (1 bit per scancode).
- ***Initiation data***
    - ***6 UCHAR*** (6 bytes) − Port buttons.
    - ***UCHAR*** (1 byte) − POV 1.
    - ***UCHAR*** (1 byte) − POV 2.
    - ***UCHAR*** (1 byte) − Throttle and rudder configuration.
    - ***UCHAR*** (1 byte) − Mode control and pinkie.
        - ◊ *bit 8* : 1 = modes On; 0 = modes Off
        - ◊ *bit 7* : 1 = pinkie mode On; 0 = pinkie mode Off
        - ◊ *bit 1−6* : Not used
- ***6 UCHAR*** (6 bytes) − Rotation type for throttle.
- ***6 UCHAR*** (6 bytes) − Rotation type for rudder.
- ***6 UCHAR*** (6 bytes) − Rotation type for rotary 1.
- ***6 UCHAR*** (6 bytes) − Rotation type for rotary 2.
- ***6*104 WORD*** (1248 bytes) − Indexes. Each WORD indicates the command number to use for each button, axis or rotary.
- ***6*36 BOOLEAN*** (216 bytes) − Auto repeats. The format is the same as in the different IOCTL calls.

*This format is coherent to the information written in the previous chapters. I.e. 6*36 BOOLEAN are 6 blocks (one per mode) of 36 bytes corresponding to the 36 positions that appear in the table to the end of chapter II.*

X36−X35 NT Driver

Documentation for developers

APPENDIX A

Codes sended by the joysticks

Buttons	Press					Raise			
	Byte 1	Byte 2	Byte 3	Byte 4		Byte 1	Byte 2	Byte 3	Byte 4
Trigger	111	16	1	13		111	16	1	77
Launch	111	16	1	4		111	16	1	68
Button A	111	16	1	2		111	16	1	66
Button B	111	16	1	3		111	16	1	67
Button C	111	16	1	1		111	16	1	65
Pinkie	111	16	1	14		111	16	1	78
Hat 1 up	111	16	1	5		111	16	1	69
Hat 1 down	111	16	1	7		111	16	1	71
Hat 1 left	111	16	1	8		111	16	1	72
Hat 1 right	111	16	1	6		111	16	1	70
Hat 2 up	111	16	1	9		111	16	1	73
Hat 2 down	111	16	1	11		111	16	1	75
Hat 2 left	111	16	1	12		111	16	1	76
Hat 2 right	111	16	1	10		111	16	1	74
Button D	111	16	1	15		111	16	1	79
Mouse button	111	16	1	16		111	16	1	80
Mode left	111	16	1	32		111	16	1	96
Mode right	111	16	1	31		111	16	1	95
Auxiliar left	111	16	1	30		111	16	1	94
Auxiliar right	111	16	1	29		111	16	1	93
Mouse hat up	111	16	1	21		111	16	1	85
Mouse hat down	111	16	1	23		111	16	1	87
Mouse hat left	111	16	1	24		111	16	1	88
Mouse hat right	111	16	1	22		111	16	1	86
Hat 3 up	111	16	1	18		111	16	1	82
Hat 3 down	111	16	1	20		111	16	1	84
Hat 3 left	111	16	1	17		111	16	1	81
Hat 3 right	111	16	1	19		111	16	1	83