USB2TTL8 User Manual

1000 Hz USB TTL Interface

Document Version: 1.0.0
Last Updated: March 1st, 2019

For Firmware Version 1.0.5

Please read this manual fully before using the USB2TTL8 device. Improper connections to the USB2TTL8 DB25 port can cause the USB2TTL8 device to reset, or even permanently damage the USB2TTL8 microcontroller. Damage caused by over or reverse voltage conditions resulting from improper wiring to the DB25 port is *not* covered by the USB2TTL8 warranty.

Contents

Overview

The USB2TTL8 is a 1000 Hz USB Serial to 8 bit TTL device that provides a flexible TTL option for computers without a parallel port. Data Pins D0-7 are used to either read or write 8 bits in parallel (DB25 pins 2 - 9). USB Serial commands are used to set device options and write data to the DB25 Data Pins. When in read mode, changes in the Data Pins are communicated to the computer as either USB Serial or as standard keyboard events.

- 1000 Hz USB Serial Interface.
- Use as a TTL Input *or* TTL Output Device.
- Provides a Flexible TTL Option for Computers Without a Parallel Port.
- Write / Set TTL Lines using Simple Serial Commands.
- Read / Detect TTL Changes as Serial Messages or Standard Keyboard Events.
- Control TTL Write Pulse Duration with usec Resolution.
- Works with Windows 10, Linux, and MacOS.
- Supports Bi-Directional Parallel Data Port Interface.
- Supported TTL Logic Levels: 0 - 3.3/5V Input; 0 - 3.3V Output.

Setup

For the USB2TTL8 to work properly, it must be:

1. Connected to a USB 2.0 port of the computer that will use the USB2TTL8 Serial Port.
2. Interfaced to a 0 - 3.3/5 V TTL system using the Female DB25 Port (specifically pins 2 - 9 for data).

USB 2.0 Cable

Connect the USB2TTL8 USB cable to the computer that will be reading / writing data via the USB2TTL8 USB Serial port. The USB2TTL8 device is also powered from this USB connection.

Female DB25 Port

The DB25 Port of the USB2TTL8 device is used to read or write up to 8 TTL signals.

Connect the DB25 port to a computer Parallel Port using a *Male to Male DB25 straight through cable* and data can be written / read from the USB2TTL8 by reading / writing to the Parallel Port DATA Register (pins 2 - 9).

Important: The USB2TTL8 will not work if connected to a parallel port using a standard printer parallel port cable; a straight through patch cable is needed.

A Male DB25 breakout board / box can be used for general purpose interfacing to the USB2TTL8 DB25 port.

Device Software

Windows 10, OS X, and Linux all automatically detect the USB2TTL8 as both a 1000 Hz USB Serial and Keyboard device; no device drivers are required.

For Windows 7, a USB Serial driver must be installed before the USB2TTL8 USB Serial interface can be used. Please install the USB Serial driver supplied by PJRC, makers of the microcontroller used in the USB2TTL8. Only install this driver if using Windows 7 / XP.

For Linux, a udev rule must be changed to allow read/write access to the USB2TTL8 USB serial port. https://www.pjrc.com/teensy/49-teensy.rules

LabHackers Device Manager

LabHackers Device Manager is used to configure the USB2TTL8 device. If using Windows or macOS, download a binary version of the software from:

http://www.labhackers.com/downloads.html

LabHackers Device Manager is written in Python and can be run from source on Linux. Please contact LabHackers for further information.

Detecting Serial Port

When the USB2TTL8 is connected to to a computer (the host), it is registered as both a USB Keyboard and USB Serial Device. The serial port address assigned to the USB2TTL8 is handled by the computer operating system. Unless the experiment software being used automatically detects connected LabHackers devices, the Serial Port address assigned to the device must be determined and manually specified in the experiment software. LabHackers provides two methods to read the USB Serial Port address assigned to an USB2TTL8 device:

1. **LabHackers Device Manager**
   Launch LabHackers Device Manager with a USB2TTL8 device connected; the Serial Port for the device is listed on the General tab

2. **Python Script**
   To determine the device serial port using Python, run the examples/python/detect_usb2ttl8.py example. If a USB2TTL8 device is connected to the computer the script should print something like:

   USB2TTL8 Device:

Name:	USB2TTL8
Serial Port:	COM276
Serial Number:	LH0002E215

DB25 Port Pin Out

The USB2TTL8 uses the following DB25 pins.

- N/A indicates that the pin can not be used as an input or output.
- GND indicates ground pins.
- Voltage ranges indicate the USB2TTL8 tolerance for the given pin in the given mode.

Pin	Name	Input	Output
1	STROBE	0 - 5.0 V	N/A
2 - 9	DATA Pins	0 - 5.0 V	0 - 3.3 V
10	RES	N/A	0 - 3.3 V
11	MODE	N/A	0 - 3.3 V
12 - 25	Ground Pins	GND	

The USB2TTL8 inputs that support 5V also work with 3.3 V inputs.

Important: Do not apply > 5V to any pin of the USB2TTL8 DB25 port or permanent damage could occur to the device.

STROBE Pin

DB25 Pin: 1
INPUT
Optionally used for READ mode hardware handshaking.
See the READ Mode section for further details.

DATA Pins

DB25 Pin: 2 - 9
INPUT or OUTPUT
Used to read or write DATA byte depending on the current USB2TTL8 data mode.

MODE Pin

DB25 Pin: 11
OUTPUT Only
The device connected to the USB2TTL8 DB25 port can use this pin to read the current data mode of the USB2TTL8.

High = WRITE Mode
LOW = READ Mode.

RES Pin

DB25 Pin: 10
This pin is currently not used by the USB2TTL8 firmware.

Status LED

The USB2TTL8 includes a status LED mounted next to the USB cable connection on the device. The color of the LED indicates if the device has an open USB serial connection to the host and whether the device is in data READ or WRITE mode.

LED Color	Meaning
Red	IDLE: No open USB Serial connection with the host computer.
Green	WRITE: The USB2TTL8 is configured to write to the data port (pins 2 - 9).
Blue	READ: The USB2TTL8 is configured to read from the data port (pins 2 - 9).

Device Configuration

USB2TTL8 settings can be updated by using the LabHackers Device Manager application or with the USB2TTL8 USB Serial interface SET command.

Label	Description	SET Cmd	Options	Default
Name	Name of device instance.	NAME	8 char ascii string	USB2TTL8
Data Mode	Set whether the 8 data lines are in Read (Input) or Write (Output) mode.	DATA_MODE	READ WRITE	WRITE
Read Event Type	When in Read Mode, set what type of event is created when the data byte value changes.	READ_EVT_TYPE	SERIAL BITS2KEYS BYTE2KEY	Serial
Enable Offline Keyboard Events	When in Read mode and generating keyboard events, controls whether the USB2TTL8 should create keyboard events regardless of USB Serial connection state. 0 = Only generate events when serial connection is open (default). 1 = Always generate keyboard events.	OFFLINE_KB_EVENTS	0, 1	0
BITS2KEYS Mappings	Set the Data bits to Keyboard key mapping used when generating BITS2KEYS read events.	BIT2KEY	See Keyboard Mapping Table	{KEY_0, KEY_1, KEY_2, KEY_3, KEY_4, KEY_5, NONE, NONE};
BYTE2KEY Mappings	Set the Data byte to Keyboard key mappings used when generating BYTE2KEY read events.	BYTE2KEY	See Keyboard Mapping Table	No Mappings
Key Press Duration	Maximum key press duration used while USB2TTL8 is operating in BITS2KEYS or BYTE2KEY event mode. 0 =	KEY_DURATION	0 - 60000 msec	100

	No maximum duration.			
Flow Control	Set the hardware flow control the USB2TTL8 will use.	FLOW_CONTROL	OFF STROBE_READ	OFF
Read Usec	Usec debounce period on data input changes.	READ_CHANGE_USEC	0 to 512 usec	5

Reading or Writing Data

The USB2TTL8 operates the 8 data pins in either READ or WRITE Mode.

- In READ mode, the USB2TTL8 device configures the data pins as 8 bit digital inputs.
- In WRITE mode, the USB2TTL8 data pins are configured as an 8 bit digital output.

Important: For correct operation, the hardware connected to the USB2TTL8 DB25 port must be configured to either write to or read from the data port (pins 2 - 9 of the DB25).

WRITE Mode

Status LED Color: Green
MODE Pin: LOW

When the USB2TTL8 is in WRITE mode, the data byte or an individual data bit value can be set using the WRITE and WRITEP commands. Please see each USB Serial commands section for more details.

Note: When a WRITE command is sent to the USB2TTL8 all 8 data bits are updated in parallel. However, the data line signals can become desynchronized from one another during transmission, resulting in the possibility of occasional (< 0.25%), short (< 5 usec), transient values being read. To handle these transients, it is suggested that the program or device reading the TTL output of the USB2TTL8 perform a double read whenever a change is detected. The value from the second read should be stable and can therefore be used.

Reading via Parallel Port

If you are interfacing the USB2TTL8 with a PC Parallel Port, ensure that the parallel port is set to be able to read from the data register. Set bit 5 of the parallel ports Control register to 1 or HIGH.

Example Python script to **enable reading on parallel port** data register:

```
v = winioport.inp(parallelPortAddress+2)
v = v | 1<<5
winioport.out(parallelPortAddress+2, v)
```

READ Mode

Status LED Color: Blue
MODE Pin: HIGH

When the USB2TTL8 is in READ mode, data is read from the USB2TTL8 data port. When the data port value changes, USB Serial or Keyboard events are generated by the USB2TTL8 device.

The current value of the USB2TTL8 data port can also be read using the READ serial command.

Writing via Parallel Port

If you are interfacing the USB2TTL8 with a PC Parallel Port, ensure that the parallel port is set to be able to write to the data register by setting bit 5 of the parallel ports Control register to 0 or LOW.
Example Python script to **enable writing on parallel port** data register:

```
v = winioport.inp(parallelPortAddress+2)
v = v & ~(1<<5)
winioport.out(parallelPortAddress+2, v)
```

Hardware Flow Control

By default, no hardware flow control is used between the USB2TTL8 data lines and device connected to them. This means, for example, that if the device writing to the Data byte does not update all 8 data bits simultaneously, it is very likely that the USB2TTL8 will read several Data byte value changes in quick succession.

When the USB2TTL8 is in Read mode, STROBE_READ flow control mode can be used. When STROBE_READ is enabled, the USB2TTL8 device only reads the data byte on the falling edge of a STROBE line (pin1) change. This allows the device connected to the DB25 port to first write to the data pins, and then toggle the STROBE line HIGH-LOW-HIGH to trigger the USB2TTL8 to read the data byte.

When using the USB2TTL8 STROBE_READ feature, the device writing to the Data lines must:

1. On initialization, set the STROBE line (Pin 1) to HIGH.
2. Write to the Data lines as normal.
3. To instruct the USB2TTL8 to read the Data lines, set the STROBE line to LOW.
4. Reset the STROBE line to HIGH.
5. Repeat 2 - 4 as required.

READ_CHANGE_USEC Command

If hardware flow control is off, the READ_CHANGE_USEC command can be used to set the number of usec the USB2TTL8 waits after detecting a change in the data port value before reading the data port again and returning this last read value. This gives the data pins time to stabilize and can catch desynchronization that may have occurred to the changed data signal lines during transmission .

Default value is 5 usec.

Read Event Types

The USB2TTL8 can generate USB Serial messages or Keyboard events when a change in the data byte value is detected. Use the LabHackers Device Manager to the event type to use, or use the SET READ_EVT_TYPE serial command.

Serial Events

Command: SET READ_EVT_TYPE

When the USB2TTL8 device is in READ mode and READ_EVT_TYPE is set to Serial, any change to the DB25 data port state will generate a serial message of the form:

value\n

where

value	Current value of data byte. 0 - 255.

Keyboard Events

The USB2TTL8 supports two modes of TTL to Keyboard event generation: BITS2KEYS and BYTE2KEY.

In BITS2KEYS mode, each bit of the data byte is mapped to a key. When the bit is HIGH, the associated key is pressed. When the bit is LOW, the associated key is released.The maximum duration of key presses in this mode is set using the KEY_DURATION device setting.

In BYTE2KEY mode, each data byte value between 0 and 255 can be assigned to a key. When the data changes value a key press is created if it has an associated keyboard key mapping. The maximum duration of key presses in this mode is set using the KEY_DURATION device setting.

BITS2KEYS

When the USB2TTL8 is in READ mode and is set to create BITS2KEYS Keyboard events, each bit of the data byte is mapped to a keyboard key or modifier. When a data bit goes HIGH, the associated key is pressed for 'Key Press Duration' msec, or until the data bit goes LOW, whichever occurs first. Therefore 0 to 8 keys (6 standard, 2 modifiers) can be pressed at any time based on the data byte value. For example, using the default key mappings for the BITS2KEYS mode:

Data Byte Value	Pressed Keys
0	None
1	'0'
2	'1'
3	'0', '1'
4	'2'
8	'3'
16	'4'
32	'5'
64	None
128	None
255	'0', '1', '2', '3', '4', '5'

BYTE2KEY

The USB2TTL8 stores a 256 Data byte value to Keyboard key mapping table. A value of 0 indicates no keyboard mapping for the given data byte value. This is also the factory default for all 256 possible values.

When the USB2TTL8 is in READ mode and is set to create BYTE2KEY Keyboard events, a change in the Data byte value causes the associated keyboard key (if any) to be pressed for

‘Key Press Duration’ msec, or until the data byte value changes, whichever comes first. At most one key is pressed at a time when in BYTE2KEY mode.

USB Serial Interface

When a connection is made to the USB Serial port of a USB2TTL8 device, the device accepts the connection and waits for commands. Each command is an ASCII line ending in a newline character.

When a command is received by a USB2TTL8 device, it is processed and a response is sent back to the computer. The response contents, if any, is a variable length string ending in a newline character.

USB Serial Port Settings

When connecting to the USB2TTL8 USB Serial port from within your own software, the device's Serial port name / address must be known. A Python example of how to detect the USB2TTL8 devices connected to a computer, see the USB2TTL8 detect_usb2ttl8.py Python script.

The baud rate set by your software for the Serial Port connection doesn't really matter as it is ignored. The USB2TTL8 USB Serial connection can send and receive up to 64 bytes / msec.

The timeout setting used for the USB2TTL8 Serial connection in your software can be an important setting to consider. The *appropriate timeout* value to use depends on the design of your program and the software being used. A timeout of 0.01 (10 msec) to 0.1 (100 msec) us often fine. For programs requiring more of a non blocking read operation inside the main experiment program, set the timeout to 0 or 0.001 (1 msec) at most.

Commands

PING

Ask the USB2TTL8 to identify itself.

Format

PING\n

Reply

{"PING":"USB2TTL8","ID":"LHxxxxxxxx"}\n

GET CONFIG

Read the current USB2TTL8 device configuration, returned as a json encoded dictionary.

Format

GET CONFIG\n

Reply

{"version": "0.7.1", "model_name": "USB2TTL8", "product_serial": "LH000xxxxx", "loop_freq": 123.00, "name": "USB2TTL8", "data_mode": 1, "handshaking_mode": 0, "read_event_type": 1, "read_dx_interval": 5, "bit2key": [61479,61470,61471,61472,61473,61474,57348,57345],"byte2key": {},"keypress_duration": 0}\n

SET

Set a USB2TTL8 device setting. SET accepts one of the following key arguments.

Note: Changes made to USB2TTL8 settings are not saved to the devices' eeprom until a SAVE_CONFIG command is received by the USB2TTL8.

NAME

Set the USB2TTL8 8 character name. Default is "USB2TTL8".

Format

SET NAME myusbttl\n

Reply

{"reply": "SET", "param": "NAME", "arg": "myusbttl", "result": "OK"}\n

DATA_MODE

Set the USB2TTL8 to Data Port to Read or Write mode. Default is READ.

Format

SET DATA_MODE [READ | WRITE]\n

Reply

None

FLOW_CONTROL

Set the hardware handshaking mode used by the device. Default is OFF.

Options:

OFF	No hardware flow control is used.
STROBE_READ	USB2TTL8 device only reads data port on falling edge of STROBE signal (DB25 pin 1)

Format

SET FLOW_CONTROL [OFF | STROBE_READ]\n

Reply

{"reply": "SET", "param": "FLOW_CONTROL", "arg": "OFF", "result": "OK"}\n

READ_CHANGE_USEC

If hardware flow control is off, set the number of usec the USB2TTL8 should wait after detecting a change in the data port value before reading the data port again. This gives the data pins time to stabilize and catch any desynchronization that may have occurred to the 8 data signals during transmission. Supported values are 0 to 512 usec. Default is 5 usec.

Format

SET READ_CHANGE_USEC 5\n

Reply

{"reply": "SET", "param": "READ_CHANGE_USEC", "arg": "5", "result": "OK"}\n

BIT2KEY

Set one of the data bit key mappings for use when BITS2KEYS Keyboard event mode is enabled. Default BIT2KEY Mappings:

Data Bit	Key
0	‘0’
1	‘1’
2	‘2’
3	‘3’
4	‘4’
5	‘5’
6	ALT
7	CTRL

Format

SET BIT2KEY [bit] [keyname]

Where:

bit = 0 - 7

Keyname = (bit 0-5): Standard Key Name

(bit 6-7): Modifier Key Name

Key Names can be found in the USB2TTL8 Keyboard Mapping Table at the end of this document.

Example

SET BIT2KEY 1 2\n

Reply

{"reply": "SET", "param": "BIT2KEY", "arg": "1", "result": "OK"}\n

BYTE2KEY

Sets a Data byte value to Keyboard Key mapping to be used when the USB2TTL8 is in READ BYTE2KEY Mode. For Data byte values that should not have an associated keyboard key, use a keyname of NONE.

Format

SET BYTE2KEY [byte] [keyname]

Where:

byte = 0 - 255
keyname = Standard Key Name

Key Names can be found in the USB2TTL8 Keyboard Mapping Table at the end of this document.

Example

SET BYTE2KEY 128 A\n

Reply

{"reply": "SET", "param": "BYTE2KEY", "arg": "128", "result": "OK"}

OFFLINE_KB_EVENTS

When the USB2TTL8 is in READ mode and is generating BITS2KEYS or BYTE2KEY Keyboard events, the OFFLINE_KB_EVENTS setting is used to control if Keyboard events are only generated when the USB2TTL8 has an active Serial connection (0, default), or if keyboard events should be generated regardless of the USB Serial connection state (1).

Format

SET OFFLINE_KB_EVENTS [0 | 1]\n

Example

SET OFFLINE_KB_EVENTS 0\n

Reply

{"reply": "SET", "param": "OFFLINE_KB_EVENTS", "arg": "0", "result": "OK"}\n

KEY_DURATION

Sets the maximum keypress duration (in msec) for keyboard events generated while the USB2TTL8 is in READ BYTE2KEY mode. Use 0 if the keyboard press event should only release after the data input changes, otherwise keyboard press events will last at most KEY_DURATION msec before the USB2TTL8 forces a keyboard release event to occur.

Format

SET KEY_DURATION 50\n

Reply

{"reply": "SET", "param": "KEY_DURATION", "arg": "50", "result": "OK"}\n

READ

Read the current state of the USB2TTL8 data port (0-255).

Format

READ\n

Reply

0\n
...
255\n

WRITE

Set the USB2TTL8 device data port value. Optionally supports arguments for usec toggle duration and simple pattern generation.

The WRITE command only has effect if the USB2TTL8 device has first been set to Write mode by issuing the SET DATA_MODE WRITE command.

Format

WRITE value [duration] [next_value] [repeat]\n

value	set data port byte to 0 - 255, i.e. atoi(value)
duration	usec duration to hold value for
next_value	0 - 255: set data port byte to atoi(next_value) after duration usec. -1: set data port byte to previous state after duration usec.
repeat	If specified, alternate between value and next_value at a duration usec interval until serial connection is closed.

If duration is 0, data port is set to value until the next WRITE command; otherwise data port is set to value for duration usec.

If next_value == -1, data port is set to its original state after duration usec; otherwise data port is set to next_value after duration usec. duration must be > 0 or next_value is ignored.

If repeat is specified (any value), data port output will automatically alternate between value and next_value at a duration usec interval until the next WRITE command is received or the serial connection is closed.

Reply

None.

WRITEP

Set a given bit of data port.

Format

WRITE dbit bvalue\n

dbit	data port bit 0 - 7
bvalue	1=HIGH, 0=LOW

Reply

None.

LOAD_DEFAULT_CONFIG

Reset USB2TTL8 settings to the factory default values.

Format

LOAD_DEFAULT_CONFIG\n

Reply

{"reply": "LOAD_DEFAULT_CONFIG", "param": "", "arg": "", "result": "OK"}\n

SAVE_CONFIG
Save USB2TTL8 settings to the device memory. When the USB2TTL8 device is connected to a computer it loads the last saved settings as default.

Format

SAVE_CONFIG\n

Reply

{"reply": "SAVE_CONFIG", "param": "", "arg": "", "result": "OK"}\n

RESET_CONFIG

Reset USB2TTL8 settings to the last saved state.

Format

RESET_CONFIG\n

Reply

{"reply": "RESET_CONFIG", "param": "", "arg": "", "result": "OK"}\n

USB2TTL8 Keyboard Mapping Table

NO KEY MAPPING	0	‘NONE’

Modifier Keys

MODIFIERKEY_CTRL	0x01 | 0xE000	‘CTRL’
MODIFIERKEY_SHIFT	0x02 | 0xE000	‘SHIFT’
MODIFIERKEY_ALT	0x04 | 0xE000	‘ALT’
MODIFIERKEY_GUI	0x08 | 0xE000	‘GUI’
MODIFIERKEY_RIGHT_CTRL	0x10 | 0xE000	‘RIGHT_CTRL’
MODIFIERKEY_RIGHT_SHIFT	0x20 | 0xE000	‘RIGHT_SHIFT’
MODIFIERKEY_RIGHT_ALT	0x40 | 0xE000	‘RIGHT_ALT’
MODIFIERKEY_RIGHT_GUI	0x80 | 0xE000	‘RIGHT_GUI’

Standard Keys

KEY_A	4 | 0xF000	‘a’
KEY_B	5 | 0xF000	‘b’
KEY_C	6 | 0xF000	‘c’
KEY_D	7 | 0xF000	‘d’
KEY_E	8 | 0xF000	‘r’

KEY_F	9 | 0xF000	‘f’
KEY_G	10 | 0xF000	‘g’
KEY_H	11 | 0xF000	‘h’
KEY_I	12 | 0xF000	‘i’
KEY_J	13 | 0xF000	‘j’
KEY_K	14 | 0xF000	‘k’
KEY_L	15 | 0xF000	‘l’
KEY_M	16 | 0xF000	‘m’
KEY_N	17 | 0xF000	‘n’
KEY_O	18 | 0xF000	‘o’
KEY_P	19 | 0xF000	‘p’
KEY_Q	20 | 0xF000	‘q’
KEY_R	21 | 0xF000	‘r’
KEY_S	22 | 0xF000	‘s’
KEY_T	23 | 0xF000	‘t’
KEY_U	24 | 0xF000	‘u’
KEY_V	25 | 0xF000	‘ v ’
KEY_W	26 | 0xF000	‘w’
KEY_X	27 | 0xF000	‘x’
KEY_Y	28 | 0xF000	‘y’
KEY_Z	29 | 0xF000	‘z’
KEY_1	30 | 0xF000	‘1’
KEY_2	31 | 0xF000	‘2’
KEY_3	32 | 0xF000	‘3’

KEY_4	33 | 0xF000	‘4’
KEY_5	34 | 0xF000	‘5’
KEY_6	35 | 0xF000	‘6’
KEY_7	36 | 0xF000	‘7’
KEY_8	37 | 0xF000	‘8’
KEY_9	38 | 0xF000	‘9’
KEY_0	39 | 0xF000	‘0’
KEY_ENTER	40 | 0xF000	‘ENTER’
KEY_ESC	41 | 0xF000	‘ESC’
KEY_BACKSPACE	42 | 0xF000	'BACKSPACE'
KEY_TAB	43 | 0xF000	‘TAB’
KEY_SPACE	44 | 0xF000	‘SPACE’
KEY_MINUS	45 | 0xF000	‘-’
KEY_EQUAL	46 | 0xF000	‘=’
KEY_BACKSLASH	49 | 0xF000	‘\’
KEY_SEMICOLON	51 | 0xF000	‘;’
KEY_QUOTE	52 | 0xF000	‘“’
KEY_TILDE	53 | 0xF000	‘~’
KEY_COMMA	54 | 0xF000	‘,’
KEY_PERIOD	55 | 0xF000	‘.’
KEY_SLASH	56 | 0xF000	‘/’
KEY_F1	58 | 0xF000	‘F1’
KEY_F2	59 | 0xF000	‘F2’
KEY_F3	60 | 0xF000	‘F3’

KEY_F4	61 | 0xF000	‘F4’
KEY_F5	62 | 0xF000	‘F5’
KEY_F6	63 | 0xF000	‘F6’
KEY_F7	64 | 0xF000	‘F7’
KEY_F8	65 | 0xF000	‘F8’
KEY_F9	66 | 0xF000	‘F9’
KEY_F10	67 | 0xF000	F10‘’
KEY_F11	68 | 0xF000	‘F11’
KEY_F12	69 | 0xF000	‘F12’
KEY_F13	104 | 0xF000	‘F13’
KEY_F14	105 | 0xF000	‘F14’
KEY_F15	106 | 0xF000	‘F15’
KEY_F16	107 | 0xF000	‘F16’
KEY_F17	108 | 0xF000	‘F17’
KEY_F18	109 | 0xF000	‘F18’
KEY_F19	110 | 0xF000	‘F19’
KEY_F20	111 | 0xF000	‘F20’
KEY_F21	112 | 0xF000	‘F21’
KEY_F22	113 | 0xF000	‘F22’
KEY_F23	114 | 0xF000	‘F23’
KEY_F24	115 | 0xF000	‘F24’
KEY_PRINTSCREEN	70 | 0xF000	'PRINTSCREEN'
KEY_PAUSE	72 | 0xF000	‘PAUSE’
KEY_INSERT	73 | 0xF000	‘INSERT’

KEY_HOME	74 | 0xF000	‘HOME’
KEY_PAGE_UP	75 | 0xF000	‘PAGE_UP’
KEY_DELETE	76 | 0xF000	‘DELETE’
KEY_END	77 | 0xF000	‘END’
KEY_PAGE_DOWN	78 | 0xF000	‘PAGE_DOWN’
KEY_RIGHT	79 | 0xF000	‘RIGHT’
KEY_LEFT	80 | 0xF000	‘LEFT’
KEY_DOWN	81 | 0xF000	‘DOWN’
KEY_UP	82 | 0xF000	‘UP’
KEYPAD_1	89 | 0xF000	'NUM_1'
KEYPAD_2	90 | 0xF000	'NUM_2'
KEYPAD_3	91 | 0xF000	'NUM_3'
KEYPAD_4	92 | 0xF000	'NUM_4'
KEYPAD_5	93 | 0xF000	'NUM_5'
KEYPAD_6	94 | 0xF000	'NUM_6'
KEYPAD_7	95 | 0xF000	'NUM_7'
KEYPAD_8	96 | 0xF000	'NUM_8'
KEYPAD_9	97 | 0xF000	'NUM_9'
KEYPAD_0	98 | 0xF000	'NUM_0'
KEYPAD_SLASH	84 | 0xF000	'NUM_SLASH'
KEYPAD_ASTERIX	85 | 0xF000	'NUM_ASTERIX'
KEYPAD_MINUS	86 | 0xF000	'NUM_MINUS'
KEYPAD_PLUS	87 | 0xF000	'NUM_PLUS'
KEYPAD_ENTER	88 | 0xF000	'NUM_ENTER'

KEYPAD_PERIOD	99 | 0xF000	'NUM_PERIOD'