ERAE TOUCH API

The ERAE Touch API is a custom sysex library with messages enabling you to take full control of the ERAE Touch point detection and LEDs states.

Contents

API Messages

API Mode enable

Full sysex message

```
F0 00 21 50 00 01 00 01 01 01 04 01 RECEIVER PREFIX BYTES F7
```

Message break down

F0	SysEx message begin
00 21 50 00 01 00 01	Erae Touch identifier prefix
01	Erae Touch current MIDI network ID
01	Erae Touch Service
04	Erae Touch [Service] API
01	Erae Touch [Service API] enable
BBs	Receiver Prefix Bytes
F7	SysEx message end

Messages sent by the Erae Touch API to the receiver will begin with the chosen ‘Receiver Prefix Bytes’. You must set at least 1 byte and max 16 bytes.

Please send an API Mode disable message before sending a new API Mode enable message.

API Mode disable

Full sysex message

```
F0 00 21 50 00 01 00 01 01 01 04 02 F7
```

Message break down

F0	SysEx message begin
00 21 50 00 01 00 01	Erae Touch identifier prefix
01	Erae Touch current MIDI network ID
01	Erae Touch Service
04	Erae Touch [Service] API
02	Erae Touch [Service API] disable
F7	SysEx message end

Fingerstream message

This message is sent by the Erae Touch to your receiving device when the API mode is enabled. Messages are sent only when touching an API Zone.

Full sysex message

```
F0 RECEIVER PREFIX BYTES DAT1 DAT2 XYZ1 ... XYZ14 CHK F7
```

Message break down

F0	SysEx message begin
BBs	Receiver Prefix Bytes
DAT1	Action/Finger Index Byte
DAT2	Zone identifier
XYZ1 ... XYZ14	7-bitized 3 floats (X, Y, Z) position
CHK	Checksum (XOR) of the encoded XYZ bytes
F7	SysEx message end

The receiver prefix bytes are the ones given in the API Mode enable message.

‘Action/Finger Index Byte’ is decomposed as 0b0aaaffff where aaa are the action bits (click 0b000 /slide 0b001 /release 0b010) and ffff are the finger index bits.

Zone Boundary Request message

Full sysex message

```
F0 00 21 50 00 01 00 01 01 01 04 10 ZONE F7
```

Message break down

F0	SysEx message begin
00 21 50 00 01 00 01	Erae Touch identifier prefix
01	Erae Touch current MIDI network ID
01	Erae Touch Service
04	Erae Touch [Service] API
10	Erae Touch [Service API] boundary request
ZONE	API Zone Index
F7	SysEx message end

Zone Boundary Reply message

The API mode must be enabled for this message to be sent by the Erae Touch to your receiving device.

Full sysex message

```
F0 RECEIVER PREFIX BYTES 7F 01 ZONE Width Height F7
```

Message break down

F0	SysEx message begin
BBs	Receiver Prefix Bytes
7F	Non finger data byte
01	Zone boundary reply byte
ZONE	Zone index
Width Height	Width & height of the zone
F7	SysEx message end

The returned size of the zone is (7F, 7F) if the zone is not used in the Erae Touch.

Clear Zone Display

Full sysex message

```
F0 00 21 50 00 01 00 01 01 01 04 20 ZONE F7
```

Message break down

F0	SysEx message begin
00 21 50 00 01 00 01	Erae Touch identifier prefix
01	Erae Touch current MIDI network ID
01	Erae Touch Service
04	Erae Touch [Service] API
20	Erae Touch [Service API] clear zone display
ZONE	Index of zone to be cleared
F7	SysEx message end

Draw pixel

Full sysex message

```
F0 00 21 50 00 01 00 01 01 01 04 21 ZONE XPOS YPOS RED GREEN BLUE F7
```

Message break down

F0	SysEx message begin
00 21 50 00 01 00 01	Erae Touch identifier prefix
01	Erae Touch current MIDI network ID
01	Erae Touch Service
04	Erae Touch [Service] API
21	Erae Touch [Service API] draw pixel
ZONE	Index of zone to be cleared
XY	Coordinates of the pixel (2 bytes)
RGB	Color of the pixel (3 7-bits bytes)
F7	SysEx message end

Draw rectangle

Full sysex message

```
F0 00 21 50 00 01 00 01 01 01 04 22 ZONE XPOS YPOS WIDTH HEIGHT RED GREEN BLUE F7
```

Message break down

F0	SysEx message begin
00 21 50 00 01 00 01	Erae Touch identifier prefix
01	Erae Touch current MIDI network ID
01	Erae Touch Service
04	Erae Touch [Service] API
22	Erae Touch [Service API] draw rectangle
ZONE	Index of zone to be cleared
XY	Coordinates of the pixel (2 bytes)
WH	Width and height of the rectangle (2 bytes)
RGB	Color of the pixel (3 7-bits bytes)
F7	SysEx message end

Draw image

Full sysex message

```
F0 00 21 50 00 01 00 01 01 01 04 23 ZONE XPOS YPOS WIDTH HEIGHT BIN BIN … BIN CHK F7
```

Message break down

F0	SysEx message begin
00 21 50 00 01 00 01	Erae Touch identifier prefix
01	Erae Touch current MIDI network ID
01	Erae Touch Service
04	Erae Touch [Service] API
23	Erae Touch [Service API] draw image
ZONE	Index of zone to be cleared
XY	Coordinates of the pixel (2 bytes)
WH	Width and height of the rectangle (2 bytes)
BIN	7-bitized 24 bits RGB data of the pixels
CHK	Checksum (XOR) of all the ‘BIN’ bytes
F7	SysEx message end

When displaying a large image, it will be best to break down the image into subimages with no more than 32 pixels each. This keeps the messages short enough for it to be managed by your operating system properly.

Example

Draw an image on API Zone 1 at location (bottom left) x = 5, y = 3 of size, width = 2, height = 2. We want to send 24 bit rgb data “white, red, green, blue” (0xFFFFFF, 0xFF0000, 0x00FF00, 0x0000FF) from left to right and bottom to top. The RGB data to be bitized is

```
FF FF FF FF 00 00 00 FF 00 00 00 FF
```

Message

```
F0 00 21 50 00 01 00 01 01 01 04 23 01 05 03 02 02 78 7F 7F 7F 7F 00 00 00 44 7F 00 00 00 7F 3C F7
```

Message break down

Draw an image	F0 00 21 50 00 01 00 01 01 01 04 23
API Zone 1	01
Position x = 5, y = 3	05 03

Width = 2, height = 2	`02 02`
Bitized 24 bits RGB data	`78 7F 7F 7F 7F 00 00 00 44 7F 00 00 00 7F`
Checksum of the bitized data	`3C`
End of message	`F7`

Helper code

7-bit-izing in Python

```python
from functools import reduce

def bitize7chksum(byteArray):
  bitized7Arr = sum(([sum((el & 0x80) >> (j+1) for j,el in enumerate(data[i:min(i+7,len(data))]))] + [el & 0x7F for el
in data[i:min(i+7,len(data))]] for i in range(0, len(data), 7)),[])
  return bitized7Arr + [reduce(lambda x,y: x^y, bitized7Arr)]
```

7-bit-izing & 7-un-bit-izing in C++

```cpp
#include <cstdint>
#include <cstddef>

/**
* @brief Get size of the resulting 7 bits bytes array obtained when using the bitize7 function
*/
constexpr size_t bitized7size(size_t len)
{
 return len / 7 * 8 + (len % 7 ? 1 + len % 7 : 0);
}

/**
* @brief Get size of the resulting 8 bits bytes array obtained when using the unbitize7 function
*/
constexpr size_t unbitized7size(size_t len)
{
 return len / 8 * 7 + (len % 8 ? len % 8 - 1 : 0);
}

/**
* @brief 7-bitize an array of bytes and get the resulting checksum
*
* @param in Input array of 8 bits bytes
* @param inlen Length in bytes of the input array of 8 bits bytes
* @param out An output array of bytes that will receive the 7-bitized bytes
* @return the output 7-bitized bytes XOR checksum
*/
constexpr uint8_t bitize7chksum(const uint8_t* in, size_t inlen, uint8_t* out)
{
 uint8_t chksum = 0;
 for (size_t i{0}, outsize{0}; i < inlen; i += 7, outsize += 8)
 {
   out[outsize] = 0;
   for (size_t j = 0; (j < 7) && (i + j < inlen); ++j)
   {
     out[outsize] |= (in[i + j] & 0x80) >> (j + 1);
     out[outsize + j + 1] = in[i + j] & 0x7F;
     chksum ^= out[outsize + j + 1];
   }
   chksum ^= out[outsize];
 }
 return chksum;
}

/**
* @brief 7-unbitize an array of bytes and get the incomming checksum
```

```
*
* @param in Input array of 7 bits bytes
* @param inlen Length in bytes of the input array of 7 bits bytes
* @param out An output array of bytes that will receive the 7-unbitized bytes
* @return the input 7-bitized bytes XOR checksum
*/
constexpr uint8_t unbitize7chksum(const uint8_t* in, size_t inlen, uint8_t* out)
{
 uint8_t chksum = 0;
 for (size_t i{0}, outsize{0}; i < inlen; i += 8, outsize += 7)
 {
   chksum ^= in[i];
   for (size_t j = 0; (j < 7) && (j + 1 + i < inlen); ++j)
   {
     out[outsize + j] = ((in[i] << (j + 1)) & 0x80) | in[i + j + 1];
     chksum ^= in[i + j + 1];
   }
 }
 return chksum;
}
```