102-9013-00 The FLIR Serial Line Packet and The FLIR Binary Protocol

29-Jul-2016

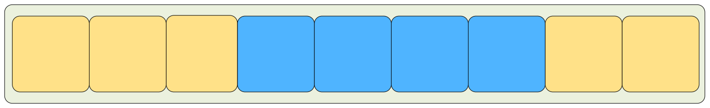


Table of Contents

1. Revision Summary

Date	Change
05-Nov-2015	Initial Release.
10-Nov-2015	Changed to byte-stuffing style framing.
02-Dec-2015	Revised ITAR verbiage. Place into Commodity codes.
17-Dec-2015	Add: Sections: Revision Summary, Document Location (now sections 1&2). Move: CRC to Frame from Payload. Modify: Section 5.1.2, 5.3
03-Feb-2016	Rename document from *FLIR Binary Protocol* to *FLIR Serial Line Packets and FLIR Binary Protocol*. Restructure document into two major parts; Packets and Protocol.
04-Feb-2016	Clarified Physical vs. Logical addressing. Increase FSLP payload size and FBP Data size. Remove ESC bytes from CRC calculation. Updated all drawings. Clarified ERROR conditions.
28-Jul-2016	Removed Appendix B. Byte stuffing now addressed in body of document. Modified byte stuffing – 'escaped' bytes now also undergo a value change. Flattened document structure. Updated and added drawings.
29-Jul-2016	Beautification of pictures and verbage. Address the prohibition of packet interleaving.

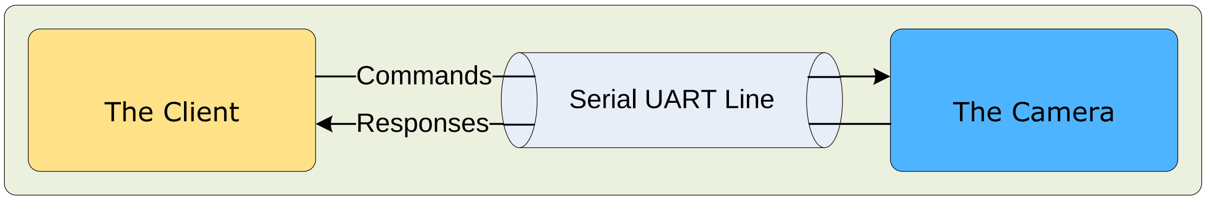
2. Introduction.

As the title suggests, this document describes two separate but related things (1) the FLIR Serial Line Packet (FSLP) and (2) the FLIR Binary Protocol (FBP).

FSLP provides a means for two entities to exchange data over a serial UART connection. FSLP provides for point-to-point communication, meaning that there are only two entities connected to the Serial UART.

FBP is Command/Response in nature. Devices issuing FBP Commands shall be referred to as Clients and devices issuing FBP Responses shall be referred to as Cameras.

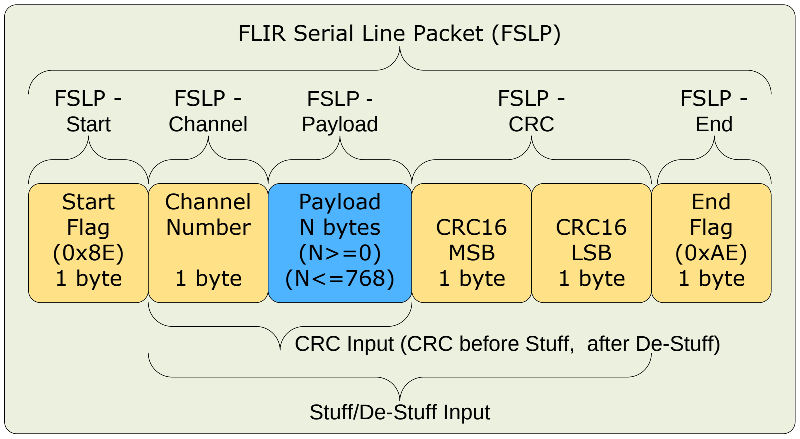
Provided below is a simplified drawing of a Client/Camera pair configured to communicate with each other via FBP encapsulated with the FSLP. It shows some of the relationships between Clients, Cameras, Serial UART Lines, Commands and Responses.



3. FSLP.

3.1. FSLP Overview.

FSLP layout shall be as follows:



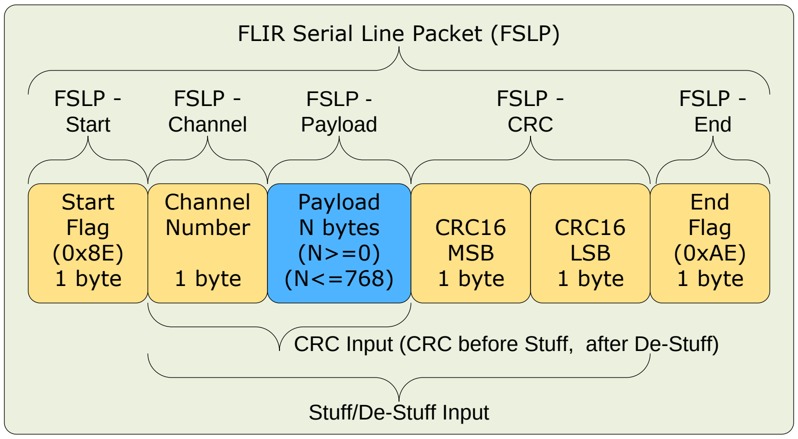
As can be seen in the diagram above, an FSLP has five five parts: Start, Channel, Payload, CRC and End. Each of these parts are described in the following five sections and then a description of the particular method of byte stuffing used in FSLP is presented. The basic idea of byte stuffing, however, is provided below.

Byte Stuffing includes using values for the Start and End Flags (and 'escaping' other occurrences of these values within the packet itself).

Framing via byte stuffing is similar to the way strings are *framed* in C – they're enclosed quotes (the Flags) and if quotes themselves need to appear in the string they are 'escaped'. Similarly if the 'escape' value itself needs to appear it is also escaped. Like this:

```
char *myStr = "Hello World";        printf(myStr);    ->    Hello World
char *myStr = "Hello \"World\"";    printf(myStr);    ->    Hello "World"
char *myStr = "Hello \\World";      printf(myStr);    ->    Hello \World
```

FSLP provides basic routing and CRC services.



3.2. Start Flag.

The Start Flag, shall be a constant single byte, 0x8E.

The primary purpose of the Start Flag is to provide the Client and the Camera a reliable way to recognize the start of a packet. Being able to recognize the start of a packet helps synchronize the Client and Camera.

Note that 0x8E has the MSB set. This value was chosen, in part, because printable ASCII characters do not have the MSB set. As such, using 0x8E as a Start Flag leaves open the possibility of a Camera being able to simultaneously support the FLIR Binary Protocol as well as an ASCII based interface.

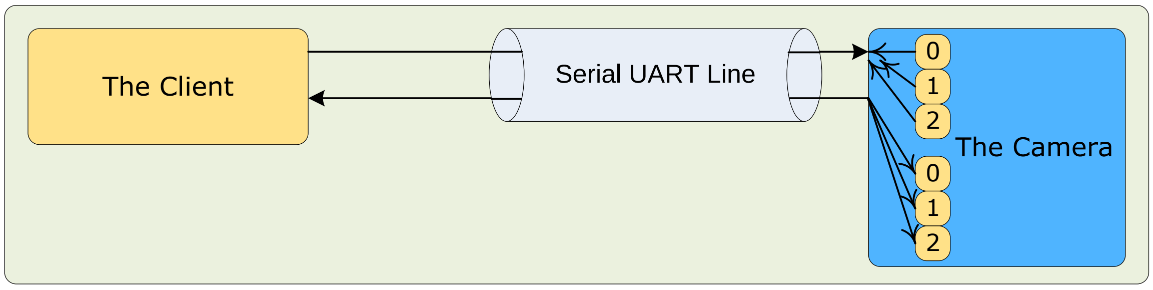
3.3. Channel Number.

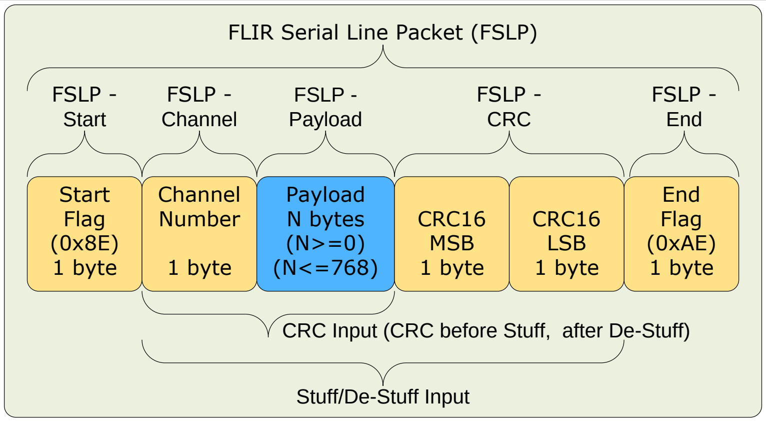
The Channel Number shall also be a single byte. It is used to specify a different logical path, or channel, of communication over the single UART physical interface. For example, a Client may communicate with a Camera in a command/response type fashion on one channel while at the same time (i.e., packet multiplexed) be receiving 'streaming debug' data from the Camera on a different Channel.

Channel 0 shall be reserved in both Clients and in Cameras for The FLIR Binary Protocol. The FLIR Binary Protocol is discussed in Section 5.

A Camera configured to both receive and transmit on three different channels is depicted below.

Commands occurring on Channel "A" shall not be interleaved with Commands occurring on channel "B". Likewise with Responses.

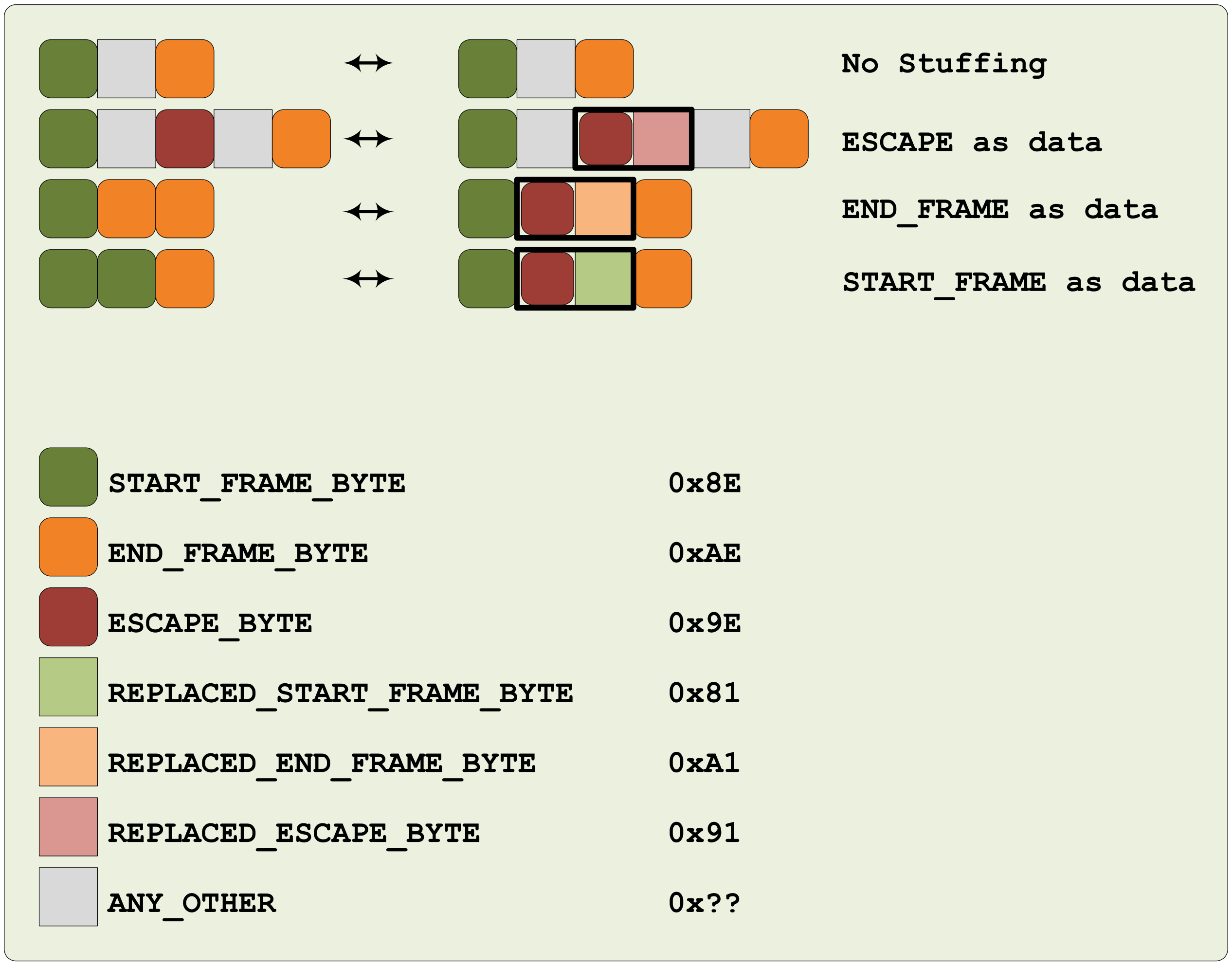


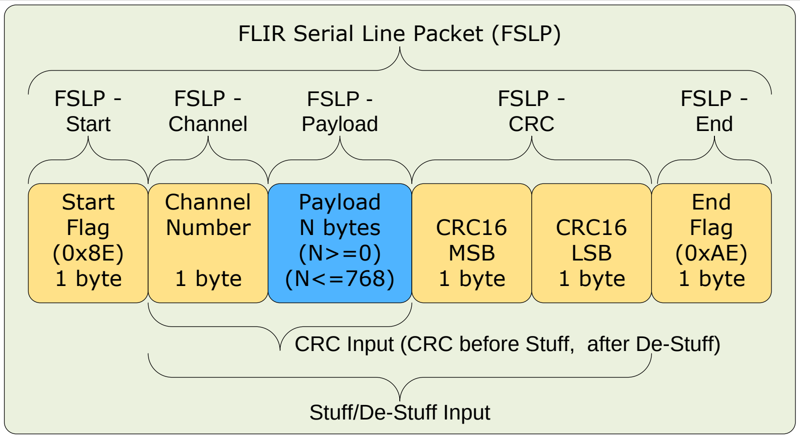


3.4. FSLP Payload.

The Payload section can be any collection of bytes subject only to the constraint that the number of bytes, N, be such that 0 <= N <= 768. ESCAPE values do not contribute to N.

When the special values 0x8E, 0xAE, and 0x9E need to appear in the payload section thay shall be escaped and their values changed per the figure below.





3.5. CRC16.

The 16-bit CRC shall be calculated per Appendix A.

The MSB/LSB of the CRC shall be placed into the FSLP (a byte oriented entity) per the picture at the top of this page.

The CRC validates the integrity of the Packet.

The first byte included in the CRC calculation shall be the Channel Number.

The last byte included in the CRC calculation shall be the last byte of the Payload.

The CRC shall (logically) be performed on a non byte stuffed entity; i.e., the CRC calculation shall be performed on exactly N+3 bytes where N is the number of bytes in the Payload (0<=N<=768).

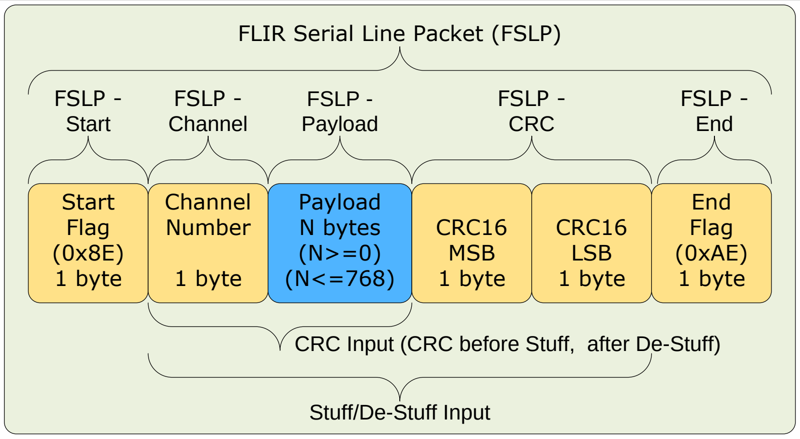
3.6. End Flag.

The End Flag shall be a single constant byte, **0xAE**.

The primary purpose of the End Flag is to provide the Client and the Camera a reliable way to recognize the end of a packet. Being able to recognize the end of a packet helps synchronize the Client and Camera.

3.7. FSLP Transmission.

Referring to the picture at the top of the page, the bytes in the FSLP shall be transmitted in a strictly left to right order. That is, the bytes are transmitted: Start Flag, Channel Number, … CRC16-LSB, End Flag.



3.8. FSLP Error Handling.

Note that true Start Flags and true End Flags will never be escaped.

Bytes received before a Start Flag shall be **discarded**.

Start Flags and all succeeding bytes (up to a maximum number) shall be accepted. Since all bytes between the Start Flag and the End Flag could be 'escaped' the maximum number of accepted bytes shall be:

1 + 1*2 (esc'd Channel Number) + 768*2 (all Payload bytes esc'd) + 2*2 (both CRC bytes esc'd) + 1 = 1544.

Actually, 1544 is the maximum number of bytes that could be transmitted over-the-wire and still represent a valid FSLP entity. Internally, the receiver may choose not to, for example, store the Start and End Flags in a memory buffer. Likewise with the escape bytes.

Therefore, 'conceptually', i.e. the above buffer storage decisions notwithstanding, after receipt of a start byte, if an end byte is not received by the 1544th byte then all 1544 bytes shall be **discarded**.

Once a valid Start Flag has been received and FSLP bytes are being collected then the receipt of another, unescaped, Start Flag shall cause all preceding bytes to be **discarded**. That last received Start Byte shall be considered valid.

Packets with an incorrect CRC shall be **discarded**.

Valid packets (Start, End, CRC all correct) shall have their payloads passed to a Channel handler.

All Channels (even unimplemented Channels) shall have a handler. The 'unimplemented Channel' Payload handler shall simply **discard** the Payload. Note that Channel 0, if implemented, shall always be a FBP handler.

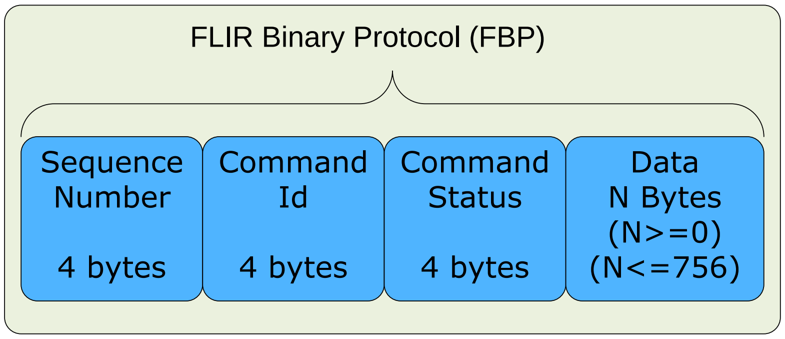
Note: All unimplemented Channels could share a common handler.

Note: Even NULL Payloads (N=0) shall be passed to the handler (e.g., keep alive signal).

4. FBP.

4.1. FBP Overview.

FBP layout shall be as follows:



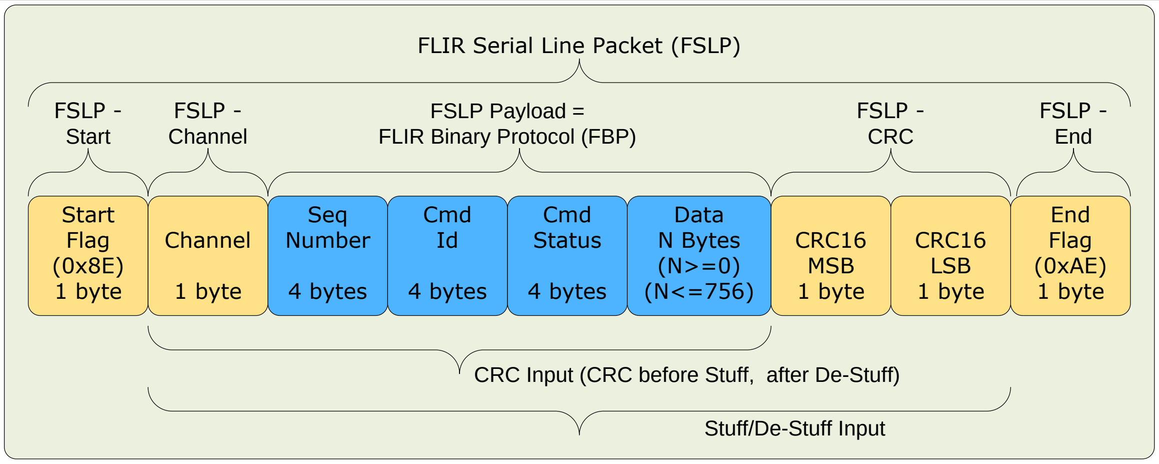
As can be seen, the FLIR Binary Protocol consists of four sections: the Sequence Number, the Command Id, Status and finally, Data.

These bytes are discussed in more detail below but in general they provide a way to mimic the following method of calling a c-function:

```
status = someFunction( inParm1, inParm2, &outValue1, &outValue2 );
```

`status` gets placed in the Command Status field, **`someFunction`** gets placed in the Command Id field and, in the case of a Command, **`inParm1`** and **`inParm2`** get placed in Data field whereas in a Response **`outValue1`**, **`outValue2`** get placed in Data field.

The remainder of this document will always show FBP within FSLP but in reality FBP can exist within any packetization scheme.



4.2. FBP Sequence Number.

The Client can sends any 4-byte value for this field and the Camera will echo it back. The Client is free to use this field in any way it sees fit. The expected use is that Sequence will increase monotonically.

4.3. FBP Command Id.

The Client uses different Command Ids (opCodes) to tell the Camera to do different things. Conceptually, one can think of the Command Id as the address of the subroutine that the Client would like the Camera to execute. The Camera shall echo the Command Id in its response.

4.4. FBP Command Status.

Commands shall always be sent with Status equal to all F's.

Commands received with a Status other that all F's shall be **discarded**.

Responses with a Payload Status of all 0's shall indicate a successful execution of the Command (Data part of the Payload is 'accurate').

Any value other than all zeros (or all F's) shall indicate the Camera encountered an error while trying to execute the functionality associated with the Command Id or that the Command Id itself was invalid.

Specific Command Status values (other than all 0 and all F) are **TBD**.

Note: Per the above, a Response containing all F's for the Command Status probably indicates existence of a 'wrap-around cable'.

4.5. FBP Data.

In the case of a Command, the Client supplies the Data that the Camera will need while it executes the particular Command Id. In the case of a Response, the Camera supplies the Client with the results of whatever calculation or process was performed.

4.6. FBP Endianness.

All simple 'atomic' c data types (char, short, int, long) shall be placed into the FSLP big endian (MSB first). Arrays of these types shall be placed into the FSLP lowest index (x[0]) first.

Data structures other than simple arrays shall be serialized prior to transmission. The serialization shall be performed as follows: **TBD**.

5. Appendix A – CRC Calculation.

```
CRC16_XMODEM_TABLE = [
        0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
        0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
        0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
        0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
        0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
        0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
        0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
        0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
        0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
        0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
        0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
        0xdbfd, 0xcbdc, 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
        0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
        0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
        0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
        0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
        0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
        0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
        0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
        0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
        0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
        0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
        0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
        0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
        0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
        0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
        0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
        0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
        0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
        0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
        0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba, 0x8fd9, 0x9ff8,
        0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0,
        ]


def crc16(data, crc=0x1d0f): #Note the new initial condition is 0x1d0f instead of 0.
# in C:return(USHORT)((crcin << 8) ^ ccitt_16Table[((crcin >> 8)^(data))&255]);
    for byte in data:
        crc = ((crc << 8) & 0xff00) ^ CRC16_XMODEM_TABLE[((crc >> 8) & 0xff) ^ byte]
    return crc & 0xffff
```