Citifax Teletext Inserter

Framing Protocol Specification

Version 1.7 24th June 2008

1 Overview

This document specifies a framing protocol for use with the Teletext Inserter. This is a simple byte-oriented, delimiter-based protocol that can accommodate frames of arbitrary length. A **byte-stuffing** technique makes it possible for the receiver to distinguish payload bytes that happen to have the same value as the control (delimiter and escape) bytes from those that are actually functioning as delimiters.

2 Constants

This protocol uses six constant values: the control bytes used in the protocol are listed in the following table (the names are taken from the ASCII code):

Name	Value (decimal)	Value (hexadecimal)
STX	2	0x02
ETX	3	0x03
DLE	16	0x10

ACK	6	0x06
NAK	21	0x15

Figure 1: Control Chars

3 Frame Layout

The layout of the frame is shown in Figure 2. Each frame begins with a single byte, STX (Hex byte value 0x02). The end of the frame is marked by the single byte ETX (HEX value 3) followed by a single byte checksum. The sending protocol implementation inserts the byte DLE before each occurrence of STX, ETX, and DLE in the payload; the receiving protocol implementation removes any such “stuffed” bytes before passing the payload to the higher layer.

Thus, the framing mechanism is completely transparent to the higher layer(s).

STX	payload = byte–stuffed data (every STX, ETX, DLE preceded by DLE)	ETX	Checksum

Figure 2: Framing layout

As an example, consider a payload consisting of four bytes with the values 3, 2, 1, 0 in that order. The result of framing that payload would be a sequence of 9 bytes with the values:

0x02	0x10 0x03 0x10 0x02 0x01 0x00	0x03	0x01
STX	Payload	ETX	Checksum

Figure 3: Framing example

The checksum of the frame is the XOR’d value of all the bytes in the frame between STX & ETX inclusive, including any DLE’s that might be added into the payload. The STX, ETX & Checksum are outside of the payload (i.e. Frame Header/Footer) and as such are not delimited.

Note that a payload of length n can become a frame of up to length 2n + 3 when it is transmitted “on the wire”.

It may seem that stuffing STX in the payload is unnecessary, since the receiver is only looking for ETX while processing the payload. Note, however, that framing protocols should be designed under the assumption that **a receiver might begin receiving at any point in the input stream of bytes**, in particular in the middle of a payload. The specified stuffing algorithm ensures that when the start delimiter sequence (STX) occurs, this is treated as a start of frame, the next end delimiter (ETX) sequence marks the end of frame and the checksum verifies the frame content. Any repeated start delimiter within the frame will mark the start of a new frame. This allows the receiver to *unambiguously* identify the first complete frame it receives (assuming no errors occur in transmission).

4 Sender Processing

Given a payload, the sender appends the single byte header STX to the beginning of the frame. Then the sender processes each byte of the payload in order, calculating the checksum and inserting the single byte DLE in front of each occurrence of DLE, STX, or ETX in the payload. Finally, the sender appends the single byte ETX and single byte checksum to the stuffed payload.

5 Receiver Processing

The receiver discards all bytes received until it recognizes the start byte STX. It then begins assembling the payload a byte at a time, as follows: If the input byte is ETX, it is discarded, the current payload is delivered to the higher layer, and the process starts over. Otherwise, if the input byte is DLE, it is discarded and the next byte is appended to the current payload. If the input byte is not STX, ETX or DLE, it is appended to the current payload.

Note that the byte following DLE in the payload is examined and should be one of the STX, ETX or DLE bytes; given a properly functioning sender, this will be the case, however the situation where this is not true can occur if a frame is corrupted in transit. This will be seen as a detected error within the frame and the receiver will discard this frame and revert back to looking for a new start or frame.

6 Frame Payload

The payload contained within the frame is of variable length with the following format payload 'length', payload 'type' then the remainder of the payload data.

LEN	TYPE	Variable length Payload

Figure 4: Payload example

7 Inserter Payload type

The Teletext Inserter on receiving a valid frame from the host will process the command and then generate a return frame with the status and any other data to be returned to the sending host.

Frame types currently supported by the inserter are as follows :

Payload Type	Description
0x00	Inserter firmware version, request from host.
0x01	Write row into the inserter, request from host.
0x02	Read Row from the Inserter, request from host.
0x03	Clear Page in the inserter, request from host.
0x04	Set Inserter time and date, request from host.
0x05	Read Inserter time and date, request from host.
0x06	Lock Page out of transmission sequence, request from host.
0x07	Unlock Page back into transmission sequence, request from host.
0x08	Clear individual magazine, request from host.
0x09	Clear all Magazines, request from host.
0x0A	Write Packet 8/30 into the inserter, request from host.
0x0B	Set Insert point, request from host.
0x0C	Read Insertpoint, request from host.
0x7F	Reboot, request from host.
0x80	Inserter firmware version, reply from Inserter.
0x81	Write row into the inserter, reply from Inserter.
0x82	Read Row from the Inserter, reply from Inserter.
0x83	Clear Page in the inserter, reply from Inserter.
0x84	Set Inserter time and date, reply from Inserter
0x85	Read Inserter time and date, reply from Inserter.
0x86	Lock Page, reply from Inserter.
0x87	Unlock Page, reply from Inserter.
0x88	Clear individual magazine, reply from inserter.
0x89	Clear all Magazines, reply from inserter.
0x8A	Write Packet 8/30 into the inserter, reply from inserter.

0x8B	Set Insert point, reply from inserter.
0x8C	Read insert point, reply from inserter.
0xFF	Reboot, reply from Inserter.

Figure5: Framing types

Example Frame to Set Clock

<STX>	*len*	*type*	*secs*	*mins*	*hours*	*date*	*month*	*year*	<ETX>	CS

| |← Payload →| |

|← Frame →|

Example Frame to Set Clock to 14:35:06 14/11/07

0x02	*0x07*	*0x04*	*0x06*	*0x23*	*0x0E*	*0x0E*	*0x0B*	*0x07*	0x03	0x2B

Example Frame to Set Clock to 02:03:36 14/11/07

0x02	*0x07*	*0x04*	*0x24*	*0x10*	*0x03*	*0x10*	*0x02*	*0x0E*	*0x0B*	*0x07*	0x03	0x24
				<DLE>		<DLE>						

Note the <DLE> bytes in the above example preceding the control bytes.

8 Inserter Payload format

Example payload packet types and replies to and from the Inserter.

Read Inserter firmware version, request from host

0x01	0x00

Read Inserter firmware version, reply from Inserter

0x08	0x80	ACK	x	y	z

Send teletext row request from host (*)

0x2C	0x01	magazine	page	row	40 character teletext row

Send teletext row reply from Inserter

0x02	0x81	ACK/NAK

Read Teletext Row request from host

0x04	0x02	magazine	page	row

Read Teletext Row reply from Inserter (*) (**)

0x2A	0x82	ACK	40 character teletext row

Read Teletext Row reply from Inserter (**)

0x02	0x82	NAK

Clear Teletext Page request from host

0x03	0x03	magazine	page

Clear Teletext Page reply from Inserter

0x02	0x83	ACK/NAK

Set Inserter Time & Date request from host (***)

0x07	0x04	secs	mins	hours	date	month	year

Set Inserter Time & Date, reply from Inserter

0x02	0x84	ACK/NAK

Read Inserter Time & Date request from host

0x01	0x05

Read Inserter Time & Date, reply from Inserter (**) (***)

0x08	0x85	ACK	secs	mins	hours	date	month	year

Read Inserter Time & Date, reply from Inserter (**)

0x02	0x85	NAK

Lock a page out of transmission sequence, request from host.(****)

0x03	0x06	magazine	page

Lock a page out of transmission sequence, reply from Inserter.

0x02	0x86	ACK/NAK

Unlock the page back into transmission sequence, request from host.

0x01	0x07

Unlock the page back into transmission sequence, reply from Inserter.

0x02	0x87	ACK/NAK

Clear individual magazine, request from host.

0x02	0x08	magazine

Clear individual magazine, reply from Inserter.

0x02	0x88	ACK/NAK

Clear all magazines, request from host.

0x01	0x09

Clear all magazines, reply from Inserter.

0x02	0x89	ACK/NAK

Send Packet 8/30 request from host (*)

0x29	0x0A	40 character Packet 8/30 row

Send Packet 8/30 reply from Inserter

0x02	0x8A	ACK/NAK

Set Insert Point, request from host

0x03	0x0B	Point	Lines

Set Insert Point, reply from Inserter

0x02	0x8B	ACK/NAK

Read Insert Point, request from host

0x03	0x0C

Read Insert Point, reply from Inserter

0x02	0x8C	ACK	x	y

Reboot Inserter, request from host.

0x01	0x7F

Reboot Inserter, reply from Inserter.

0x02	0xFF	ACK/NAK

In their status all successful packets will return ‘ACK’; all unsuccessful packets will return ‘NAK’.

(*) Note all row data is to be transferred to and from the inserter with all Hamming and parity correct, as it would be transmitted to air.
(**) Replies containing data in the payload will only have the data bytes filled if the packet status is ‘ACK’, if the status is ‘NAK’ the data bytes will not be returned.
(***) All times are in 24hr format.
(****) Only one page can be locked from the transmission sequence, this feature is used to allow editing of a page and to ensure page is not transmitted while being edited.