Terminal Emulation

- Once a physical connection has been established between two DTEs, the DTEs are then instructed to transmit and receive data over this connection.

- Typically these instructions are carried out by applications such as HyperTerminal, xterm, etc.

- In addition to file transfer functions, these programs also provide "terminal emulation" for computers. This means that the program can be told to make the machine behave as if it were a terminal.

- In this mode, the host will send the characters you type on the keyboard out the serial port, and to display all the characters that arrive at the serial port on the screen.

- No error detection or correction is done, any more than a real terminal would do.

- The particular terminal being emulated may range from a "dumb" terminal with no special features to some particular "smart" terminal, depending on the needs, whim, ambition, and skill of the contributing programmer.

The Layered approach

- In the design and development of any communication application it is efficient to adopt a layered (modular) approach for reasons of reducing complexity and standardization.

- A good example of such a model is the OSI layered model. Though it is largely defunct as an industry standard, it does serve as a very useful model for software development.

- Layers will be implemented as the level of services increases or becomes increasingly complex. In other words not all layers will be implemented initially.

- For example, consider a dumb terminal emulator which will only require the physical and session layer functions:

Physical Layer

- This layer will contain the RS-232 and system level flow control functions.
- The functions in this layer will provide access to the physical communications link, or to the operating system functions (device drivers) that control and service it.
- The content of these functions is necessarily system-dependent. For example the system calls to access the serial port for Win32 are quite different from those for UNIX.
- As a matter of practice, these functions would be kept together in a separate module with its own set of variables, allowing functions to share file descriptors, modes and similar information.
- This system dependent module should be completely replaceable by an equivalent module for a different system.
- Examples of functions are:
    - ***initialize*** : Initialize the port, baud, parity, etc.
    - ***rx_stat*** : receive status of the port.
    - ***tx_stat*** : transmit status of the port.
    - ***rx_char*** : receive a character from the port.
    - ***tx_char*** : transmit a character to the port.

Session Layer

- In this case all the session layer will do is to establish a session i.e., it will enable the user to connect (session), initialize or terminate the session.
- Examples of this function are:
    - ***initialize*** : get initialization parameters from user.
    - ***connect*** : "dumb" terminal emulation function.
    - ***exit*** : terminate the session.

Intelligent Terminal Emulation

- The number of layers and functions will increase substantially as in the case of a full-blown communications program.

- To illustrate this point, consider a program which has and "intelligent" terminal emulator and full file transfer capabilities:

Physical Layer

- basically same as above.

Data Link Layer:

- Error detection/correction, flow control, acknowledgments and negative acknowledgments (ACK/NAK).

- Packet framing, encoding and decoding packets. This includes packet sequencing, length, etc.

Network Layer

Null. This is a point-to-point link.

Transport Layer

- Sequence assurance, i.e., ensure that the packets are delivered and received with no errors and in the proper sequence.

- Provide a reliable data stream to the session layer. In other words get data from a file during a file transfer and send packetized data to the session layer.

Session Layer

- Basically the same as described above.

- Establish a logical link between two processes and call upon the transport layer to provide the processes with reliable streams of data.

Presentation Layer

- Format and character set conversion for filenames and textual data.
- An important goal for file transfers is that text files remain useful after transfer between any pair of systems, no matter how different their text file formats.
- The program need not have detailed knowledge of the text file format of every conceivable system it could send text to, a common intermediate representation for text files is selected, which we call the canonical form.
- Canonic form for text files is simply this: a stream of ASCII characters, one per 8-bit byte, with each line (record) terminated by a carriage return and a line feed (CRLF).
- On many systems (Win32, UNIX) no conversion need be done at all. On others (such as IBM mainframes), both character set (EBCDIC/ASCII) and record format (block/stream) conversion are required.
- For intelligent terminal emulation, this layer would contain the functions for translating certain character sequences when certain keys (including function keys) are pressed.
- Functions to provide change character display attributes, cursor positioning, etc.

Application Layer

- Implemented to provide a high-level GUI for users to access and use the communication functions.

The UART

- In the early days, it was rarely possible to connect computing equipment from two different manufacturers. The reason being, there was little regard for standards (when they existed at all).
- By 1969, the RS-232 standard had matured to its present level (RS-232-C), and it was not long before manufacturers began to mass-produce a device to implement this standard.
- Today, thanks to this device, called a Universal Asynchronous Receiver/Transmitter (UART), the RS-232 connections are taken for granted.
- The UART allows a wide variety of terminals to communicate uniformly with a large variety of computers, and computers themselves to exchange data over asynchronous serial connections.

- The UART's task is to convert a byte in the computer's memory to a series of voltages on the communication line, and vice versa.
- To transmit a byte, the UART simply feeds the bits (in voltage form) to the line, one at a time, in the prescribed order. The order is that the least significant (low order) bits go first.
- The transmitting UART observes a "framing" convention, which works as follows:
- Whenever no data is flowing (idle state), the UART applies a steady high voltage (mark or binary 1) to the line.
- To signal the beginning of a character, the UART now drops the line to a low voltage (space or binary 0) for a one-bit duration.
- As soon as the receiving UART sees this high to low transition it samples the line voltage eight more times to assemble a byte.
- Now an additional bit is required to signal the end of the character. The sending UART now raises the line to a mark state for a one-bit duration.
- Note that after sampling eight bits after a start bit, the receiving UART checks to make sure that the line is in mark state (i.e. sample the stop bit).
- If not, it informs the computer that a framing error has occurred.
- Once the character has been received, the UART copies the character from the shift register to a holding register and indicates to the computer that a new character has arrived.
- It should be apparent now why the asynchronous serial communication medium is said to be character oriented. The computer is shielded from the serial nature of the transmission by the UART.
- The computer sees the UART, and therefore the communication line, strictly as a character-at-a-time input/output device.