

## *Functions Used*

### *Serial Data Transmission in Asynchronous Mode*

-----

1. In this task example, a Serial Communication Interface (SCI) is used for serial data transmission in asynchronous mode. Figure 2 shows a block diagram of serial data transmission in asynchronous mode which is described below.

- In asynchronous mode, serial data communication is performed asynchronously, with synchronization provided character by character.
- Serial data can be communicated with standard asynchronous communication LSIs such as Universal Asynchronous Receiver/Transmitter (UART) and Asynchronous Communication Interface Adapter (ACIA).
- A multi-processor communication function is provided to enable serial data communications with multiple processors.
- The transfer format can be selected from 16 transfer format types.
- The transmitter and receiver are independent, enabling simultaneous transmission and reception. Both the transmitter and receiver have a double-buffer architecture to achieve continuous transmission and reception.
- Any desired bit rate can be selected using the on-chip baud rate generator.
- The transmit/receive clock source can be selected from internal or external clocks.
- There are six interrupt factors, namely, transmit complete, transmit data empty, receive data full, overrun error, framing error and parity error.
- The Receive Shift Register (RSR) is a register to receive serial data. Serial data input from RXD32 pin is set in RSR in the receiving order that is starting from the LSB (Bit 0), and is converted into parallel data. When one-byte data is received, it is transferred automatically to RDR. RSR cannot be read from or written to directly by the CPU.
- The Receive Data Register (RDR) is an 8-bit register to store received serial data. Receiving one-byte data, the received data is transferred from RSR to RDR to complete receive operation. RSR is then ready to receive data. RSR and RDR have a double buffer, enabling continuous receive operations. RDR is a receive-only register and cannot be written to by the CPU.
- The Transmission Shift Register (TSR) is a register to transmit serial data. Transmit data is temporarily transferred from TDR to TSR and is sent to TXD32 pin starting from the LSB (Bit 0) for serial data transmission. Transmitting one-byte data, the next transmit data is transferred automatically from TDR to TSR to start transmitting. If data is not

written in TDR (1 is set in TDRE), data is not transferred from TDR to TSR. TSR cannot be read from or written to directly by the CPU.

- The Transmit Data Register (TDR) is an 8-bit register to store transmit data. Detecting that TSR is "empty", transmit data written in TDR is transferred to TSR to start serial data transmission. By writing next transmit data in TDR during TSR serial data transmission, continuous transmission is possible. TDR can always be read from or written to by the CPU.
- The Serial Mode Register (SMR) is an 8-bit register for setting of a serial data transfer format and selecting a clock source for the baud rate generator. SMR can always be read from or written to by the CPU.
- The Serial Control Register 3 (SCR3) is an 8-bit register for selecting transmit/receive operation, clock output in asynchronous mode, interrupt request enable/disable, and transmit/receive clock source. SCR3 can always be read from or written to by the CPU.
- The Serial Port Control Register (SPCR) is an 8-bit register to control P42/TXD32 pin. In this task example, P42/TXD32 pin is set as TXD32 output pin, and the input data of TXD32 pin is set not to be inverted.
- The Serial Status Register (SSR) is an 8-bit register with on-chip status flags indicating operation status of SCI3, and multi-processor bits. SSR can always be read from or written to by the CPU, except 1 cannot be written in TDRE, RDRF, OER, PER or FER. 1 must be read in advance to clear them by writing 0. TEND and MPBR are for read only and data cannot be written in them.
- The Bit Rate Register (BRR) is an 8-bit register to set a transmit/receive bit rate matched to the operating clock for baud rate generator selected by CKS0 and CKS1 in SMR. BRR can always be read from or written to by the CPU.
- Table 1 shows an example of BRR setting in asynchronous mode. Table 1 shows values in the active mode when OSC is 10 MHz.

**Table 1 Example of BRR Settings for Bit Rates (Asynchronous Mode)**

R Bit Rate (Bps)	110	150	200	250	1200	2400	31250
n	2	2	2	2	0	0	0
N	88	64	48	38	129	64	4
Error(%)	-0.25	+0.16	-0.35	+0.16	+0.16	+0.16	0

Notes: 1.Set errors to be less than 1%.

2.BRR set values can be calculated as follows:

$$N = \frac{OSC}{64 \times 2^{2n} \times B} \times 10^6 - 1$$

B:Bit rate (bps)

N:Set value of baud rate generator BRR ( $0 \leq N \leq 255$ )

OSC:Value of  $f_{OSC}$  (MHz) = 10 MHz or subclock f w 32.768 kHz

n:Value set in CKS1 and CKS0 in SMR ( $0 \leq n \leq 3$ )

(See Table 2 for the relation between n and clock.)

**Table 2 Relationship between n and Clock**

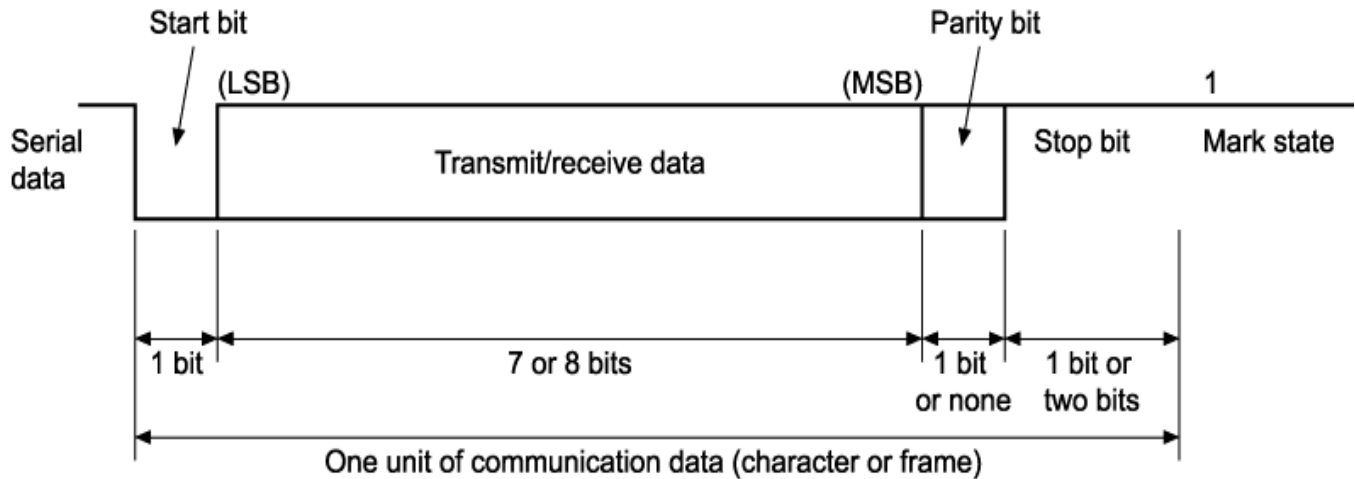
n	Clock	Set Value of SMR	
		CKS1	CKS0
0	$\tilde{O}$	0	0
1	$\tilde{O}w/4, \tilde{O}w$	0	1
2	$\tilde{O}/16$	1	0
3	$\tilde{O}/64$	1	1

3.The error shown in Table 1 is given by the following equation.( rounded off to two decimals )

$$\text{Error (\%)} = \left\{ \frac{\phi \times 10^6}{(N+1) \times B \times 64 \times 2^{2n-1}} - 1 \right\} \times 100$$

4.When OSC is 10 MHz, the maximum bit rate (asynchronous mode) is 31250 bps, provided n=0 and N=4 are set.

- In asynchronous mode, serial communication is performed with synchronization provided character by character, transmitting and receiving characters added with a start bit indicating the start of communication and a stop bit indicating the end of communication.
- The transmitter and receiver are independent inside SCI3 and full duplex communications are possible. Both the transmitter and receiver have a double-buffer architecture to achieve continuous transmission and reception. Data writing during transmission and data reading during reception can make continuous transmission and reception possible.
- Figure 3 shows data format of asynchronous communications. In asynchronous communications, the communication line is normally maintained in the mark state ("High" level). SCI3 monitors communication line and starts serial communications when it detects the place which has become a space ("Low" level) to serve as a start bit.
- One character in communication data consists of the start bit ("Low" level), followed by transmit/receive data (LSB first, starting from the least significant bit), parity bit ("High" or "Low" level) and stop bit ("High" level) at the end.
- In asynchronous mode, synchronization is achieved by the falling edge of the start bit during reception. Data is sampled on the eighth clock of a frequency obtained by multiplying 16 times the one bit period and communication data is fetched in the center of each bit.



**Figure 3 Data Format in Asynchronous Communications**

- SCI3 clock (SCK32) is a clock input/output pin of SCI3.
- SCI3 receive data input (RXD32) is a receive data input pin of SCI3.
- SCI3 transmit data output (TXD32) is a transmit data output pin of SCI3.
- SCI3 interrupt factors total six, transmit complete, transmit data empty, receive data full and three receive errors (overrun error, framing error and parity error). Common vector address is assigned to them.
- Each interrupt request can be enabled/disabled by TIE and RIE in SCR3.
- If TDRE in SSR is set to 1, TXI is generated. If TEND in SSR is set to 1, TEI is generated. These two interrupts are generated during transmission.
- The initial value of TDRE in SSR is 1. Therefore, by setting TIE in SCR3 to 1 and by enabling a transmit data empty interrupt request (TXI) before transferring transmit data to TDR, TXI is generated even when transmit data is not ready.
- The initial value of TEND in SSR is 1. Therefore, by setting TEIE in SCR3 to 1 and by enabling a transmit end interrupt request (TEI) before transferring transmit data to TDR, TEI is generated even when transmit data is not sent.
- By processing which transfers transmit data to TDR within the interrupt handling routine, these interrupts can be utilized effectively. To prevent these interrupt requests (TXI and TEI), the enable bits (TIE and TEIE) interacting to these interrupt requests should be set to 1 after transmit data has been transferred to TDR.
- RXI is generated when RDRF in SSR is set to 1. ERI is generated when OER, PER or FER is set to 1. These two interrupt requests are generated during reception.

2. Table 3 shows assignment of functions in this task example. Serial data transmission in asynchronous mode is performed by assigning the functions as shown in Table 3.

**Table 3 Assignment of Functions**

<b>Function</b>	<b>Assignment</b>
TSR	A register to transmit serial data
TDR	A register to store transmit data
SMR	Sets a serial data transfer format and clock source for baud rate generator
SSR	Status flags to indicate operation status of SCI3
BRR	Sets transmit/receive bit rate
SCR3	Enables transmit operation and sets TXD32 output pin
TXD32	SCI3 transmit data output pin
SPCR	Sets TXD32 output pin

---

**Contents    Specifications    Functions Used    Operations    Software**

---



---



---

## **Built-in Peripheral Functions**

8-bit Counter Count-Up by Interval Function	Simultaneous Serial Data Transmission and Reception in Asynchronous Mode	Asynchronous Event Counter
LED Flickering by Clock Time-Base Function	Multiprocessor Communication	LCD Display with Static Duty
Interrupt Period Setting by Auto-Reload Timer Function	Voltage Measurement by 4-Channel A/D Converter	LCD Display with 1/4 Duty
Pulse Frequency Measurement by Event Counter Function	Duty Pulse Output by 10-Bit PWM Function	Oscillation Stabilization Time Settings
Interrupt Counting by 16-Bit Timer Counter Function	Flickering of LEDs Connected to I/O Port	Module Standby Mode Settings
Count of Input Pulses by 16-Bit Event Counter Function	Count Start by External Interrupt	Clock Operation Using Timer F
PWM Output by Output Compare Function	Multiple Interrupt Operation by Internal Interrupt	ii
Pulse Period Measurement by Input Capture Function	Transition to Sleep (Medium Speed) Mode	ii
Watchdog Timer	Transition to Sleep (High Speed) Mode	ii
Serial Data Transmission in Synchronous Mode	Transition to Standby Mode	ii
Serial Data Reception in Synchronous Mode	Transition to Watch Mode	ii
Simultaneous Serial Data Transmission and Reception in Synchronous mode	Transition to Subsleep Mode	ii
Serial Data Transmission in Asynchronous Mode	Transition to Subactive Mode	ii
Serial Data Reception in Asynchronous Mode	Transition to Active (Medium Speed) Mode	ii

# Older One For All IR Remote Serial Protocol

This protocol supposedly works for:

- URC-4000 (One For All 6)
- URC-5000 (One For All 12)

For newer remotes, [click here](#).

## Serial Settings

4800 baud, 1 start bit, 8 data bits, no parity, 1 stop bit, half-duplex.

## DTR High, RTS Low

The active components hidden in the serial cable's DB9 housing draw power from DTR. Before communication, you should lower DTR and CTS: this resets the circuitry. During communication, you must raise DTR to power the serial cable.

## Wake Up Sequence

You must repeat this wake up sequence for each command you send to the remote.

To wake up the remote:

1. raise DTR (to power the serial cable)
2. send a serial BREAK for at least 50 msec (15 msec minimum, but some remotes take longer than that, maybe even 100 msec)
3. receive a wake-up acknowledge from the remote: 0x6E
4. send a serial execute command to the remote: 0xBC
5. receive a serial execute command acknowledge: 0x6F

Once the remote's awake, you can send any single-byte keycode to the remote. The remote will go to sleep after execute the keycode. The remote will echo back the keycode.

If you send a macro command that issues multiple keycodes, the remote echoes back only the last keycode.

After each command:

1. lower DTR (power down the serial cable)
  2. wait 200 msec
-

## Table of Keycodes

These keycodes are probably wrong: each new OFA model has a unique set of keycodes. If you map out a particular unit's keycodes, please send a list to Rob at [remotes@stormloader.com](mailto:remotes@stormloader.com) to add to the codes page.

Name	Code	Name	Code	Name	Code	Name	Code	Name	Code	Name	Code	Name	Code
	00	6	10		20	Aux	30	Enter	40	Play	50		
Mute	01	3	11	B/Audio	21	TV	31		41	Pause	51	Sleep	
Vol -	02		12	Amp	22		32		42	Stop	52	F1	
Vol +	03		13	VCR	23	CH +	33	FF	43	Display	53		
	04	C/Tuner /Video	14	7	24	CH -	34		44		54		
Power	05	Cable	15	4	25		35		45	F2	55		
CD	06	8	16	1	26		36	A/B	46	Recall	56		
Satellite	07	5	17		27	Record	37	F3	47		57		
9	08		18		28	Program	38	0	48		58		
	09	2	19	A	29	F4	39		49	Rewind	59		



# The RS232 STANDARD

*A Tutorial with Signal Names and Definitions*

(renamed the "EIA232 Standard" in the early 1990's)

Written by Christopher E. Strangio

Copyright © 1993-2003 by CAMI Research Inc., Lexington, Massachusetts

Send Us Your Comments . . .

## Contents

What is EIA232?  
Likely Problems when Using an EIA232 Interface  
Pin Assignments  
Cable Wiring Examples (New!)  
Signal Definitions  
Signal Ground and Shield  
Primary Communications Channel  
Secondary Communications Channel  
Modem Status and Control Signals  
Transmitter and Receiver Timing Signals  
Channel Test Signals  
Electrical Standards  
Common Signal Ground  
Signal Characteristics  
Signal Timing  
Accepted Simplifications of the Standard

Pin Description Index

References to EIA Publications

[Back to CableEye® Home Page](#)

---

## What is EIA232?

[Next Topic](#) | [TOC](#)

In the early 1960s, a standards committee, today known as the Electronic Industries Association, developed a common interface standard for data communications equipment. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the

interconnection of equipment produced by different manufacturers, thereby fostering the benefits of mass production and competition. From these ideas, the RS232 standard was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

Over the 40+ years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232E standard introduced in 1991. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

---

## **Likely Problems when Using an EIA232 Interface**

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

During this 40-year-long, rapidly evolving period in electronics, manufacturers adopted simplified versions of this interface for applications that were impossible to envision in the 1960s. Today, virtually all contemporary serial interfaces are EIA232-like in their signal voltages, protocols, and connectors, whether or not a modem is involved. Because no single "simplified" standard was agreed upon, however, many slightly different protocols and cables were created that obligingly mate with any EIA232 connector, but are incompatible with each other. Most of the difficulties you will encounter in EIA232 interfacing include at least one of the following:

1 - The absence or misconnection of flow control (handshaking) signals, resulting in buffer overflow or communications lock-up.

2 - Incorrect communications function (DTE versus DCE) for the cable in use, resulting in the reversal of the Transmit and Receive data lines as well as one or more handshaking lines.

3 - Incorrect connector gender or pin configuration, preventing cable connectors from mating properly.

Fortunately, EIA232 driver circuitry is highly tolerant of misconnections, and will usually survive a drive signal being connected to ground, or two drive signals connected to each other. In any case, if the serial interface between two devices is not operating correctly, disconnect the cable joining this equipment until the problem is isolated.

---

## **Pin Assignments**

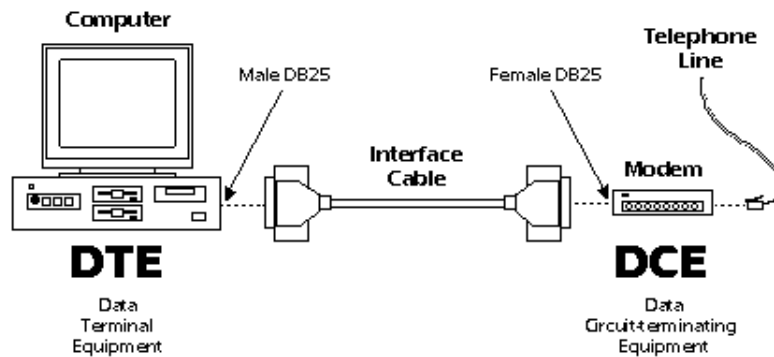
[Next Topic](#) | [Previous Topic](#) | [TOC](#)

Go to DTE Pinout (looking into the computer's serial connector)

Go to DCE Pinout (looking into the modem's serial connector)

If the full EIA232 standard is implemented as defined, the equipment at the far end of the connection is

named the DTE device (Data Terminal Equipment, usually a computer or terminal), has a male DB25 connector, and utilizes 22 of the 25 available pins for signals or ground. Equipment at the near end of the connection (the telephone line interface) is named the DCE device (Data Circuit-terminating Equipment, usually a modem), has a female DB25 connector, and utilizes the same 22 available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable with no cross-overs or self-connects in the connector hoods. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. This drawing shows the orientation and connector types for DTE and DCE devices:



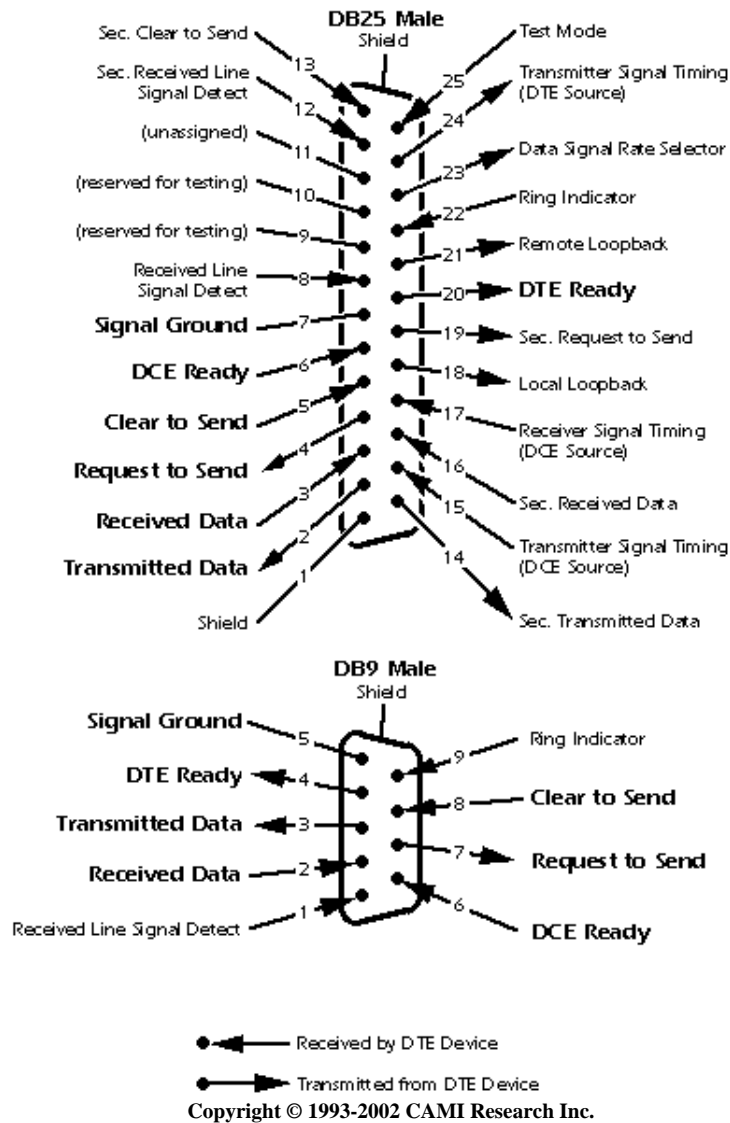
*EIA232 communication function and connector types for a personal computer and modem. DCE devices are sometimes called "Data Communications Equipment" instead of Data Circuit-terminating Equipment.*

---

Here is the full EIA232 signal definition for the DTE device (usually the PC). The most commonly used signals are shown in bold.

[back to Pin Assignments description]

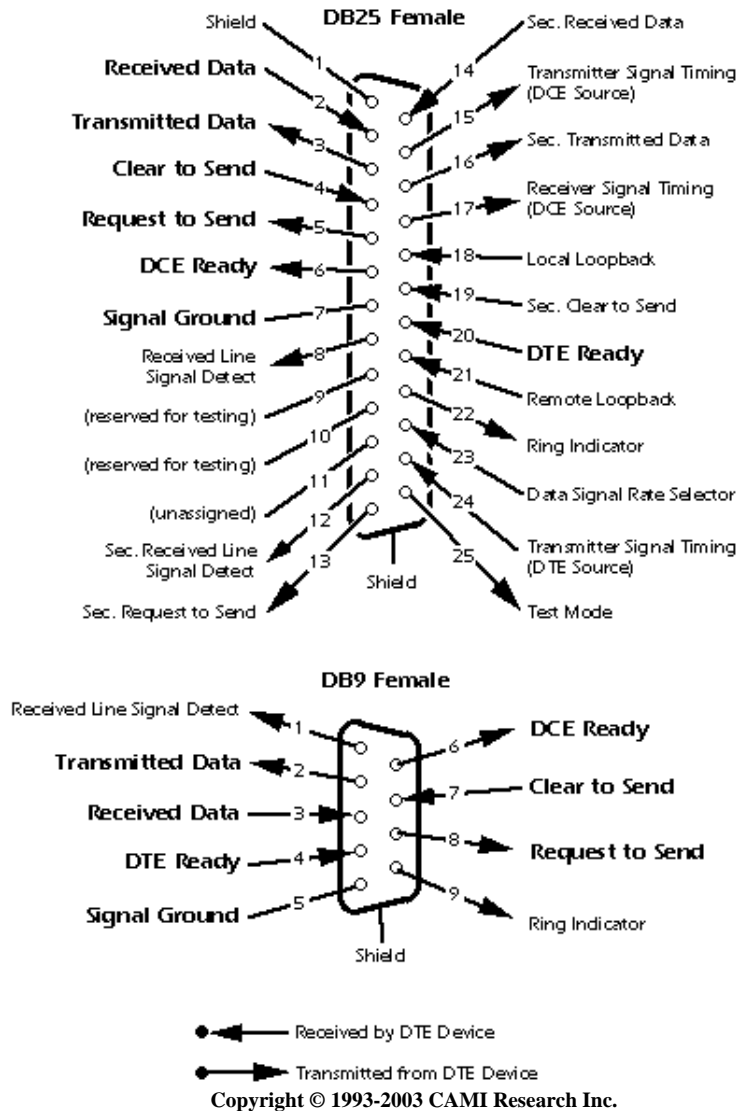
## Looking Into the DTE Device Connector



This shows the full EIA232 signal definition for the DCE device (usually the modem). The most commonly used signals are shown in bold.

[back to Pin Assignments description]

## Looking Into the DCE Device Connector



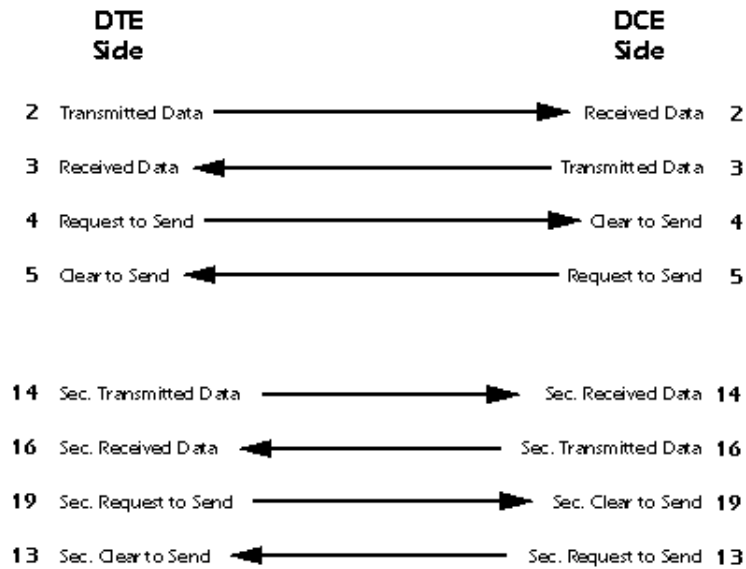
Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, far fewer signal lines are necessary.

You may have noticed in the pinout drawings that there is a secondary channel which includes a duplicate set of flow-control signals. This secondary channel provides for management of the remote modem, enabling baud rates to be changed on the fly, retransmission to be requested if a parity error is detected, and other control functions. This secondary channel, when used, is typically set to operate at a very low baud rate in comparison with the primary channel to ensure reliability in the control path. In addition, it may operate as either a simplex, half-duplex, or full-duplex channel, depending on the capabilities of the modem.

Transmitter and receiver timing signals (pins 15, 17, and 24) are used only for a synchronous

transmission protocol. For the standard asynchronous 8-bit protocol, external timing signals are unnecessary.

**IMPORTANT:** Signal names that imply a direction, such as Transmit Data and Receive Data, are named from the point of view of the DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, this is not done in practice by most engineers, probably because no one can keep straight which side is DTE and which is DCE. As a result, direction-sensitive signal names are changed at the DCE side to reflect their *drive direction* at DCE. The following list gives the conventional usage of signal names:




---

## Cable Wiring Examples

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

The following wiring diagrams come from actual cables scanned by the CableEye® PC-Based Cable Test System. CableEye’s software automatically draws schematics whenever it tests a cable. [Click here](#) to learn more about CableEye.

### 1 - DB9 All-Line Direct Extension

[Next Cable](#) | (no previous cable) || [Next Topic](#)

This shows a 9-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 9 pins plus shield are directly extended from DB9 Female to DB9 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any device that uses a DB9 connector to a PC’s serial port.

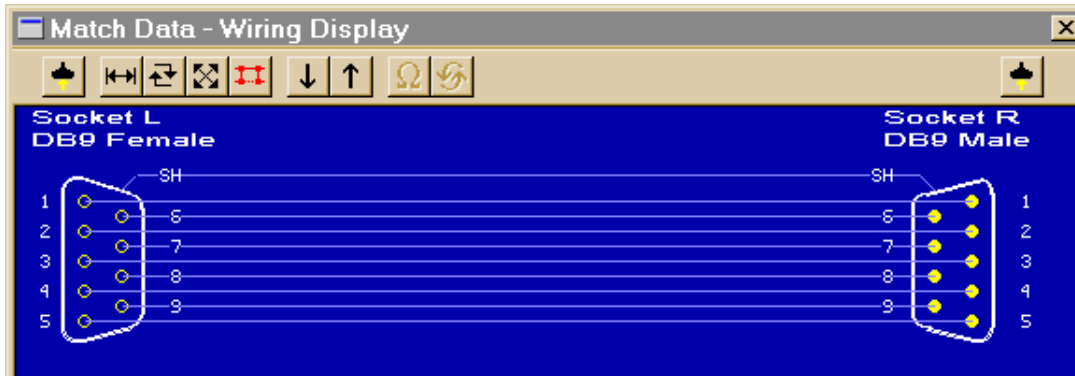


80K

This cable may also serve as an extension cable to increase the distance between a computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

*Left Side:* Connect to **DTE**  
(computer)

*Right Side:* Connect to **DCE** (modem or other  
serial device)



Cable image created by CableEye®

## 2 - DB9 Loopback Connector

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB9 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable to verify the cable.

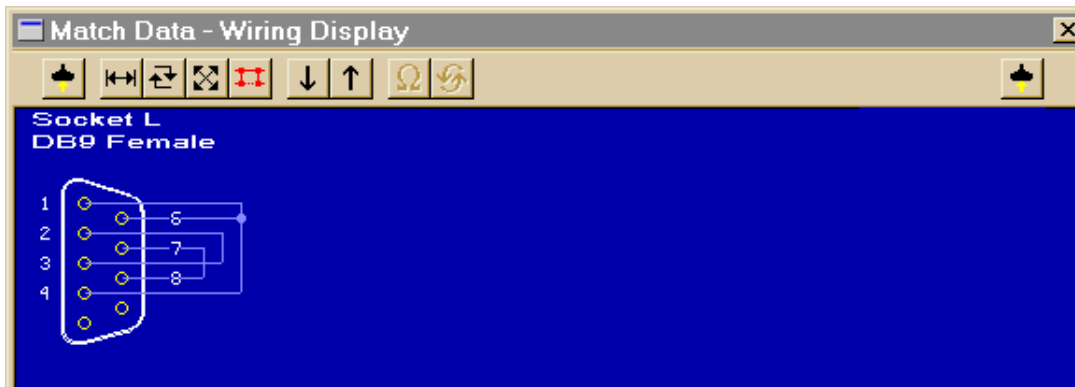


80K

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

*Left Side:* Connect to **DTE** (computer)

*Right Side:* (none)



Cable image created by CableEye®

---

### 3 - DB9 Null Modem Cable

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.



80K

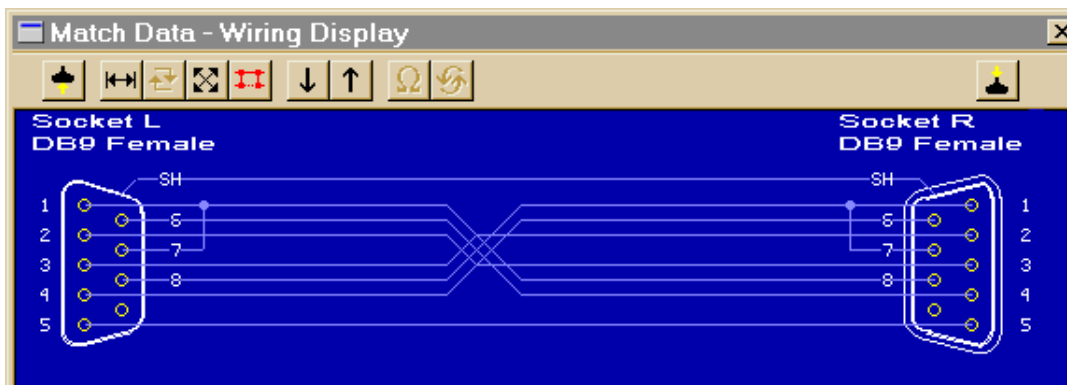
The cable shown below is intended for RS232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals, and a DB25 connector would be necessary.

NOTE: Not all null modem cables connect handshaking lines the same way. In this cable, Request-to-Send (RTS, pin 7) asserts the Carrier Detect (pin 1) on the same side and the Clear-to-Send (CTS, pin 8) on the other side of the cable.

This device may also be available in the form of an adapter.

*Left Side:* Connect to 9-pin **DTE**  
(computer)

*Right Side:* Connect to 9-pin **DTE**  
(computer)



Cable image created by CableEye®

---

### 4 - DB25 to DB9 Adapter

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Signals on the DB25 DTE side are directly mapped to the DB9 assignments for a DTE device. Use this to adapt a 25-pin COM connector on the back of a computer to mate with a 9-pin serial DCE device, such as a 9-pin serial mouse or modem. This adapter may also be in the form of a cable.

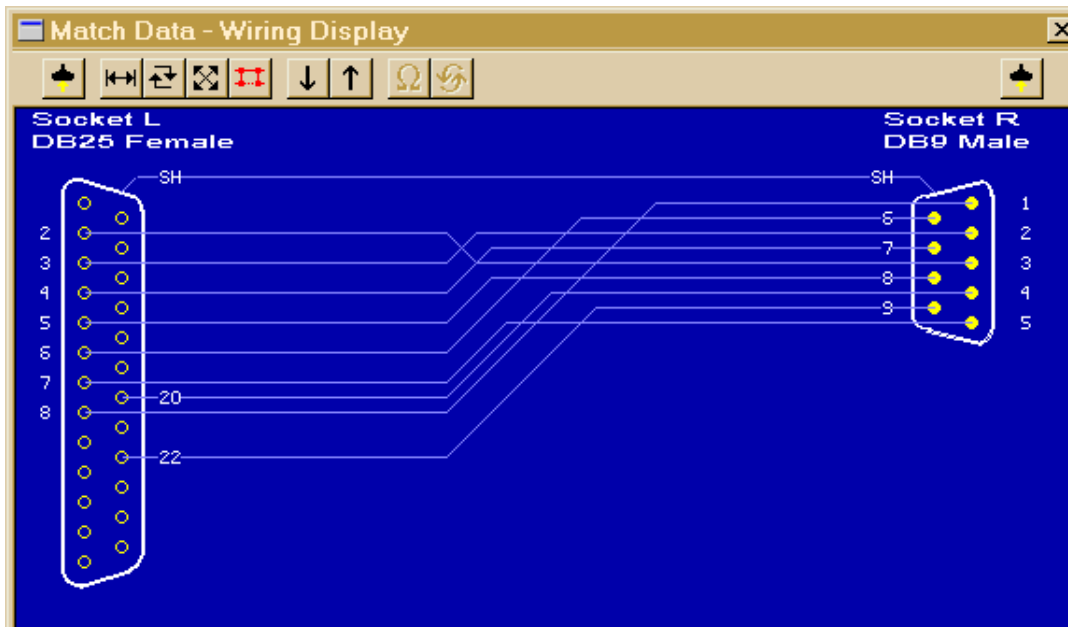


80K



*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 9-pin **DCE**  
(modem)



Cable image created by CableEye®

### 5 - DB25 to DB9 Adapter (pin 1 connected to shield)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

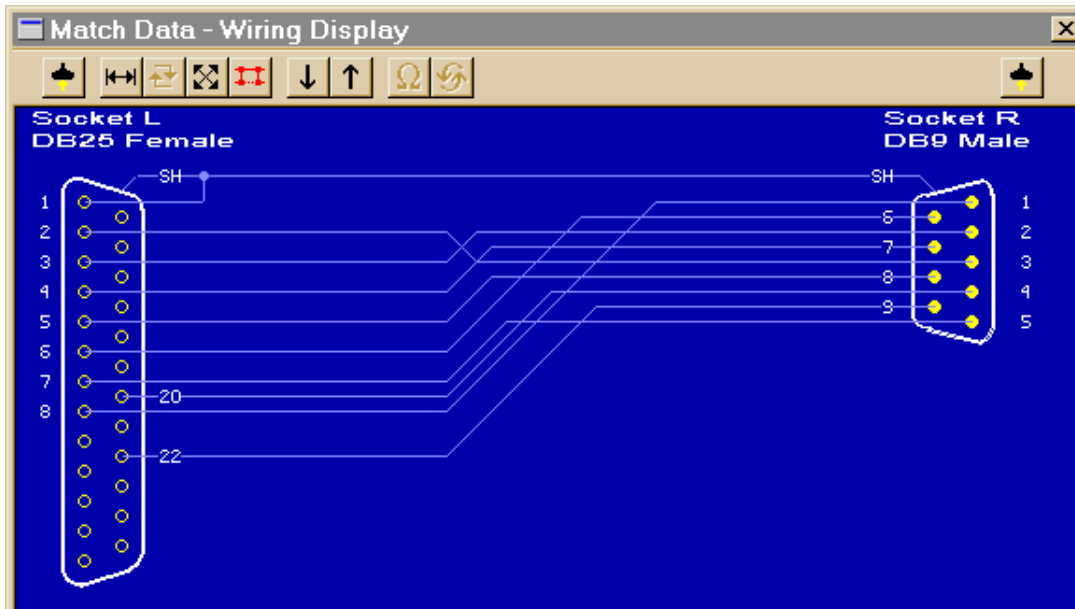
This adapter has the same wiring as the previous cable (#4) except that pin 1 is wired to the connector shell (shield). Note that the cable's shield is usually a foil blanket surrounding all conductors running the length of the cable and joining the connector shells. Pin 1 of the EIA232 specification, called out as "shield", may be separate from the earth ground usually associated with the connector shells.



84K

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 9-pin **DCE**  
(modem)



Cable image created by CableEye®

## 6 - DB9 to DB25 Adapter

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

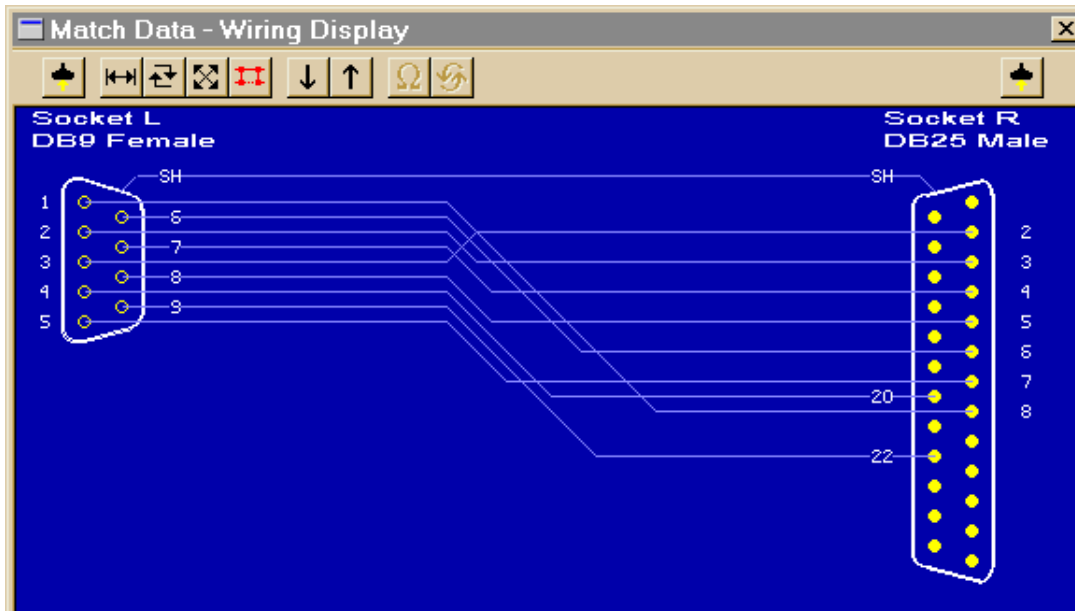
Signals on the DB9 DTE side are directly mapped to the DB25 assignments for a DTE device. Use this to adapt a 9-pin COM connector on the back of a computer to mate with a 25-pin serial DCE devices, such as a modem. This adapter may also be in the form of a cable.



80K

*Left Side:* Connect to 9-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DCE**  
(modem)



Cable image created by CableEye®

## 7 - DB25 All-Line Direct Extension

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

This shows a 25-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 25 pins plus shield are directly extended from DB25 Female to DB25 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any serial device that uses a DB25 connector to a PC's serial port.



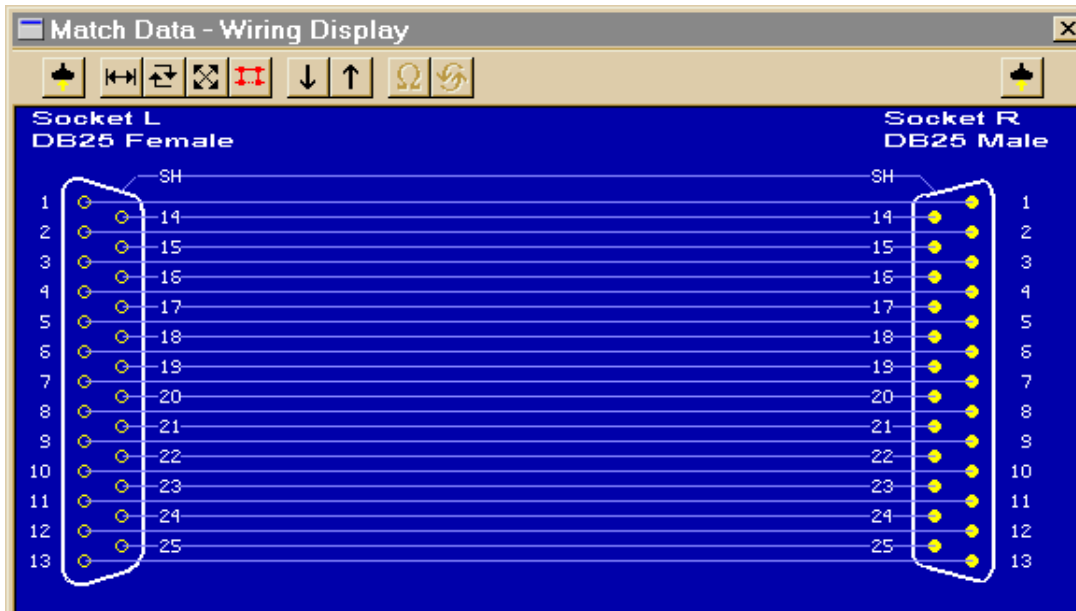
84K

This cable may also serve as an extension cable to increase the distance between computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

*Caution:* the male end of this cable (right) also fits a PC's parallel printer port. You may use this cable to extend the length of a printer cable, but **DO NOT** attach a serial device to the computer's parallel port. Doing so may cause damage to both devices.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DCE**  
(modem)



Cable image created by CableEye®

## 8 - DB25 Loopback Connector

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB25 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable to verify the cable.

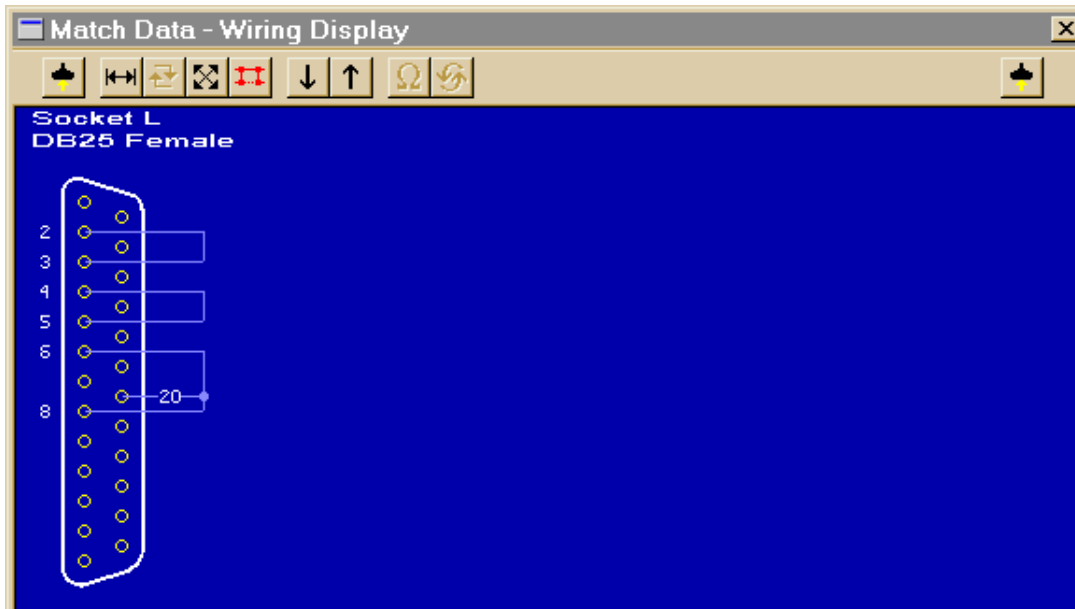


80K

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

*Left Side:* Connect to 25-pin **DTE** (computer)

*Right Side:* (none)



Cable image created by CableEye®

### 9 - DB25 Null Modem (no handshaking)

Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.



84K

Note that Pins 11 and 12 are not necessary for this null modem cable to work. As is often the case, the manufacturer of equipment that uses this cable had a proprietary application in mind. We show it here to emphasize that custom serial cables may include connections for which no purpose is clear.

**IMPORTANT:** This cable employs **NO** handshaking lines between devices. The handshake signals on each side are artificially made to appear asserted by the use of self-connects on each side of the cable (for example, between pins 4 and 5). Without hardware handshaking, you risk buffer overflow at one or both ends of the transmission unless STX and ETX commands are inserted in the dataflow by software.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

### 10 - DB25 Null Modem (standard handshaking)

Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.



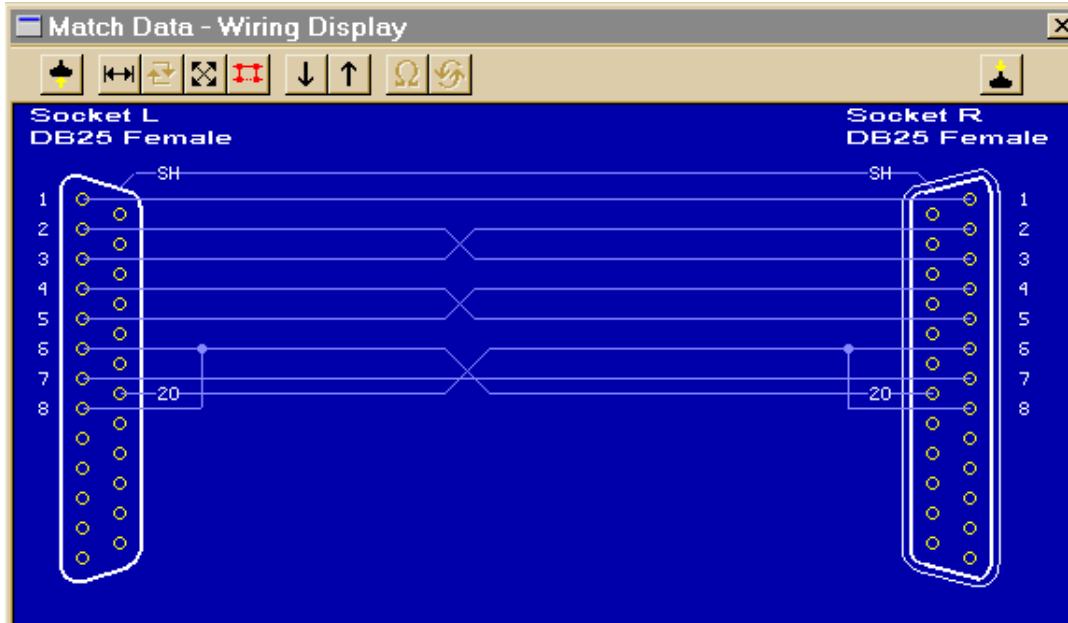
84K

The cable shown below is intended for EIA232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals not shown here.

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6) and the Request to Send (pin 5) on the other side.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

### 11 - DB25 Null Modem (unusual handshaking)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

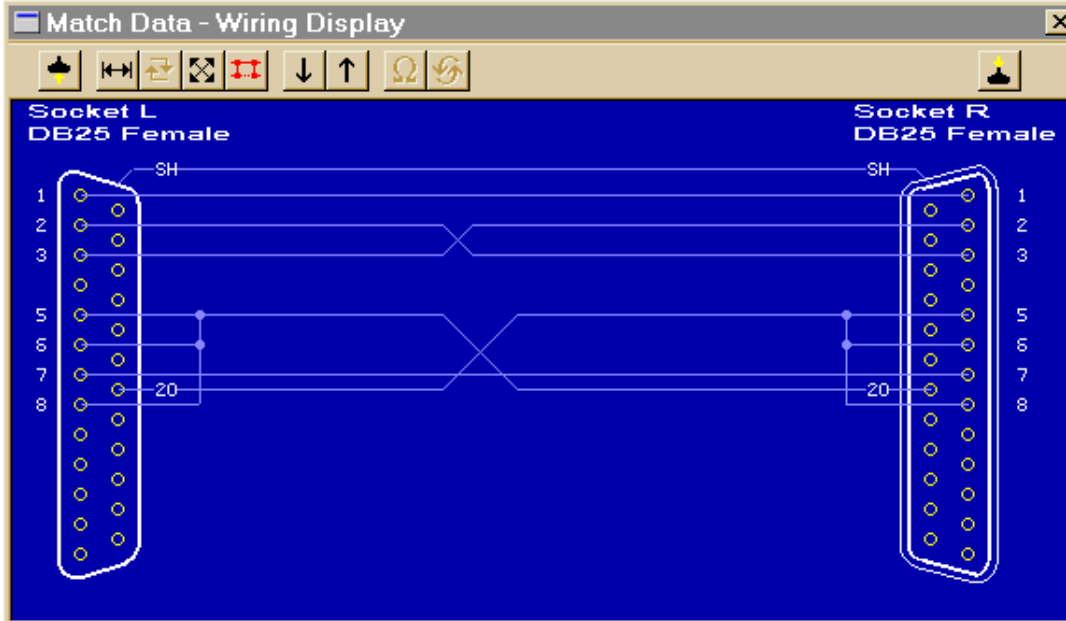


84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear to Send (pin 5), DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

## 12 - DB25 Null Modem (unusual handshaking)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.



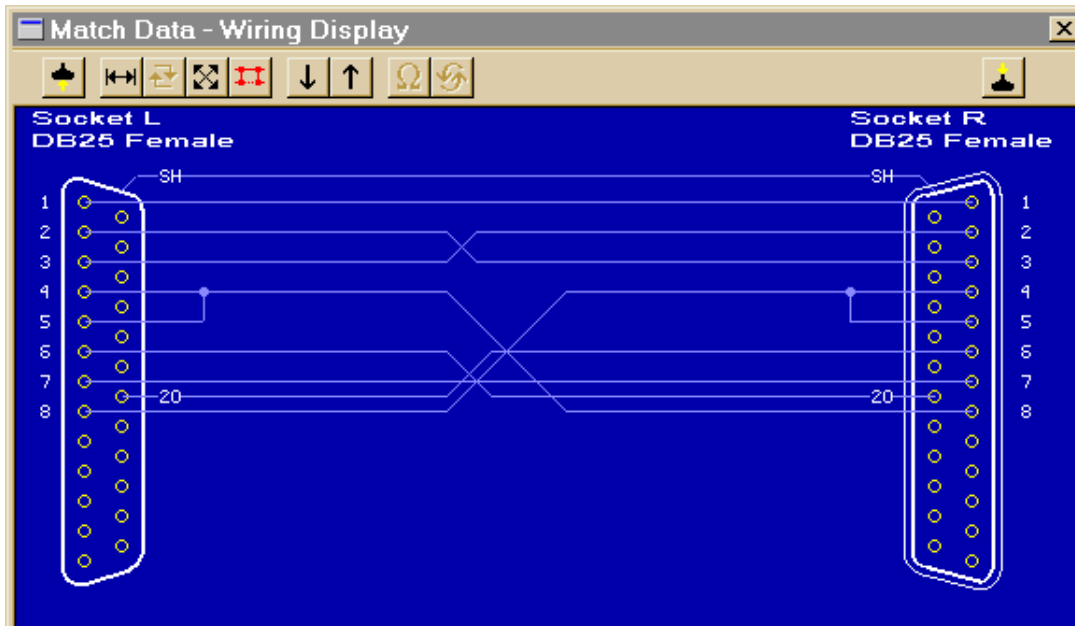
84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the Request-to-Send (pin 4) on one side asserts the Clear-to-Send (pin 5) on the SAME side (self-connect) and the Carrier Detect (pin 8) on the other side. The other handshaking signals are employed in a conventional manner.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)





Cable image created by CableEye®

### 13 - DB25 Null Modem (unusual handshaking)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

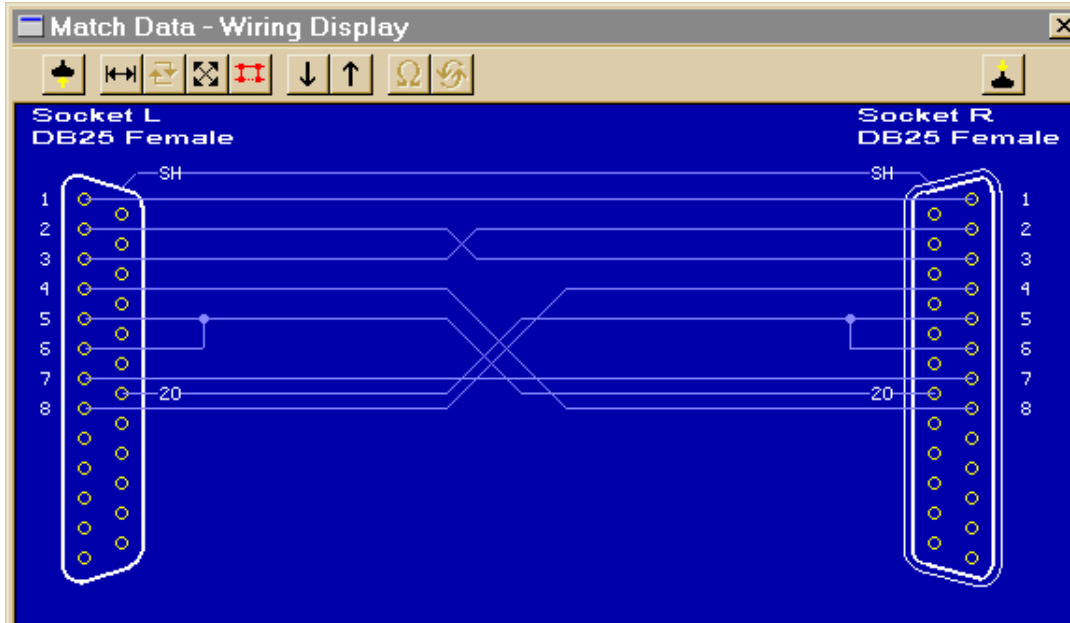


84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear-to-Send (pin 5) and the DCE Ready (pin 6) on the other side. Request-to-Send (pin 4) on one side asserts Received Line Signal Detect (pin 8) on the other side.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

#### 14 - DB25 Null Modem (unusual handshaking)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

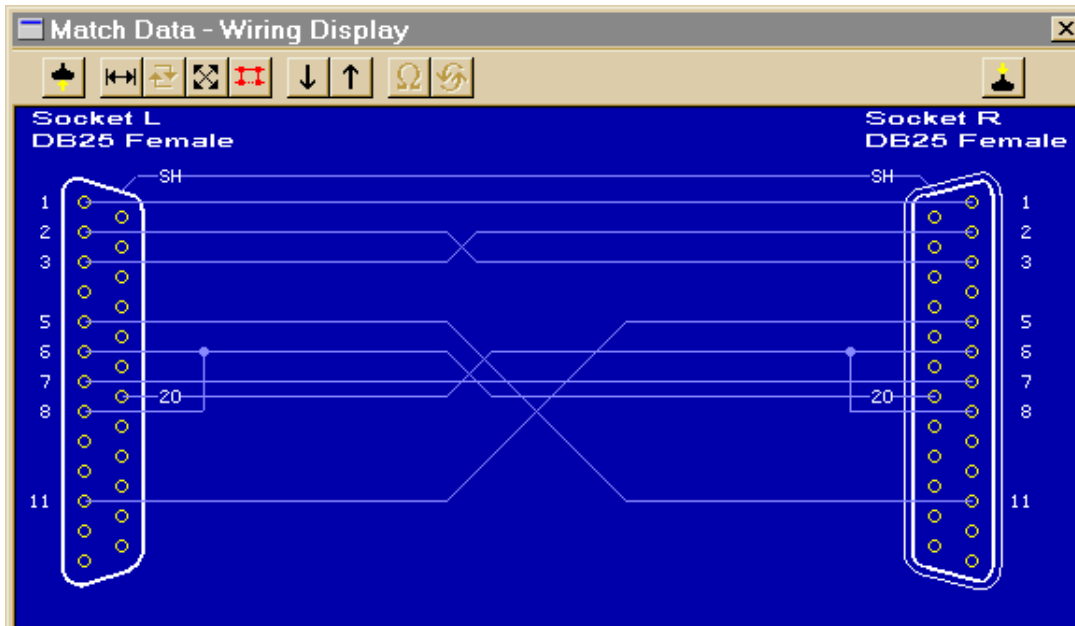


84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side. Request to Send (pin 4) is unused, and Clear-to-Send (pin 5) is driven by a proprietary signal (pin 11) determined by the designer of this cable.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

### 15 - DB25 Null Modem Cable (synchronous communications)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

This female-to-female cable is intended for **synchronous** EIA232 connections, and is designed to connect two DTE devices. It contains the standard connections of an asynchronous null modem cable, plus additional connections on pins 15, 17, and 24 for synchronous timing signals. To connect two DCE devices, use a male-to-male equivalent of this cable.

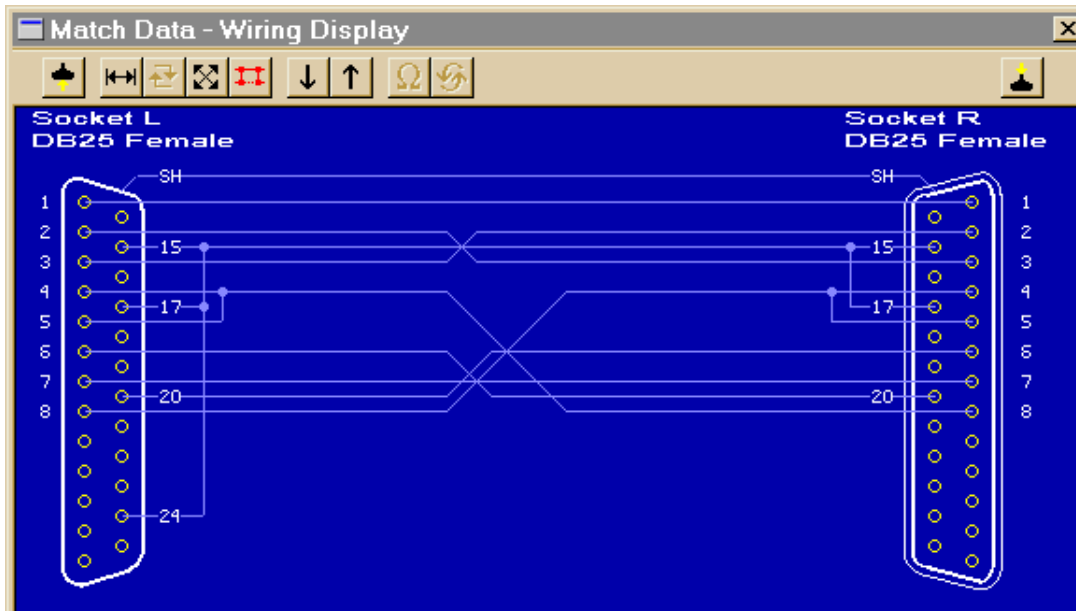


84K

For synchronous communications, the null modem cable includes an additional conductor for timing signals, and joins pins 15, 17, and 24 on one side to pins 15 and 17 on the other. Pin 24 on the right side should connect to the timing signal source.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

**16 - DB25 Null Modem Cable** (unconventional, may pose risk)  
 (no more) | Previous Cable || Next Topic

This simplified null modem cable uses only Request-to-Send (pin 4) and Clear-to-Send (pin 5) as handshaking lines; DTE Ready, DCE Ready, and Carrier Detect are not employed, so this cable should not be used with modems.

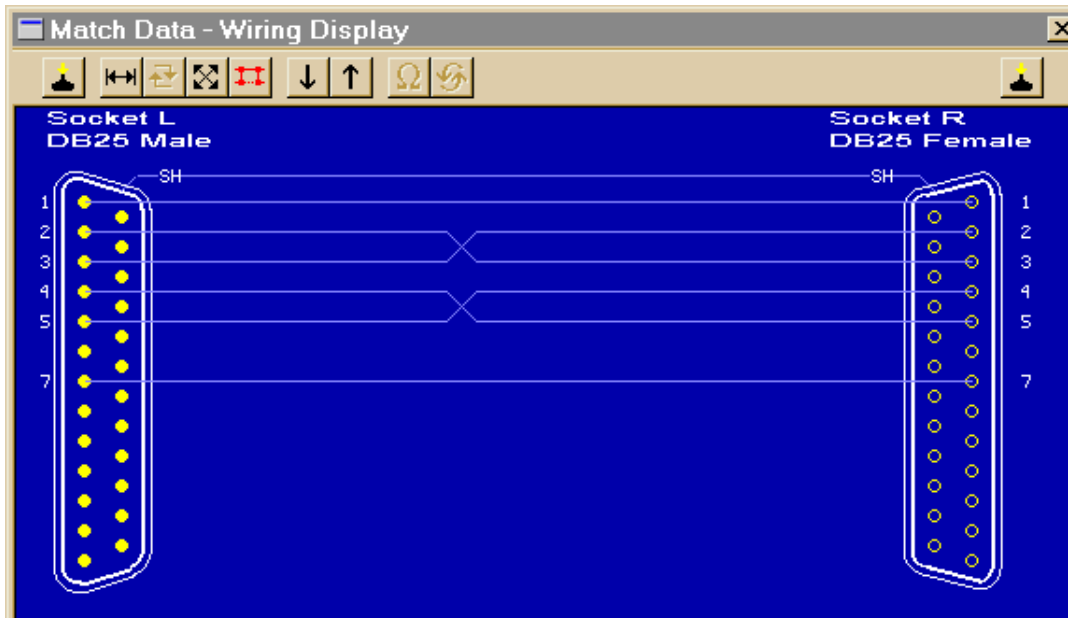


80K

**CAUTION!** Normally, null modem cables have the same gender on each connector (either both male for two DTE devices, or both female for two DCE devices). This cable would be used when the gender on one of the devices does not conform to the standard. However, the opposite genders imply usage as a straight through cable, and if used in that manner will not function. Further, if used as a standard null-modem between two computers, the opposite gender allows you to connect one end to the parallel port, an impermissible situation that may cause hardware damage.

*Left Side:* Connect to 25-pin **DTE** (computer) with Gender Changer

*Right Side:* Connect to 25-pin **DTE** (computer)



Cable image created by CableEye®

---

## Signal Definitions

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

Signal functions in the EIA232 standard can be subdivided into six categories. These categories are summarized below, after which each signal described.

1 - Signal ground and shield.

2 - Primary communications channel. This is used for data interchange, and includes flow control signals.

3 - Secondary communications channel. When implemented, this is used for control of the remote modem, requests for retransmission when errors occur, and governance over the setup of the primary channel.

4 - Modem status and control signals. These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.

5 - Transmitter and receiver timing signals. If a synchronous protocol is used, these signals provide timing information for the transmitter and receiver, which may operate at different baud rates.

6 - Channel test signals. Before data is exchanged, the channel may be tested for its integrity, and the baud rate automatically adjusted to the maximum rate that the channel can support.

---

## Signal Ground and Shield

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 7, Pin 1**, and the **shell** are included in this category. Cables provide separate paths for each, but internal wiring often connects pin 1 and the cable shell/shield to signal ground on pin 7.

**Pin 7 - Ground** All signals are referenced to a common ground, as defined by the voltage on pin 7. This conductor may or may not be connected to protective ground inside the DCE device. The existence of a defined ground potential within the cable makes the EIA232 standard different from a balanced differential voltage standard, such as EIA530, which provides far greater noise immunity.

---

## Primary Communications Channel

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 2 - Transmitted Data (TxD)** This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic '1', negative voltage).

*NOTE:* Pin 2 on the DCE device is commonly labeled "Received Data", although by the EIA232 standard it should still be called Transmitted Data because the data is thought to be destined for a remote DTE device.

**Pin 3 - Received Data (RxD)** This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic '1', negative voltage).

*NOTE:* Pin 3 on the DCE device is commonly labeled "Transmitted Data", although by the EIA232 standard it should still be called Received Data because the data is thought to arrive from a remote DTE device.

**Pin 4 - Request to Send (RTS)** This signal is asserted (logic '0', positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device. Such preparation might include enabling the receive circuits, or setting up the channel direction in half-duplex applications. When the DCE is ready, it acknowledges by asserting Clear to Send.

*NOTE:* Pin 4 on the DCE device is commonly labeled "Clear to Send", although by the EIA232 standard it should still be called Request to Send because the request is thought to be destined for a remote DTE device.

**Pin 5 - Clear to Send (CTS)** This signal is asserted (logic '0', positive voltage) by the DCE device to inform the DTE device that transmission may begin. RTS and CTS are commonly used as handshaking signals to moderate the flow of data into the DCE device.

*NOTE:* Pin 5 on the DCE device is commonly labeled "Request to Send", although by the

EIA232 standard it should still be called Clear to Send because the signal is thought to originate from a remote DTE device.

---

## Secondary Communications Channel

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 14 - Secondary Transmitted Data (STxD)**

**Pin 16 - Secondary Received Data (SRxD)**

**Pin 19 - Secondary Request to Send (SRTS)**

**Pin 13 - Secondary Clear to Send (SCTS)**

These signals are equivalent to the corresponding signals in the primary communications channel. The baud rate, however, is typically much slower in the secondary channel for increased reliability.

---

## Modem Status and Control Signals

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 6 - DCE Ready (DSR)** When originating from a modem, this signal is asserted (logic '0', positive voltage) when the following three conditions are all satisfied:

- 1 - The modem is connected to an active telephone line that is "off-hook";
- 2 - The modem is in data mode, not voice or dialing mode; and
- 3 - The modem has completed dialing or call setup functions and is generating an answer tone.

If the line goes "off-hook", a fault condition is detected, or a voice connection is established, the DCE Ready signal is deasserted (logic '1', negative voltage).

*IMPORTANT:* If DCE Ready originates from a device other than a modem, it may be asserted to indicate that the device is turned on and ready to function, or it may not be used at all. If unused, DCE Ready should be permanently asserted (logic '0', positive voltage) within the DCE device or by use of a self-connect jumper in the cable. Alternatively, the DTE device may be programmed to ignore this signal.

**Pin 20 - DTE Ready (DTR)** This signal is asserted (logic '0', positive voltage) by the DTE device when it wishes to open a communications channel. If the DCE device is a modem, the assertion of DTE Ready prepares the modem to be connected to the telephone circuit, and, once connected, maintains the

connection. When DTE Ready is deasserted (logic '1', negative voltage), the modem is switched to "on-hook" to terminate the connection.

*IMPORTANT:* If the DCE device is not a modem, it may require DTE Ready to be asserted before the device can be used, or it may ignore DTE Ready altogether. If the DCE device (for example, a printer) is not responding, confirm that DTE Ready is asserted before you search for other explanations.

**Pin 8 - Received Line Signal Detector (CD)** (also called carrier detect) This signal is relevant when the DCE device is a modem. It is asserted (logic '0', positive voltage) by the modem when the telephone line is "off-hook", a connection has been established, and an answer tone is being received from the remote modem. The signal is deasserted when no answer tone is being received, or when the answer tone is of inadequate quality to meet the local modem's requirements (perhaps due to a noisy channel).

**Pin 12 - Secondary Received Line Signal Detector (SCD)** This signal is equivalent to the Received Line Signal Detector (pin 8), but refers to the secondary channel.

**Pin 22 - Ring Indicator (RI)** This signal is relevant when the DCE device is a modem, and is asserted (logic '0', positive voltage) when a ringing signal is being received from the telephone line. The assertion time of this signal will approximately equal the duration of the ring signal, and it will be deasserted between rings or when no ringing is present.

**Pin 23 - Data Signal Rate Selector** This signal may originate either in the DTE or DCE devices (but not both), and is used to select one of two prearranged baud rates. The asserted condition (logic '0', positive voltage) selects the higher baud rate.

---

## Transmitter and Receiver Timing Signals

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 15 - Transmitter Signal Element Timing (TC)** (also called Transmitter Clock) This signal is relevant only when the DCE device is a modem and is operating with a synchronous protocol. The modem generates this clock signal to control exactly the rate at which data is sent on Transmitted Data (pin 2) from the DTE device to the DCE device. The logic '1' to logic '0' (negative voltage to positive voltage) transition on this line causes a corresponding transition to the next data element on the Transmitted Data line. The modem generates this signal continuously, except when it is performing internal diagnostic functions.

**Pin 17 - Receiver Signal Element Timing (RC)** (also called Receiver Clock) This signal is similar to TC described above, except that it provides timing information for the DTE receiver.

**Pin 24 - Transmitter Signal Element Timing (ETC)** (also called External Transmitter Clock) Timing signals are provided by the DTE device for use by a modem. This signal is used only when TC and RC (pins 15 and 17) are not in use. The logic '1' to logic '0' transition (negative voltage to positive voltage) indicates the time-center of the data element. Timing signals will be provided whenever the DTE is turned on, regardless of other signal conditions.



---

## Channel Test Signals

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 18 - Local Loopback (LL)** This signal is generated by the DTE device and is used to place the modem into a test state. When Local Loopback is asserted (logic '0', positive voltage), the modem redirects its modulated output signal, which is normally fed into the telephone line, back into its receive circuitry. This enables data generated by the DTE to be echoed back through the local modem to check the condition of the modem circuitry. The modem asserts its Test Mode signal on Pin 25 to acknowledge that it has been placed in local loopback condition.

**Pin 21 - Remote Loopback (RL)** This signal is generated by the DTE device and is used to place the remote modem into a test state. When Remote Loopback is asserted (logic '0', positive voltage), the remote modem redirects its received data back to its transmitted data input, thereby remodulating the received data and returning it to its source. When the DTE initiates such a test, transmitted data is passed through the local modem, the telephone line, the remote modem, and back, to exercise the channel and confirm its integrity. The remote modem signals the local modem to assert Test Mode on pin 25 when the remote loopback test is underway.

**Pin 25 - Test Mode (TM)** This signal is relevant only when the DCE device is a modem. When asserted (logic '0', positive voltage), it indicates that the modem is in a Local Loopback or Remote Loopback condition. Other internal self-test conditions may also cause Test Mode to be asserted, and depend on the modem and the network to which it is attached.

---

## Electrical Standards

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

The EIA232 standard uses negative, bipolar logic in which a negative voltage signal represents logic '1', and positive voltage represents logic '0'. This probably originated with the pre-RS232 current loop standard used in 1950s-vintage teletype machines in which a flowing current (and hence a low voltage) represents logic '1'. Be aware that the negative logic assignment of EIA232 is the reverse of that found in most modern digital circuit designs. See the inside rear cover of the CableEye manual for a comparison.

---

## Common Signal Ground

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

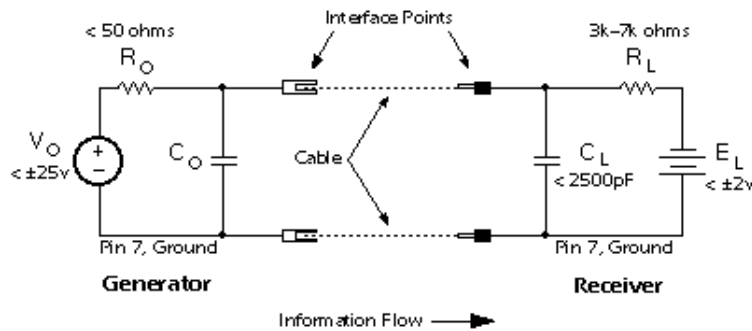
The EIA232 standard includes a common ground reference on Pin 7, and is frequently joined to Pin 1 and a circular shield that surrounds all 25 cable conductors. Data, timing, and control signal voltages are measured with respect to this common ground. EIA232 cannot be used in applications where the equipment on opposite ends of the connection must be electrically isolated.

*NOTE:* optical isolators may be used to achieve ground isolation, however, this option is not mentioned or included in the EIA232 specification.

## Signal Characteristics

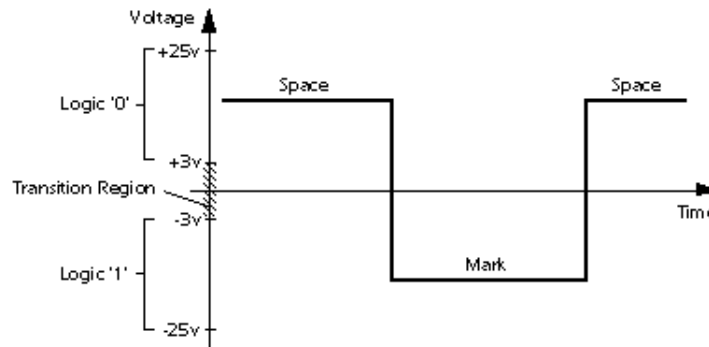
[Next Topic](#) | [Previous Topic](#) | [TOC](#)

Equivalent Circuit - All signal lines, regardless of whether they provide data, timing, or control information, may be represented by the electrical equivalent circuit shown here:



*This is the equivalent circuit for an EIA232 signal line and applies to signals originating at either the DTE or DCE side of the connection. "C<sub>o</sub>" is not specified in the standard, but is assumed to be small and to consist of parasitic elements only. "R<sub>o</sub>" and "V<sub>o</sub>" are chosen so that the short-circuit current does not exceed 500ma. The cable length is not specified in the standard; acceptable operation is experienced with cables that are less than 25 feet in length.*

**Signal State Voltage Assignments** - Voltages of -3v to -25v with respect to signal ground (pin 7) are considered logic '1' (the marking condition), whereas voltages of +3v to +25v are considered logic '0' (the spacing condition). The range of voltages between -3v and +3v is considered a transition region for which a signal state is not assigned.



*Logic states are assigned to the voltage ranges shown here. Note that this is a "negative logic" convention, which is the reverse of that used in most modern*

## *digital designs.*

Most contemporary applications will show an open-circuit signal voltage of -8 to -14 volts for logic '1' (mark), and +8 to +14 volts for logic '0' (space). Voltage magnitudes will be slightly less when the generator and receiver are connected (when the DTE and DCE devices are connected with a cable).

*IMPORTANT:* If you insert an LED signal tester in an EIA232 circuit to view signal states, the signal voltage may drop in magnitude to very near the minimum values of -3v for logic '1', and +3v for logic '0'. Also note that some inexpensive EIA232 peripherals are powered directly from the signal lines to avoid using a power supply of their own. Although this usually works without problems, keep the cable short, and be aware that noise immunity will be reduced.

**Short-Circuit Tolerance** - The generator is designed to withstand an open-circuit (unconnected) condition, or short-circuit condition between its signal conductor and any other signal conductor, including ground, without sustaining damage to itself or causing damage to any associated circuitry. The receiver is also designed to accept any signal voltage within the range of  $\pm 25$  volts without sustaining damage.

*CAUTION:* Inductive loads or magnetically induced voltages resulting from long cables may cause the received voltage to exceed the  $\pm 25$ -volt range momentarily during turn-on transients or other abnormal conditions, possibly causing damage to the generator, receiver, or both. Keep the cable length as short as possible, and avoid running the cable near high-current switching loads like electric motors or relays.

**Fail-Safe Signals** - Four signals are intended to be fail-safe in that during power-off or cable-disconnected conditions, they default to logic '1' (negative voltage). They are:

Request to Send - Default condition is deasserted.

Sec. Request to Send - Default condition is deasserted.

DTE Ready - Default condition is DTE not ready.

DCE Ready - Default condition is DCE not ready.

Note specifically that if the cable is connected but the power is off in the generator side, or if the cable is disconnected, there should be adequate bias voltage in the receiver to keep the signal above +3v (logic '0') to ensure that the fail-safe requirement is met.

Schmitt triggers or other hysteresis devices may be used to enhance noise immunity in some designs, but should never be adjusted to compromise the fail-safe requirement.

---

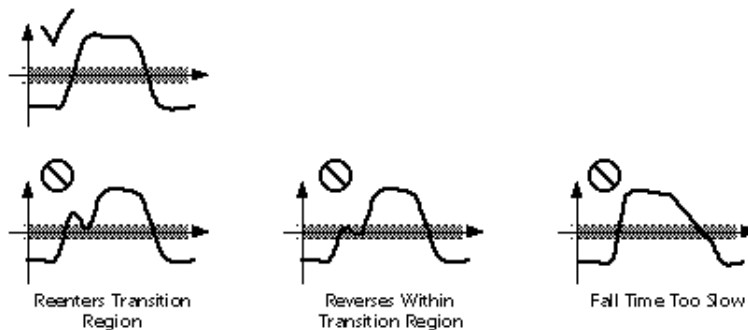
## **Signal Timing**

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

The EIA232 standard is applicable to data rates of up to 20,000 bits per second (the usual upper limit is 19,200 baud). Fixed baud rates are not set by the EIA232 standard. However, the commonly used values are 300, 1200, 2400, 9600, and 19,200 baud. Other accepted values that are not often used are 110 (mechanical teletype machines), 600, and 4800 baud.

Changes in signal state from logic '1' to logic '0' or vice versa must abide by several requirements, as follows:

- 1 - Signals that enter the transition region during a change of state must move through the transition region to the opposite signal state without reversing direction or reentering.
- 2 - For control signals, the transit time through the transition region should be less than 1ms.
- 3 - For Data and Timing signals, the transit time through the transition region should be
  - a - less than 1ms for bit periods greater than 25ms,
  - b - 4% of the bit period for bit periods between 25ms and 125 $\mu$ s,
  - c - less than 5 $\mu$ s for bit periods less than 125 $\mu$ s.The rise and fall times of data and timing signals ideally should be equal, but in any case vary by no more than a factor of three.



*An acceptable pulse (top) moves through the transition region quickly and without hesitation or reversal. Defective pulses (bottom) could cause data errors.*

- 
- 4 - The slope of the rising and falling edges of a transition should not exceed 30v/ $\mu$ S. Rates higher than this may induce crosstalk in adjacent conductors of a cable.

Note that neither the ASCII alphabet nor the asynchronous serial protocol that defines the start bit, number of data bits, parity bit, and stop bit, is part of the EIA232 specification. For your reference, it is discussed in the Data Communications Basics section of this web site.

---

## **Accepted Simplifications of the Standard**

[Previous Topic](#) | [TOC](#)

The EIA232 document published by the Electronic Industries Association describes 14 permissible configurations of the original 22-signal standard. Each configuration uses a subset of the 22 defined signals, and serves a more limited communications requirement than that suggested by using all the available 22-signals. Applications for transmit-only, receive-only, half-duplex operation, and similar variations, are described. Unfortunately, connection to DCE devices other than modems is not considered. Because many current serial interface applications involve direct device-to-device connections, manufacturers do not have a standard reference when producing printers, plotters, print spoolers, or other common peripherals. Consequently, you must acquire the service manual for each peripheral device purchased to determine exactly which signals are utilized in its serial interface.

END

[Return to TOC](#)

# The RS232 STANDARD

*A Tutorial with Signal Names and Definitions*

(renamed the "EIA232 Standard" in the early 1990's)

Written by Christopher E. Strangio  
Copyright © 1993-2003 by CAMI Research Inc., Lexington, Massachusetts

Send Us Your Comments . . .

## Contents

What is EIA232?  
Likely Problems when Using an EIA232 Interface  
Pin Assignments  
Cable Wiring Examples (New!)  
Signal Definitions  
Signal Ground and Shield  
Primary Communications Channel  
Secondary Communications Channel  
Modem Status and Control Signals  
Transmitter and Receiver Timing Signals  
Channel Test Signals  
Electrical Standards  
Common Signal Ground  
Signal Characteristics  
Signal Timing  
Accepted Simplifications of the Standard

Pin Description Index

References to EIA Publications

[Back to CableEye® Home Page](#)

---

## What is EIA232?

[Next Topic](#) | [TOC](#)

In the early 1960s, a standards committee, today known as the Electronic Industries Association, developed a common interface standard for data communications equipment. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the

interconnection of equipment produced by different manufacturers, thereby fostering the benefits of mass production and competition. From these ideas, the RS232 standard was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

Over the 40+ years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232E standard introduced in 1991. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

---

## **Likely Problems when Using an EIA232 Interface**

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

During this 40-year-long, rapidly evolving period in electronics, manufacturers adopted simplified versions of this interface for applications that were impossible to envision in the 1960s. Today, virtually all contemporary serial interfaces are EIA232-like in their signal voltages, protocols, and connectors, whether or not a modem is involved. Because no single "simplified" standard was agreed upon, however, many slightly different protocols and cables were created that obligingly mate with any EIA232 connector, but are incompatible with each other. Most of the difficulties you will encounter in EIA232 interfacing include at least one of the following:

1 - The absence or misconnection of flow control (handshaking) signals, resulting in buffer overflow or communications lock-up.

2 - Incorrect communications function (DTE versus DCE) for the cable in use, resulting in the reversal of the Transmit and Receive data lines as well as one or more handshaking lines.

3 - Incorrect connector gender or pin configuration, preventing cable connectors from mating properly.

Fortunately, EIA232 driver circuitry is highly tolerant of misconnections, and will usually survive a drive signal being connected to ground, or two drive signals connected to each other. In any case, if the serial interface between two devices is not operating correctly, disconnect the cable joining this equipment until the problem is isolated.

---

## **Pin Assignments**

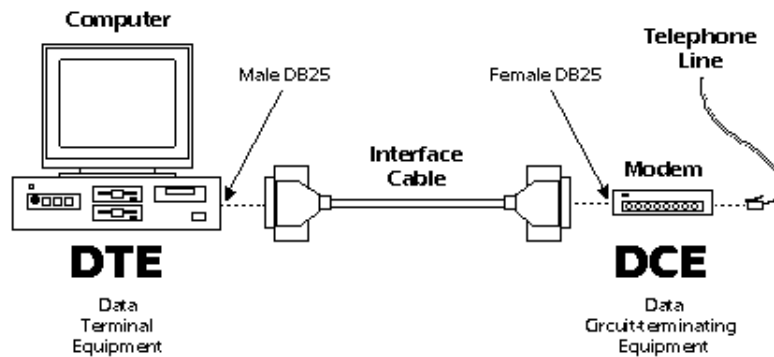
[Next Topic](#) | [Previous Topic](#) | [TOC](#)

Go to DTE Pinout (looking into the computer's serial connector)

Go to DCE Pinout (looking into the modem's serial connector)

If the full EIA232 standard is implemented as defined, the equipment at the far end of the connection is

named the DTE device (Data Terminal Equipment, usually a computer or terminal), has a male DB25 connector, and utilizes 22 of the 25 available pins for signals or ground. Equipment at the near end of the connection (the telephone line interface) is named the DCE device (Data Circuit-terminating Equipment, usually a modem), has a female DB25 connector, and utilizes the same 22 available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable with no cross-overs or self-connects in the connector hoods. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. This drawing shows the orientation and connector types for DTE and DCE devices:



*EIA232 communication function and connector types for a personal computer and modem. DCE devices are sometimes called "Data Communications Equipment" instead of Data Circuit-terminating Equipment.*

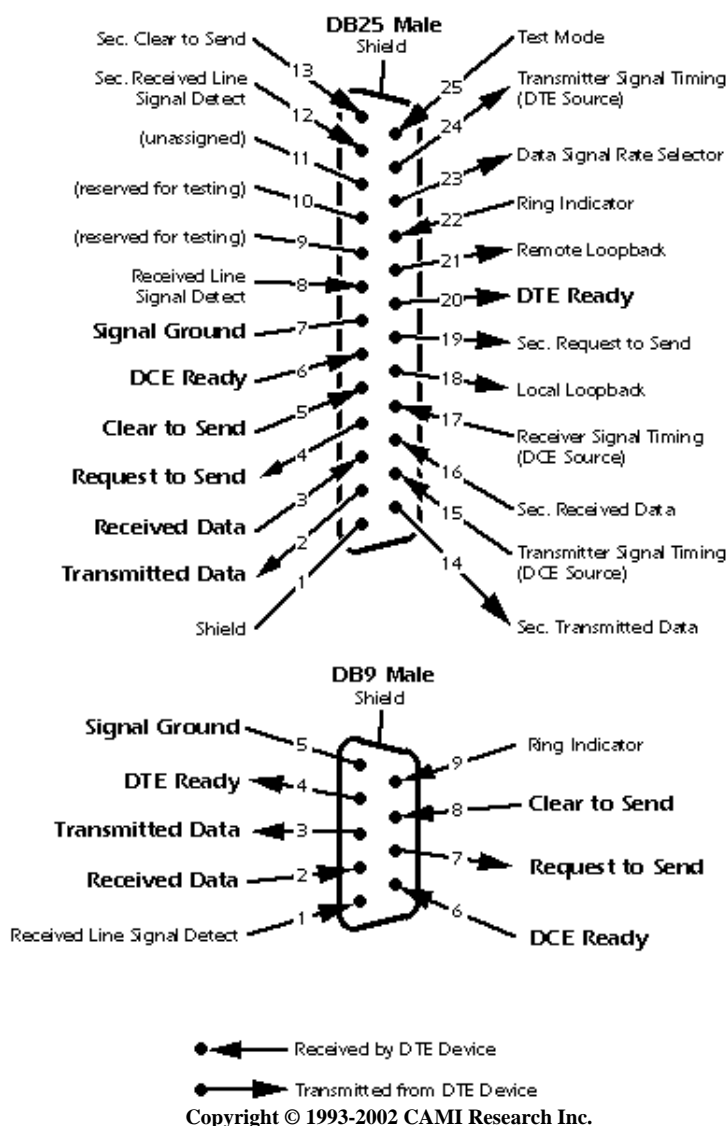
---

Here is the full EIA232 signal definition for the DTE device (usually the PC). The most commonly used signals are shown in bold.

[back to Pin Assignments description]



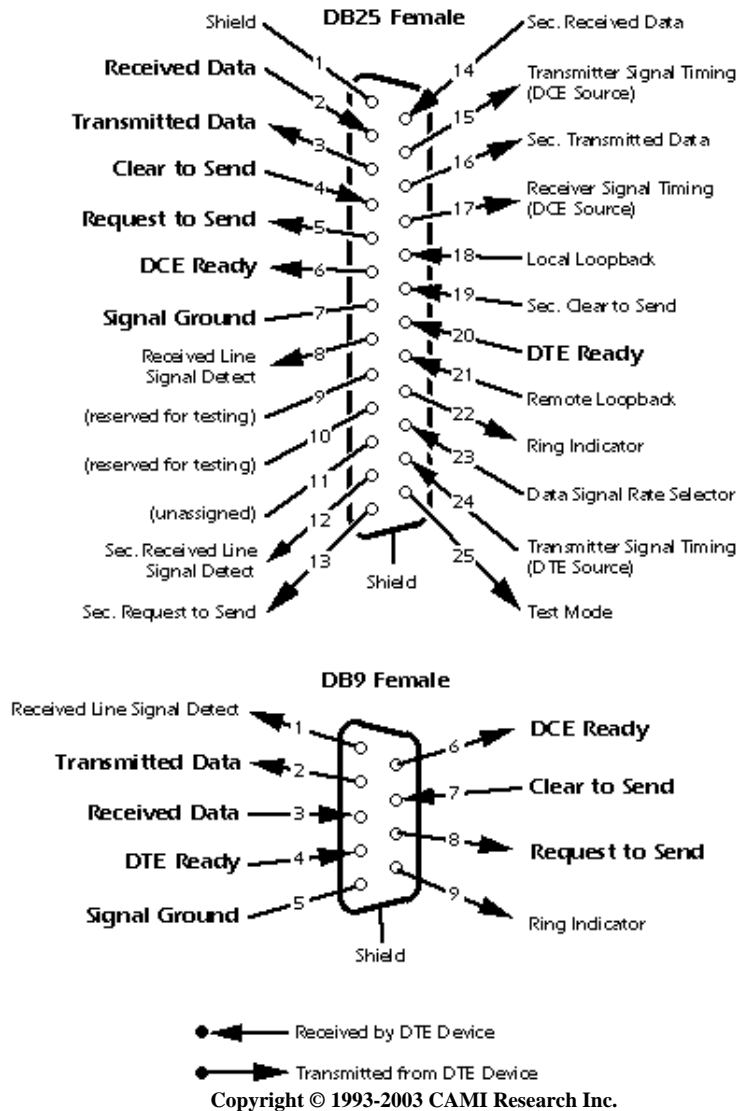
### Looking Into the DTE Device Connector



This shows the full EIA232 signal definition for the DCE device (usually the modem). The most commonly used signals are shown in bold.

[back to Pin Assignments description]

## Looking Into the DCE Device Connector



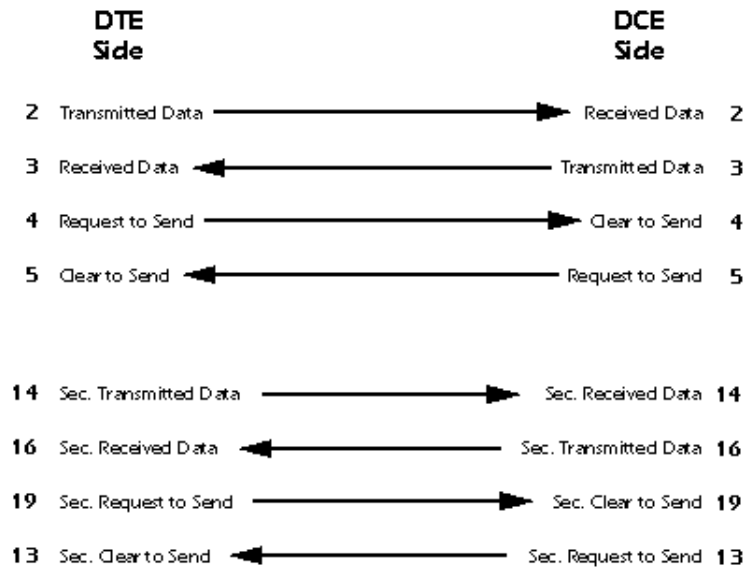
Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, far fewer signal lines are necessary.

You may have noticed in the pinout drawings that there is a secondary channel which includes a duplicate set of flow-control signals. This secondary channel provides for management of the remote modem, enabling baud rates to be changed on the fly, retransmission to be requested if a parity error is detected, and other control functions. This secondary channel, when used, is typically set to operate at a very low baud rate in comparison with the primary channel to ensure reliability in the control path. In addition, it may operate as either a simplex, half-duplex, or full-duplex channel, depending on the capabilities of the modem.

Transmitter and receiver timing signals (pins 15, 17, and 24) are used only for a synchronous

transmission protocol. For the standard asynchronous 8-bit protocol, external timing signals are unnecessary.

**IMPORTANT:** Signal names that imply a direction, such as Transmit Data and Receive Data, are named from the point of view of the DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, this is not done in practice by most engineers, probably because no one can keep straight which side is DTE and which is DCE. As a result, direction-sensitive signal names are changed at the DCE side to reflect their *drive direction* at DCE. The following list gives the conventional usage of signal names:




---

## Cable Wiring Examples

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

The following wiring diagrams come from actual cables scanned by the CableEye® PC-Based Cable Test System. CableEye’s software automatically draws schematics whenever it tests a cable. [Click here](#) to learn more about CableEye.

### 1 - DB9 All-Line Direct Extension

[Next Cable](#) | (no previous cable) || [Next Topic](#)

This shows a 9-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 9 pins plus shield are directly extended from DB9 Female to DB9 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any device that uses a DB9 connector to a PC’s serial port.

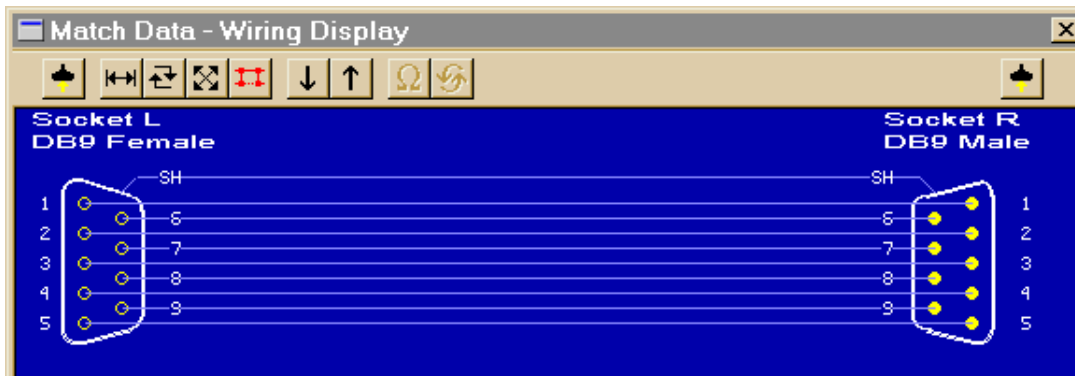


80K

This cable may also serve as an extension cable to increase the distance between a computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

*Left Side:* Connect to **DTE**  
(computer)

*Right Side:* Connect to **DCE** (modem or other  
serial device)



Cable image created by CableEye®

## 2 - DB9 Loopback Connector

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB9 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable to verify the cable.

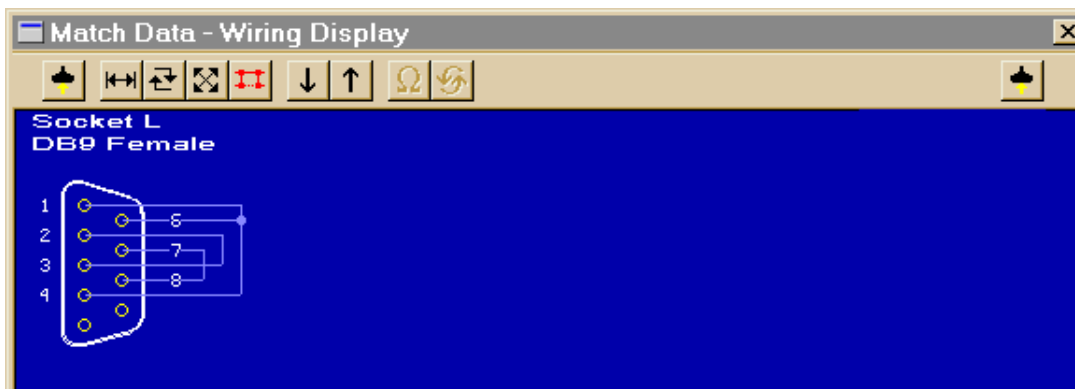


80K

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

*Left Side:* Connect to **DTE** (computer)

*Right Side:* (none)



Cable image created by CableEye®

---

### 3 - DB9 Null Modem Cable

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.



80K

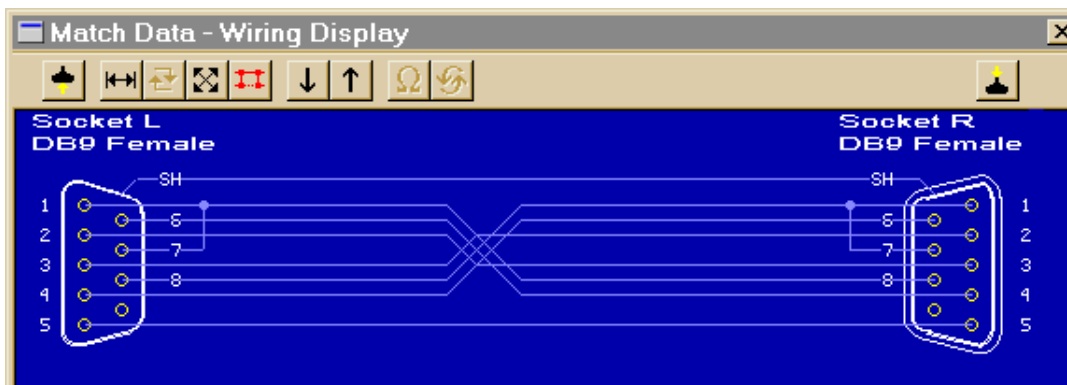
The cable shown below is intended for RS232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals, and a DB25 connector would be necessary.

NOTE: Not all null modem cables connect handshaking lines the same way. In this cable, Request-to-Send (RTS, pin 7) asserts the Carrier Detect (pin 1) on the same side and the Clear-to-Send (CTS, pin 8) on the other side of the cable.

This device may also be available in the form of an adapter.

*Left Side:* Connect to 9-pin **DTE**  
(computer)

*Right Side:* Connect to 9-pin **DTE**  
(computer)



Cable image created by CableEye®

---

### 4 - DB25 to DB9 Adapter

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

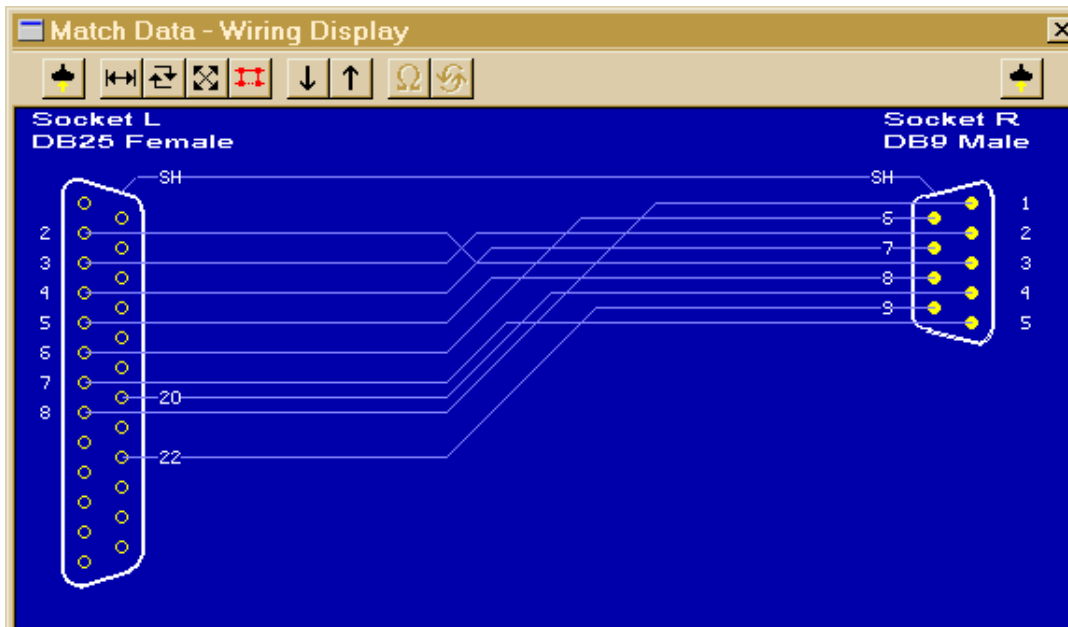
Signals on the DB25 DTE side are directly mapped to the DB9 assignments for a DTE device. Use this to adapt a 25-pin COM connector on the back of a computer to mate with a 9-pin serial DCE device, such as a 9-pin serial mouse or modem. This adapter may also be in the form of a cable.



80K

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 9-pin **DCE**  
(modem)



Cable image created by CableEye®

### 5 - DB25 to DB9 Adapter (pin 1 connected to shield)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

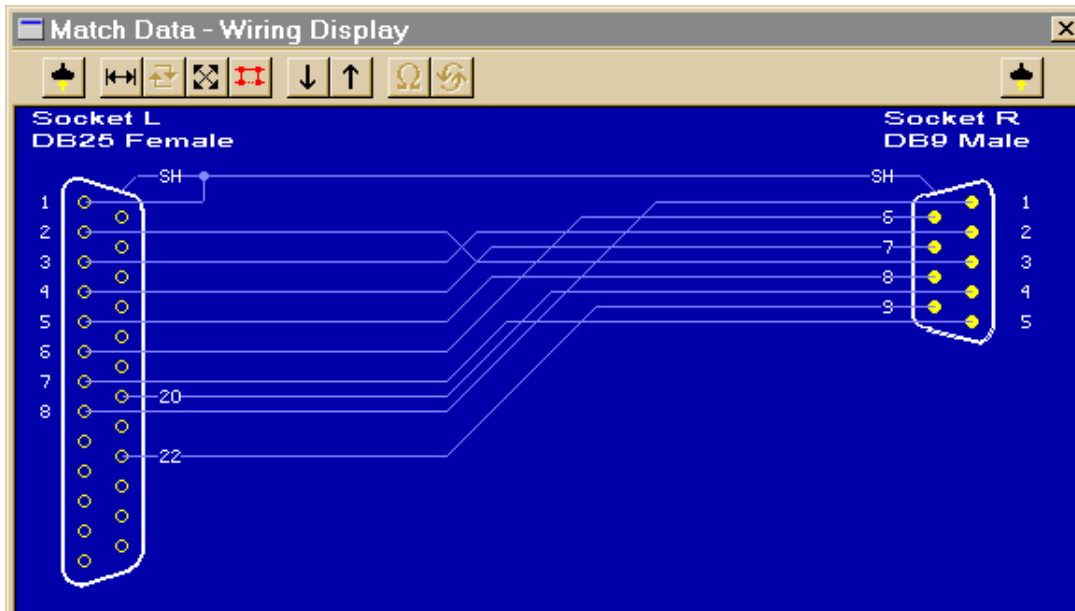
This adapter has the same wiring as the previous cable (#4) except that pin 1 is wired to the connector shell (shield). Note that the cable's shield is usually a foil blanket surrounding all conductors running the length of the cable and joining the connector shells. Pin 1 of the EIA232 specification, called out as "shield", may be separate from the earth ground usually associated with the connector shells.



84K

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 9-pin **DCE**  
(modem)



Cable image created by CableEye®

## 6 - DB9 to DB25 Adapter

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

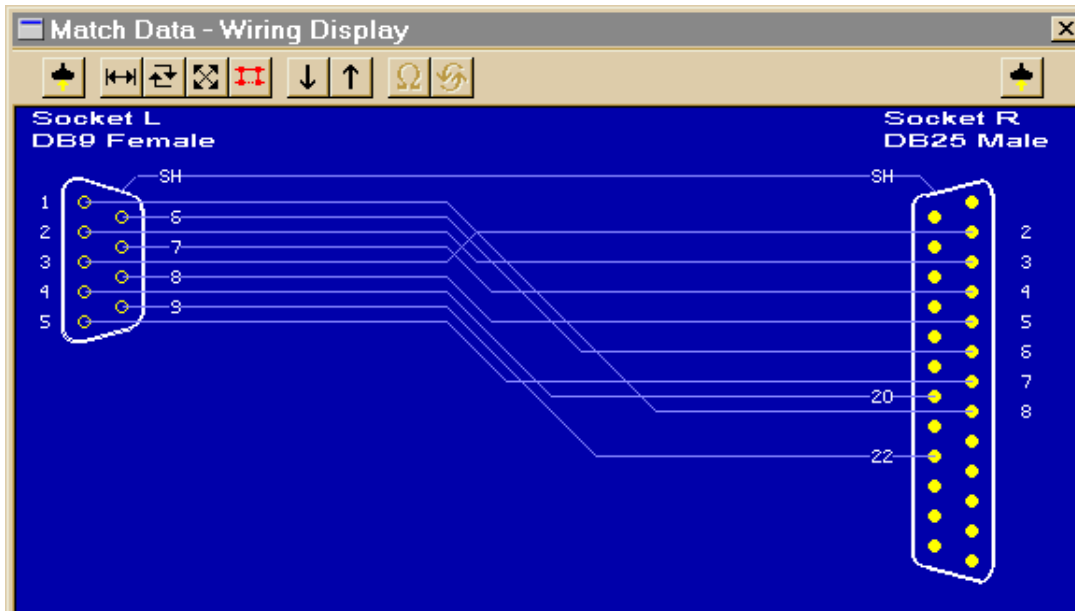
Signals on the DB9 DTE side are directly mapped to the DB25 assignments for a DTE device. Use this to adapt a 9-pin COM connector on the back of a computer to mate with a 25-pin serial DCE devices, such as a modem. This adapter may also be in the form of a cable.



80K

*Left Side:* Connect to 9-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DCE**  
(modem)



Cable image created by CableEye®

## 7 - DB25 All-Line Direct Extension

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

This shows a 25-pin DTE-to-DCE serial cable that would result if the EIA232 standard were strictly followed. All 25 pins plus shield are directly extended from DB25 Female to DB25 Male. There are no crossovers or self-connects present. Use this cable to connect modems, printers, or any serial device that uses a DB25 connector to a PC's serial port.



84K

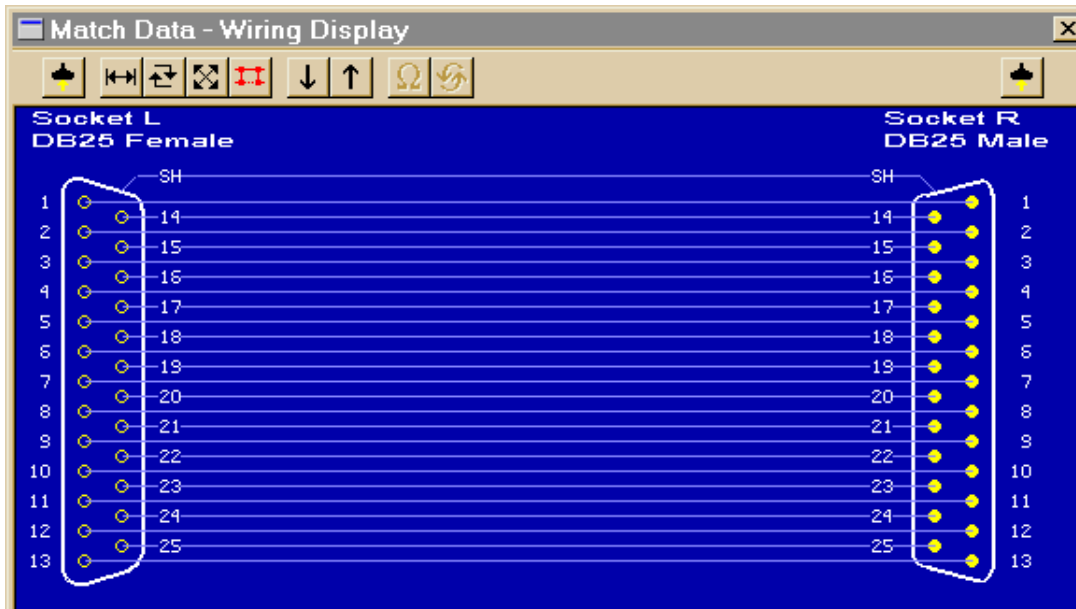
This cable may also serve as an extension cable to increase the distance between computer and serial device. *Caution:* do not exceed 25 feet separation between devices without a signal booster!

*Caution:* the male end of this cable (right) also fits a PC's parallel printer port. You may use this cable to extend the length of a printer cable, but **DO NOT** attach a serial device to the computer's parallel port. Doing so may cause damage to both devices.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DCE**  
(modem)





Cable image created by CableEye®

## 8 - DB25 Loopback Connector

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

A loopback connector usually consists of a connector without a cable and includes internal wiring to reroute signals back to the sender. This DB25 female connector would attach to a DTE device such as a personal computer. When the computer receives data, it will not know whether the signals it receives come from a remote DCE device set to echo characters, or from a loopback connector. Use loopback connectors to confirm proper operation of the computer's serial port. Once confirmed, insert the serial cable you plan to use and attach the loopback to the end of the serial cable to verify the cable.

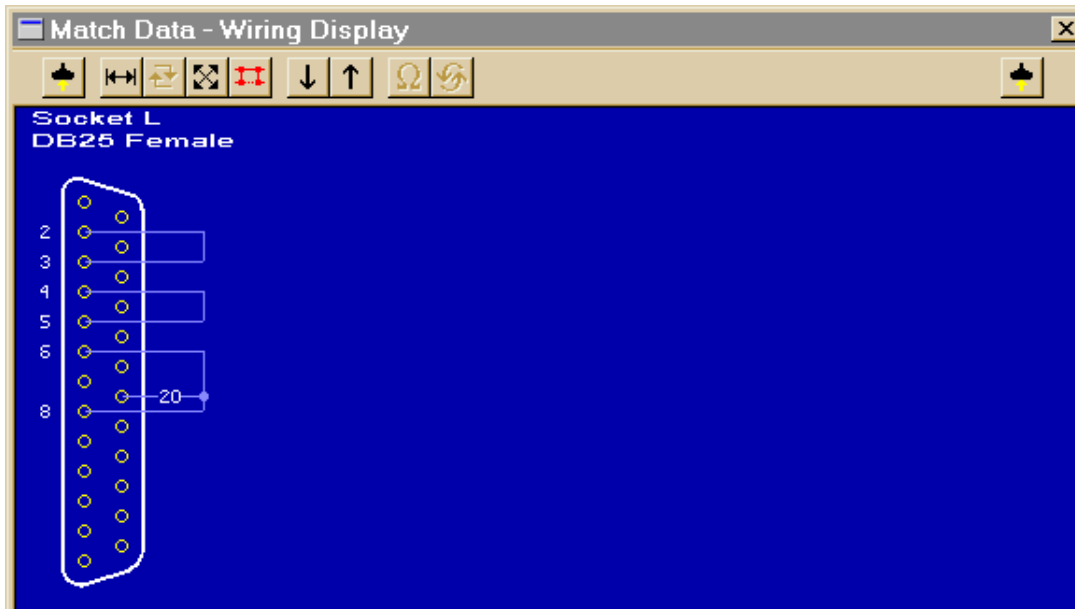


80K

In this case, Transmit Data joins to Received Data, Request-to-Send joins to Clear-to-Send, and DTE-Ready joins to DCE-Ready and Received Line Signal Detect.

*Left Side:* Connect to 25-pin **DTE** (computer)

*Right Side:* (none)



Cable image created by CableEye®

### 9 - DB25 Null Modem (no handshaking)

Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.



84K

Note that Pins 11 and 12 are not necessary for this null modem cable to work. As is often the case, the manufacturer of equipment that uses this cable had a proprietary application in mind. We show it here to emphasize that custom serial cables may include connections for which no purpose is clear.

**IMPORTANT:** This cable employs **NO** handshaking lines between devices. The handshake signals on each side are artificially made to appear asserted by the use of self-connects on each side of the cable (for example, between pins 4 and 5). Without hardware handshaking, you risk buffer overflow at one or both ends of the transmission unless STX and ETX commands are inserted in the dataflow by software.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

### 10 - DB25 Null Modem (standard handshaking)

Next Cable | Previous Cable || Next Topic

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.



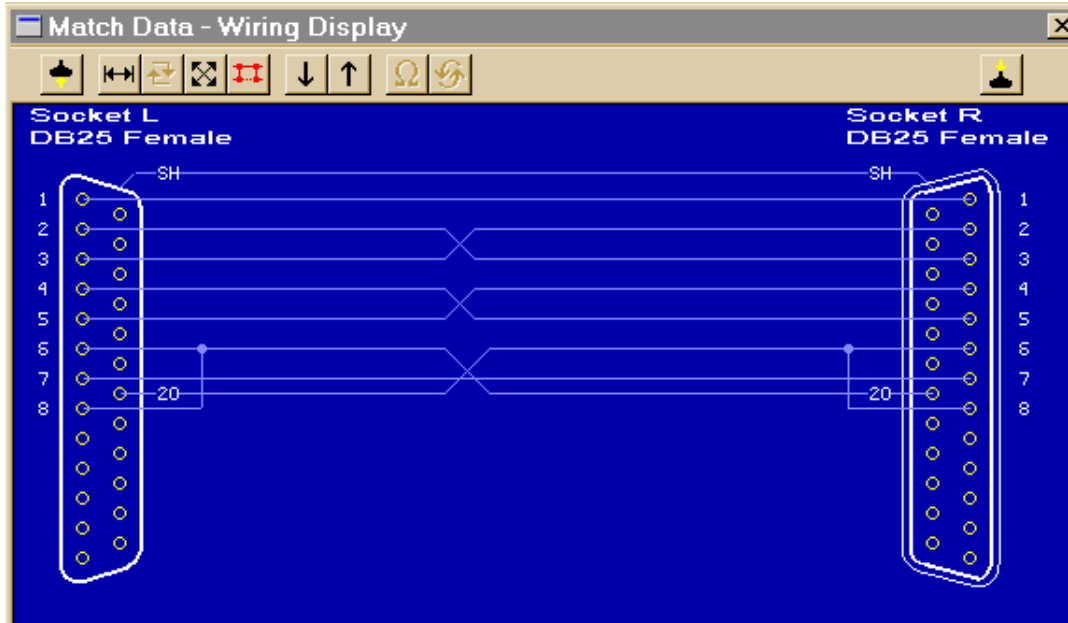
84K

The cable shown below is intended for EIA232 asynchronous communications (most PC-based systems). If you are using synchronous communications, the null modem will have additional connections for timing signals not shown here.

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6) and the Request to Send (pin 5) on the other side.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

### 11 - DB25 Null Modem (unusual handshaking)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

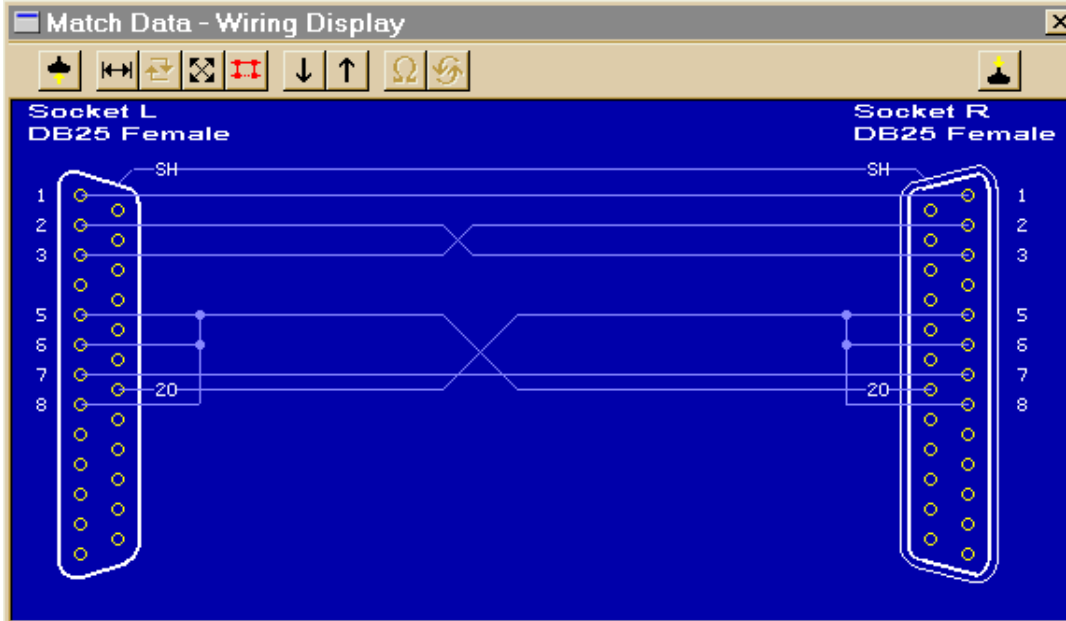


84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear to Send (pin 5), DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

## 12 - DB25 Null Modem (unusual handshaking)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

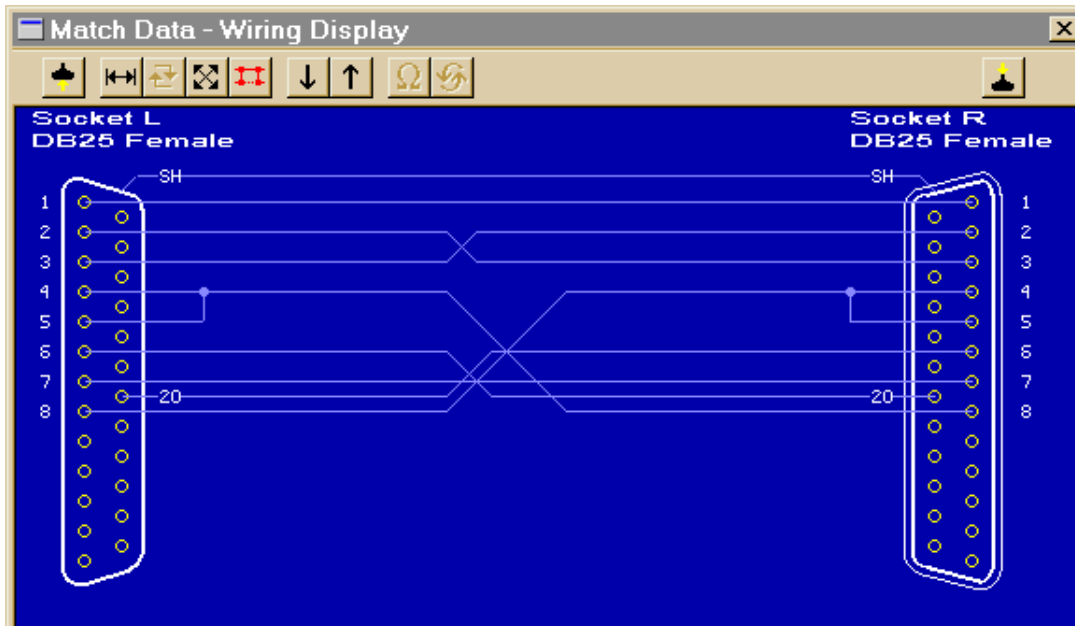


84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the Request-to-Send (pin 4) on one side asserts the Clear-to-Send (pin 5) on the SAME side (self-connect) and the Carrier Detect (pin 8) on the other side. The other handshaking signals are employed in a conventional manner.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

### 13 - DB25 Null Modem (unusual handshaking)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

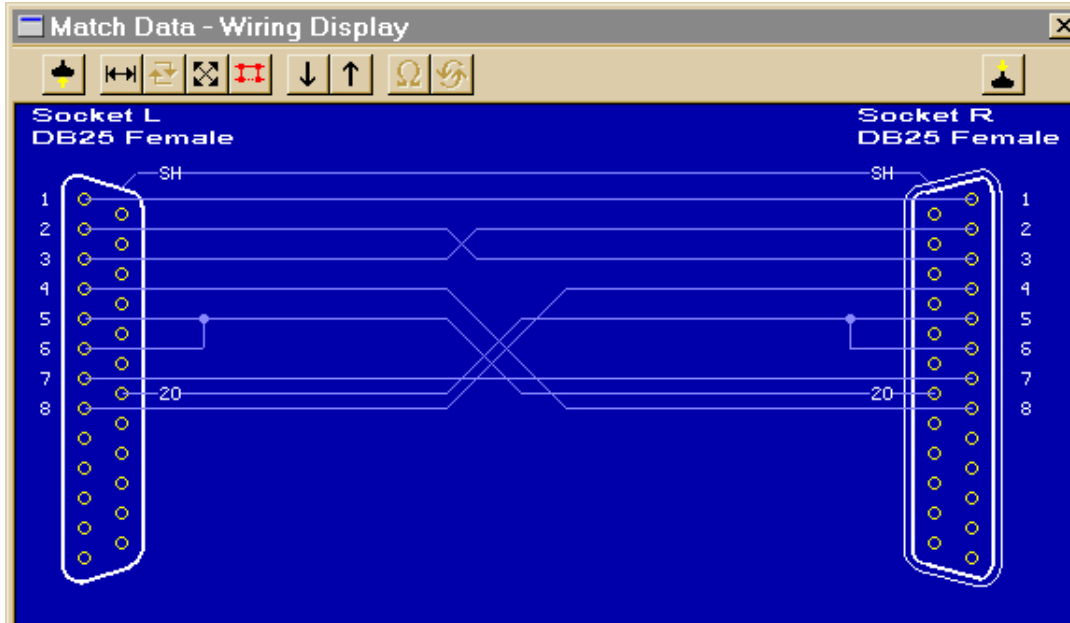


84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the Clear-to-Send (pin 5) and the DCE Ready (pin 6) on the other side. Request-to-Send (pin 4) on one side asserts Received Line Signal Detect (pin 8) on the other side.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

#### 14 - DB25 Null Modem (unusual handshaking)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

Use this female-to-female cable in any application where you wish to connect two DTE devices (for example, two computers). A male-to-male equivalent of this cable would be used to connect two DCE devices.

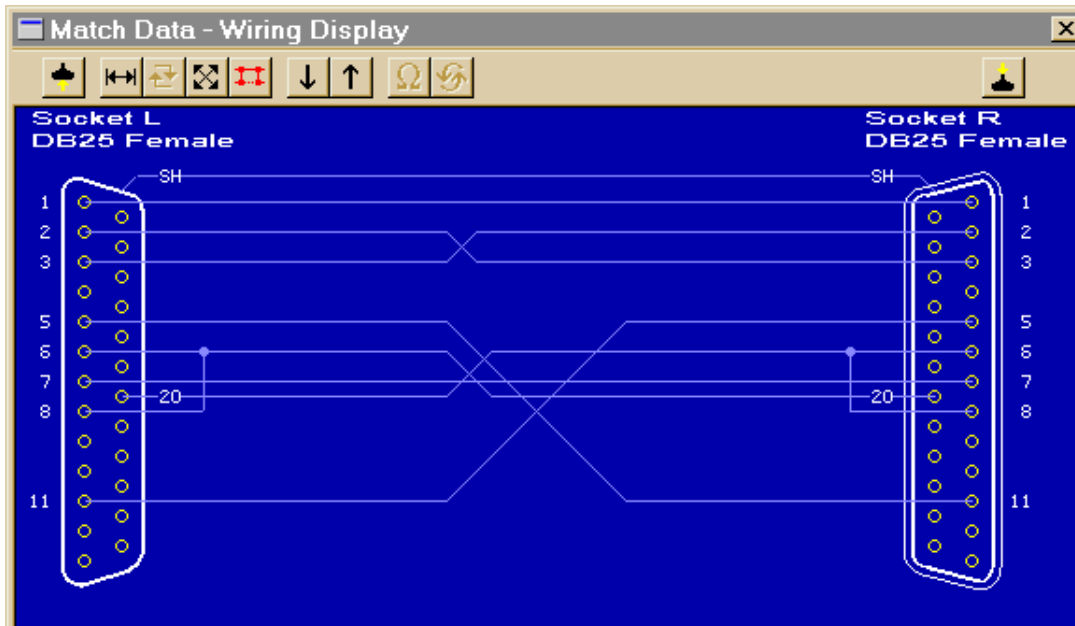


84K

NOTE: Not all null modem cables connect handshaking lines the same way. Refer to the manual for your equipment if you experience problems. In this cable, the DTE Ready (pin 20) on one side asserts the DCE Ready (pin 6), and Carrier Detect (pin 8) on the other side. Request to Send (pin 4) is unused, and Clear-to-Send (pin 5) is driven by a proprietary signal (pin 11) determined by the designer of this cable.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)



Cable image created by CableEye®

### 15 - DB25 Null Modem Cable (synchronous communications)

[Next Cable](#) | [Previous Cable](#) || [Next Topic](#)

This female-to-female cable is intended for **synchronous** EIA232 connections, and is designed to connect two DTE devices. It contains the standard connections of an asynchronous null modem cable, plus additional connections on pins 15, 17, and 24 for synchronous timing signals. To connect two DCE devices, use a male-to-male equivalent of this cable.



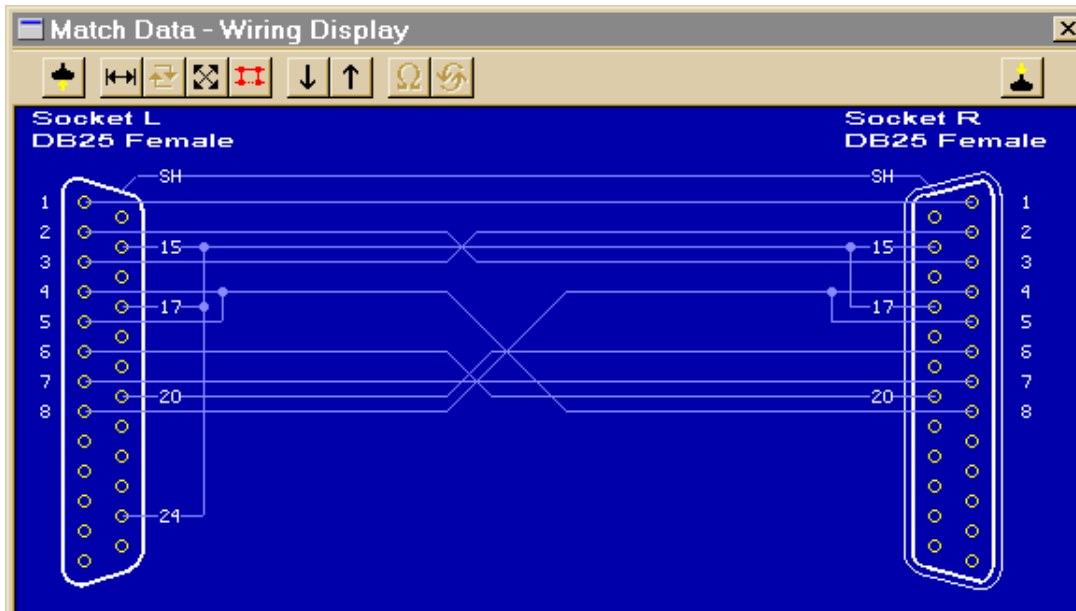
84K

For synchronous communications, the null modem cable includes an additional conductor for timing signals, and joins pins 15, 17, and 24 on one side to pins 15 and 17 on the other. Pin 24 on the right side should connect to the timing signal source.

*Left Side:* Connect to 25-pin **DTE**  
(computer)

*Right Side:* Connect to 25-pin **DTE**  
(computer)





Cable image created by CableEye®

**16 - DB25 Null Modem Cable** (unconventional, may pose risk)  
 (no more) | Previous Cable || Next Topic

This simplified null modem cable uses only Request-to-Send (pin 4) and Clear-to-Send (pin 5) as handshaking lines; DTE Ready, DCE Ready, and Carrier Detect are not employed, so this cable should not be used with modems.

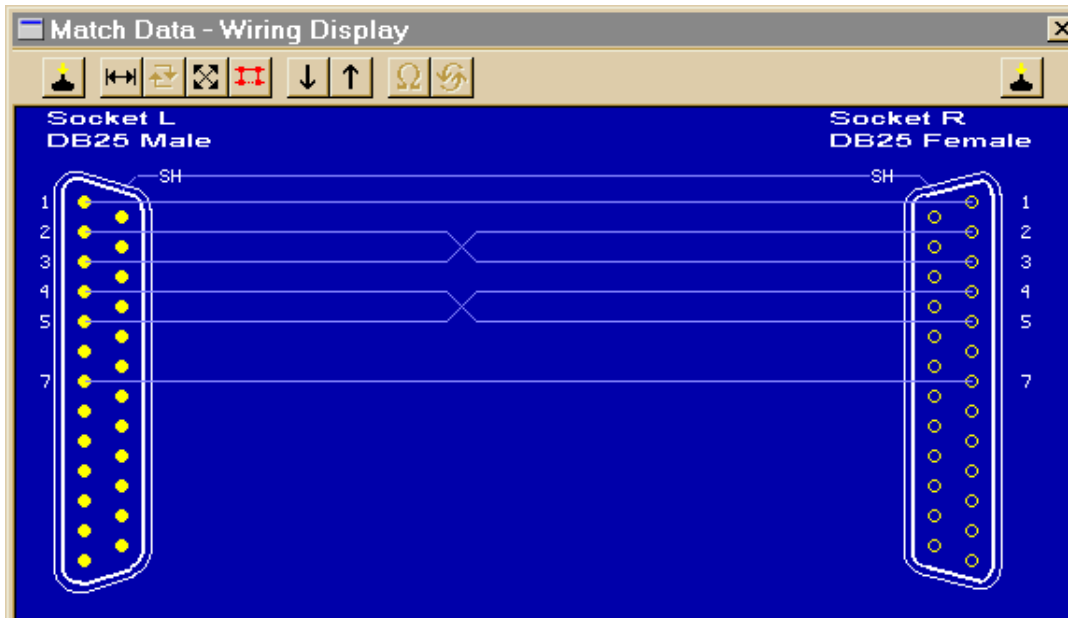


80K

**CAUTION!** Normally, null modem cables have the same gender on each connector (either both male for two DTE devices, or both female for two DCE devices). This cable would be used when the gender on one of the devices does not conform to the standard. However, the opposite genders imply usage as a straight through cable, and if used in that manner will not function. Further, if used as a standard null-modem between two computers, the opposite gender allows you to connect one end to the parallel port, an impermissible situation that may cause hardware damage.

*Left Side:* Connect to 25-pin **DTE**  
 (computer) with Gender Changer

*Right Side:* Connect to 25-pin **DTE**  
 (computer)



Cable image created by CableEye®

---

## Signal Definitions

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

Signal functions in the EIA232 standard can be subdivided into six categories. These categories are summarized below, after which each signal described.

1 - Signal ground and shield.

2 - Primary communications channel. This is used for data interchange, and includes flow control signals.

3 - Secondary communications channel. When implemented, this is used for control of the remote modem, requests for retransmission when errors occur, and governance over the setup of the primary channel.

4 - Modem status and control signals. These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.

5 - Transmitter and receiver timing signals. If a synchronous protocol is used, these signals provide timing information for the transmitter and receiver, which may operate at different baud rates.

6 - Channel test signals. Before data is exchanged, the channel may be tested for its integrity, and the baud rate automatically adjusted to the maximum rate that the channel can support.

---

## Signal Ground and Shield

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 7, Pin 1**, and the **shell** are included in this category. Cables provide separate paths for each, but internal wiring often connects pin 1 and the cable shell/shield to signal ground on pin 7.

**Pin 7 - Ground** All signals are referenced to a common ground, as defined by the voltage on pin 7. This conductor may or may not be connected to protective ground inside the DCE device. The existence of a defined ground potential within the cable makes the EIA232 standard different from a balanced differential voltage standard, such as EIA530, which provides far greater noise immunity.

---

## Primary Communications Channel

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 2 - Transmitted Data (TxD)** This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic '1', negative voltage).

*NOTE:* Pin 2 on the DCE device is commonly labeled "Received Data", although by the EIA232 standard it should still be called Transmitted Data because the data is thought to be destined for a remote DTE device.

**Pin 3 - Received Data (RxD)** This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic '1', negative voltage).

*NOTE:* Pin 3 on the DCE device is commonly labeled "Transmitted Data", although by the EIA232 standard it should still be called Received Data because the data is thought to arrive from a remote DTE device.

**Pin 4 - Request to Send (RTS)** This signal is asserted (logic '0', positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device. Such preparation might include enabling the receive circuits, or setting up the channel direction in half-duplex applications. When the DCE is ready, it acknowledges by asserting Clear to Send.

*NOTE:* Pin 4 on the DCE device is commonly labeled "Clear to Send", although by the EIA232 standard it should still be called Request to Send because the request is thought to be destined for a remote DTE device.

**Pin 5 - Clear to Send (CTS)** This signal is asserted (logic '0', positive voltage) by the DCE device to inform the DTE device that transmission may begin. RTS and CTS are commonly used as handshaking signals to moderate the flow of data into the DCE device.

*NOTE:* Pin 5 on the DCE device is commonly labeled "Request to Send", although by the

EIA232 standard it should still be called Clear to Send because the signal is thought to originate from a remote DTE device.

---

## Secondary Communications Channel

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 14 - Secondary Transmitted Data (STxD)**

**Pin 16 - Secondary Received Data (SRxD)**

**Pin 19 - Secondary Request to Send (SRTS)**

**Pin 13 - Secondary Clear to Send (SCTS)**

These signals are equivalent to the corresponding signals in the primary communications channel. The baud rate, however, is typically much slower in the secondary channel for increased reliability.

---

## Modem Status and Control Signals

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 6 - DCE Ready (DSR)** When originating from a modem, this signal is asserted (logic '0', positive voltage) when the following three conditions are all satisfied:

- 1 - The modem is connected to an active telephone line that is "off-hook";
- 2 - The modem is in data mode, not voice or dialing mode; and
- 3 - The modem has completed dialing or call setup functions and is generating an answer tone.

If the line goes "off-hook", a fault condition is detected, or a voice connection is established, the DCE Ready signal is deasserted (logic '1', negative voltage).

*IMPORTANT:* If DCE Ready originates from a device other than a modem, it may be asserted to indicate that the device is turned on and ready to function, or it may not be used at all. If unused, DCE Ready should be permanently asserted (logic '0', positive voltage) within the DCE device or by use of a self-connect jumper in the cable. Alternatively, the DTE device may be programmed to ignore this signal.

**Pin 20 - DTE Ready (DTR)** This signal is asserted (logic '0', positive voltage) by the DTE device when it wishes to open a communications channel. If the DCE device is a modem, the assertion of DTE Ready prepares the modem to be connected to the telephone circuit, and, once connected, maintains the

connection. When DTE Ready is deasserted (logic '1', negative voltage), the modem is switched to "on-hook" to terminate the connection.

*IMPORTANT:* If the DCE device is not a modem, it may require DTE Ready to be asserted before the device can be used, or it may ignore DTE Ready altogether. If the DCE device (for example, a printer) is not responding, confirm that DTE Ready is asserted before you search for other explanations.

**Pin 8 - Received Line Signal Detector (CD)** (also called carrier detect) This signal is relevant when the DCE device is a modem. It is asserted (logic '0', positive voltage) by the modem when the telephone line is "off-hook", a connection has been established, and an answer tone is being received from the remote modem. The signal is deasserted when no answer tone is being received, or when the answer tone is of inadequate quality to meet the local modem's requirements (perhaps due to a noisy channel).

**Pin 12 - Secondary Received Line Signal Detector (SCD)** This signal is equivalent to the Received Line Signal Detector (pin 8), but refers to the secondary channel.

**Pin 22 - Ring Indicator (RI)** This signal is relevant when the DCE device is a modem, and is asserted (logic '0', positive voltage) when a ringing signal is being received from the telephone line. The assertion time of this signal will approximately equal the duration of the ring signal, and it will be deasserted between rings or when no ringing is present.

**Pin 23 - Data Signal Rate Selector** This signal may originate either in the DTE or DCE devices (but not both), and is used to select one of two prearranged baud rates. The asserted condition (logic '0', positive voltage) selects the higher baud rate.

---

## Transmitter and Receiver Timing Signals

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 15 - Transmitter Signal Element Timing (TC)** (also called Transmitter Clock) This signal is relevant only when the DCE device is a modem and is operating with a synchronous protocol. The modem generates this clock signal to control exactly the rate at which data is sent on Transmitted Data (pin 2) from the DTE device to the DCE device. The logic '1' to logic '0' (negative voltage to positive voltage) transition on this line causes a corresponding transition to the next data element on the Transmitted Data line. The modem generates this signal continuously, except when it is performing internal diagnostic functions.

**Pin 17 - Receiver Signal Element Timing (RC)** (also called Receiver Clock) This signal is similar to TC described above, except that it provides timing information for the DTE receiver.

**Pin 24 - Transmitter Signal Element Timing (ETC)** (also called External Transmitter Clock) Timing signals are provided by the DTE device for use by a modem. This signal is used only when TC and RC (pins 15 and 17) are not in use. The logic '1' to logic '0' transition (negative voltage to positive voltage) indicates the time-center of the data element. Timing signals will be provided whenever the DTE is turned on, regardless of other signal conditions.

---

## Channel Test Signals

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 18 - Local Loopback (LL)** This signal is generated by the DTE device and is used to place the modem into a test state. When Local Loopback is asserted (logic '0', positive voltage), the modem redirects its modulated output signal, which is normally fed into the telephone line, back into its receive circuitry. This enables data generated by the DTE to be echoed back through the local modem to check the condition of the modem circuitry. The modem asserts its Test Mode signal on Pin 25 to acknowledge that it has been placed in local loopback condition.

**Pin 21 - Remote Loopback (RL)** This signal is generated by the DTE device and is used to place the remote modem into a test state. When Remote Loopback is asserted (logic '0', positive voltage), the remote modem redirects its received data back to its transmitted data input, thereby remodulating the received data and returning it to its source. When the DTE initiates such a test, transmitted data is passed through the local modem, the telephone line, the remote modem, and back, to exercise the channel and confirm its integrity. The remote modem signals the local modem to assert Test Mode on pin 25 when the remote loopback test is underway.

**Pin 25 - Test Mode (TM)** This signal is relevant only when the DCE device is a modem. When asserted (logic '0', positive voltage), it indicates that the modem is in a Local Loopback or Remote Loopback condition. Other internal self-test conditions may also cause Test Mode to be asserted, and depend on the modem and the network to which it is attached.

---

## Electrical Standards

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

The EIA232 standard uses negative, bipolar logic in which a negative voltage signal represents logic '1', and positive voltage represents logic '0'. This probably originated with the pre-RS232 current loop standard used in 1950s-vintage teletype machines in which a flowing current (and hence a low voltage) represents logic '1'. Be aware that the negative logic assignment of EIA232 is the reverse of that found in most modern digital circuit designs. See the inside rear cover of the CableEye manual for a comparison.

---

## Common Signal Ground

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

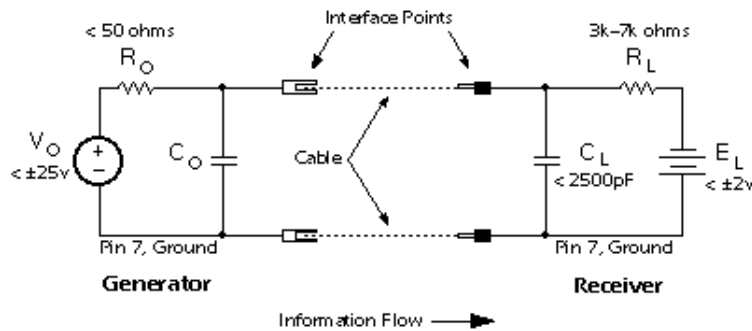
The EIA232 standard includes a common ground reference on Pin 7, and is frequently joined to Pin 1 and a circular shield that surrounds all 25 cable conductors. Data, timing, and control signal voltages are measured with respect to this common ground. EIA232 cannot be used in applications where the equipment on opposite ends of the connection must be electrically isolated.

*NOTE:* optical isolators may be used to achieve ground isolation, however, this option is not mentioned or included in the EIA232 specification.

## Signal Characteristics

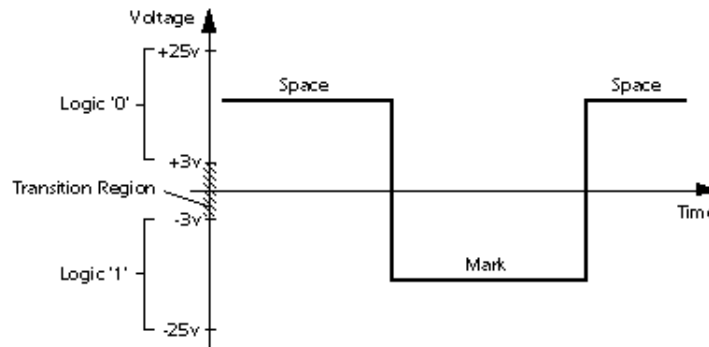
[Next Topic](#) | [Previous Topic](#) | [TOC](#)

Equivalent Circuit - All signal lines, regardless of whether they provide data, timing, or control information, may be represented by the electrical equivalent circuit shown here:



*This is the equivalent circuit for an EIA232 signal line and applies to signals originating at either the DTE or DCE side of the connection. "C<sub>o</sub>" is not specified in the standard, but is assumed to be small and to consist of parasitic elements only. "R<sub>o</sub>" and "V<sub>o</sub>" are chosen so that the short-circuit current does not exceed 500ma. The cable length is not specified in the standard; acceptable operation is experienced with cables that are less than 25 feet in length.*

**Signal State Voltage Assignments** - Voltages of -3v to -25v with respect to signal ground (pin 7) are considered logic '1' (the marking condition), whereas voltages of +3v to +25v are considered logic '0' (the spacing condition). The range of voltages between -3v and +3v is considered a transition region for which a signal state is not assigned.



*Logic states are assigned to the voltage ranges shown here. Note that this is a "negative logic" convention, which is the reverse of that used in most modern*

## *digital designs.*

Most contemporary applications will show an open-circuit signal voltage of -8 to -14 volts for logic '1' (mark), and +8 to +14 volts for logic '0' (space). Voltage magnitudes will be slightly less when the generator and receiver are connected (when the DTE and DCE devices are connected with a cable).

*IMPORTANT:* If you insert an LED signal tester in an EIA232 circuit to view signal states, the signal voltage may drop in magnitude to very near the minimum values of -3v for logic '1', and +3v for logic '0'. Also note that some inexpensive EIA232 peripherals are powered directly from the signal lines to avoid using a power supply of their own. Although this usually works without problems, keep the cable short, and be aware that noise immunity will be reduced.

**Short-Circuit Tolerance** - The generator is designed to withstand an open-circuit (unconnected) condition, or short-circuit condition between its signal conductor and any other signal conductor, including ground, without sustaining damage to itself or causing damage to any associated circuitry. The receiver is also designed to accept any signal voltage within the range of  $\pm 25$  volts without sustaining damage.

*CAUTION:* Inductive loads or magnetically induced voltages resulting from long cables may cause the received voltage to exceed the  $\pm 25$ -volt range momentarily during turn-on transients or other abnormal conditions, possibly causing damage to the generator, receiver, or both. Keep the cable length as short as possible, and avoid running the cable near high-current switching loads like electric motors or relays.

**Fail-Safe Signals** - Four signals are intended to be fail-safe in that during power-off or cable-disconnected conditions, they default to logic '1' (negative voltage). They are:

Request to Send - Default condition is deasserted.

Sec. Request to Send - Default condition is deasserted.

DTE Ready - Default condition is DTE not ready.

DCE Ready - Default condition is DCE not ready.

Note specifically that if the cable is connected but the power is off in the generator side, or if the cable is disconnected, there should be adequate bias voltage in the receiver to keep the signal above +3v (logic '0') to ensure that the fail-safe requirement is met.

Schmitt triggers or other hysteresis devices may be used to enhance noise immunity in some designs, but should never be adjusted to compromise the fail-safe requirement.

---

## **Signal Timing**

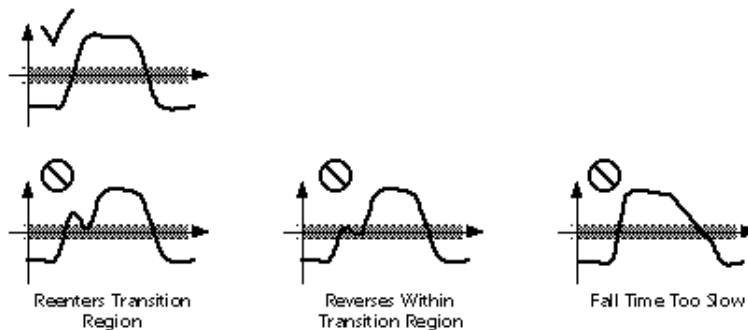
[Next Topic](#) | [Previous Topic](#) | [TOC](#)



The EIA232 standard is applicable to data rates of up to 20,000 bits per second (the usual upper limit is 19,200 baud). Fixed baud rates are not set by the EIA232 standard. However, the commonly used values are 300, 1200, 2400, 9600, and 19,200 baud. Other accepted values that are not often used are 110 (mechanical teletype machines), 600, and 4800 baud.

Changes in signal state from logic '1' to logic '0' or vice versa must abide by several requirements, as follows:

- 1 - Signals that enter the transition region during a change of state must move through the transition region to the opposite signal state without reversing direction or reentering.
- 2 - For control signals, the transit time through the transition region should be less than 1ms.
- 3 - For Data and Timing signals, the transit time through the transition region should be
  - a - less than 1ms for bit periods greater than 25ms,
  - b - 4% of the bit period for bit periods between 25ms and 125 $\mu$ s,
  - c - less than 5 $\mu$ s for bit periods less than 125 $\mu$ s.The rise and fall times of data and timing signals ideally should be equal, but in any case vary by no more than a factor of three.



*An acceptable pulse (top) moves through the transition region quickly and without hesitation or reversal. Defective pulses (bottom) could cause data errors.*

- 
- 4 - The slope of the rising and falling edges of a transition should not exceed 30v/ $\mu$ S. Rates higher than this may induce crosstalk in adjacent conductors of a cable.

Note that neither the ASCII alphabet nor the asynchronous serial protocol that defines the start bit, number of data bits, parity bit, and stop bit, is part of the EIA232 specification. For your reference, it is discussed in the Data Communications Basics section of this web site.

---

## **Accepted Simplifications of the Standard**

[Previous Topic](#) | [TOC](#)

The EIA232 document published by the Electronic Industries Association describes 14 permissible configurations of the original 22-signal standard. Each configuration uses a subset of the 22 defined signals, and serves a more limited communications requirement than that suggested by using all the available 22-signals. Applications for transmit-only, receive-only, half-duplex operation, and similar variations, are described. Unfortunately, connection to DCE devices other than modems is not considered. Because many current serial interface applications involve direct device-to-device connections, manufacturers do not have a standard reference when producing printers, plotters, print spoolers, or other common peripherals. Consequently, you must acquire the service manual for each peripheral device purchased to determine exactly which signals are utilized in its serial interface.

END

[Return to TOC](#)

# The RS232 STANDARD

*A Tutorial with Signal Names and Definitions*

(renamed the "EIA232 Standard" in the early 1990's)

Written by Christopher E. Strangio

Copyright © 1993-1997 by CAMI Research Inc., Lexington, Massachusetts

Send Us Your Comments . . .

## Contents

What is EIA232?  
Likely Problems when Using an EIA232 Interface  
Pin Assignments  
Signal Definitions  
Signal Ground and Shield  
Primary Communications Channel  
Secondary Communications Channel  
Modem Status and Control Signals  
Transmitter and Receiver Timing Signals  
Channel Test Signals  
Electrical Standards  
Common Signal Ground  
Signal Characteristics  
Signal Timing  
Accepted Simplifications of the Standard

Pin Description Index

References to EIA Publications

[Back to CableEye Home Page](#)

---

## What is EIA232?

[Next Topic | TOC](#)

In the early 1960s, a standards committee, today known as the Electronic Industries Association, developed a common interface standard for data communications equipment. At that time, data communications was thought to mean digital data exchange between a centrally located mainframe computer and a remote computer terminal, or possibly between two terminals without a computer involved. These devices were linked by telephone voice lines, and consequently required a modem at each end for signal translation. While simple in concept, the many opportunities for data error that occur when transmitting data through an analog channel require a relatively complex design. It was thought that a standard was needed first to ensure reliable communication, and second to enable the interconnection of equipment produced by different manufacturers, thereby fostering the benefits of

mass production and competition. From these ideas, the RS232 standard was born. It specified signal voltages, signal timing, signal function, a protocol for information exchange, and mechanical connectors.

Over the 30+ years since this standard was developed, the Electronic Industries Association published three modifications, the most recent being the EIA232E standard introduced in 1991. Besides changing the name from RS232 to EIA232, some signal lines were renamed and various new ones were defined, including a shield conductor.

---

## **Likely Problems when Using an EIA232 Interface**

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

During this 30-year-long, rapidly evolving period in electronics, manufacturers adopted simplified versions of this interface for applications that were impossible to envision in the 1960s. Today, virtually all contemporary serial interfaces are EIA232-like in their signal voltages, protocols, and connectors, whether or not a modem is involved. Because no single "simplified" standard was agreed upon, however, many slightly different protocols and cables were created that obligingly mate with any EIA232 connector, but are incompatible with each other. Most of the difficulties you will encounter in EIA232 interfacing include at least one of the following:

1 - The absence or misconnection of flow control (handshaking) signals, resulting in buffer overflow or communications lock-up.

2 - Incorrect communications function (DTE versus DCE) for the cable in use, resulting in the reversal of the Transmit and Receive data lines as well as one or more handshaking lines.

3 - Incorrect connector gender or pin configuration, preventing cable connectors from mating properly.

Fortunately, EIA232 driver circuitry is highly tolerant of misconnections, and will usually survive a drive signal being connected to ground, or two drive signals connected to each other. In any case, if the serial interface between two devices is not operating correctly, disconnect the cable joining this equipment until the problem is isolated.

---

## **Pin Assignments**

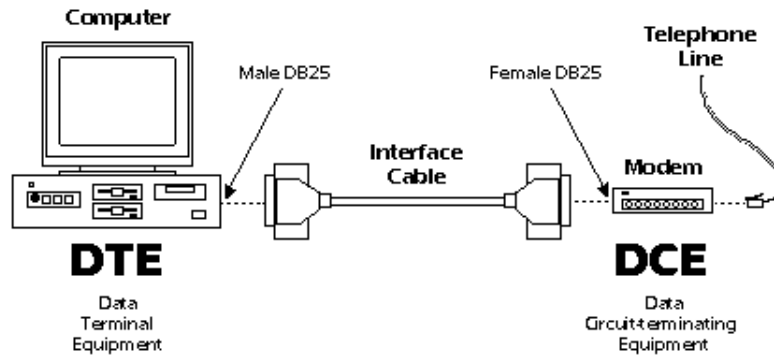
[Next Topic](#) | [Previous Topic](#) | [TOC](#)

Go to DTE Pinout (looking into the computer's serial connector)

Go to DCE Pinout (looking into the modem's serial connector)

If the full EIA232 standard is implemented as defined, the equipment at the far end of the connection is named the DTE device (Data Terminal Equipment, usually a computer or terminal), has a male DB25

connector, and utilizes 22 of the 25 available pins for signals or ground. Equipment at the near end of the connection (the telephone line interface) is named the DCE device (Data Circuit-terminating Equipment, usually a modem), has a female DB25 connector, and utilizes the same 22 available pins for signals and ground. The cable linking DTE and DCE devices is a parallel straight-through cable with no cross-overs or self-connects in the connector hoods. If all devices exactly followed this standard, all cables would be identical, and there would be no chance that an incorrectly wired cable could be used. This drawing shows the orientation and connector types for DTE and DCE devices:



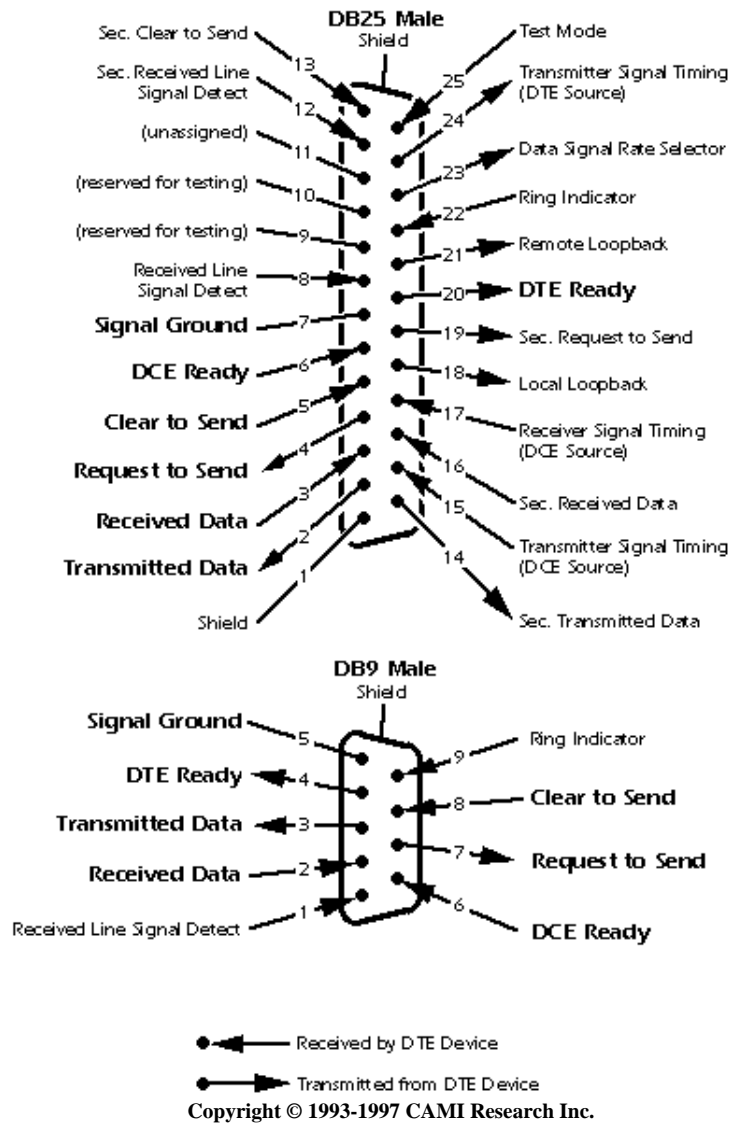
*EIA232 communication function and connector types for a personal computer and modem. DCE devices are sometimes called "Data Communications Equipment" instead of Data Circuit-terminating Equipment.*

---

Here is the full EIA232 signal definition for the DTE device (usually the PC). The most commonly used signals are shown in bold.

[back to Pin Assignments description]

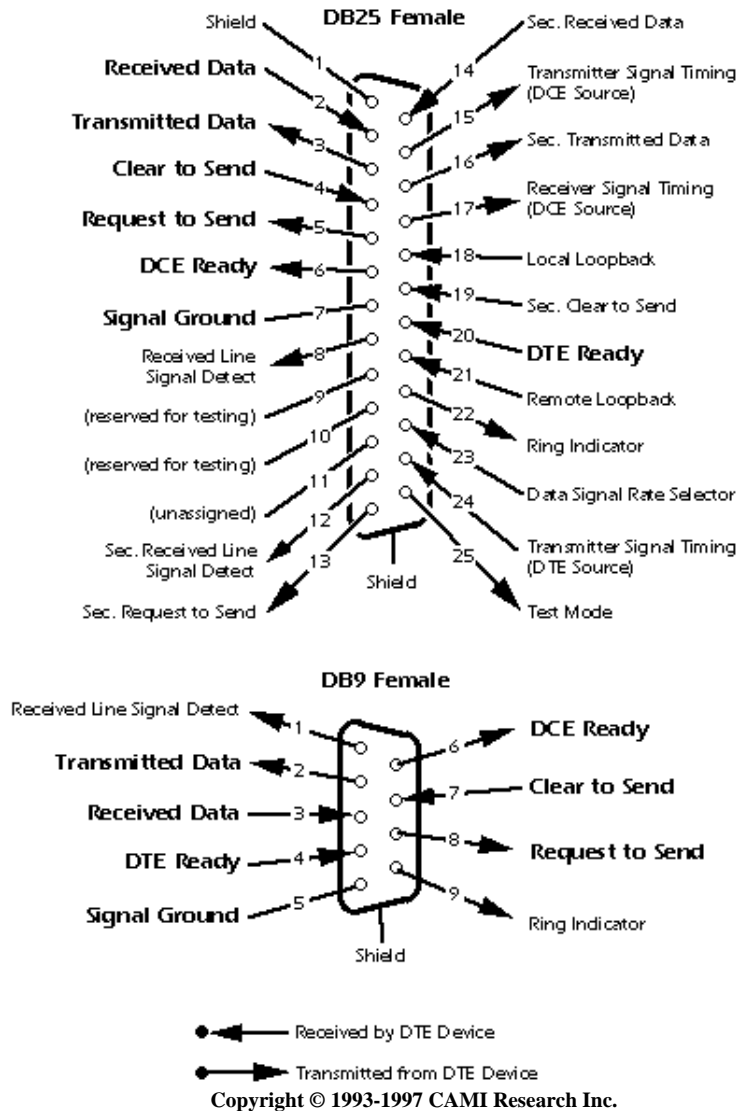
## Looking Into the DTE Device Connector



This shows the full EIA232 signal definition for the DCE device (usually the modem). The most commonly used signals are shown in bold.

[back to Pin Assignments description]

## Looking Into the DCE Device Connector



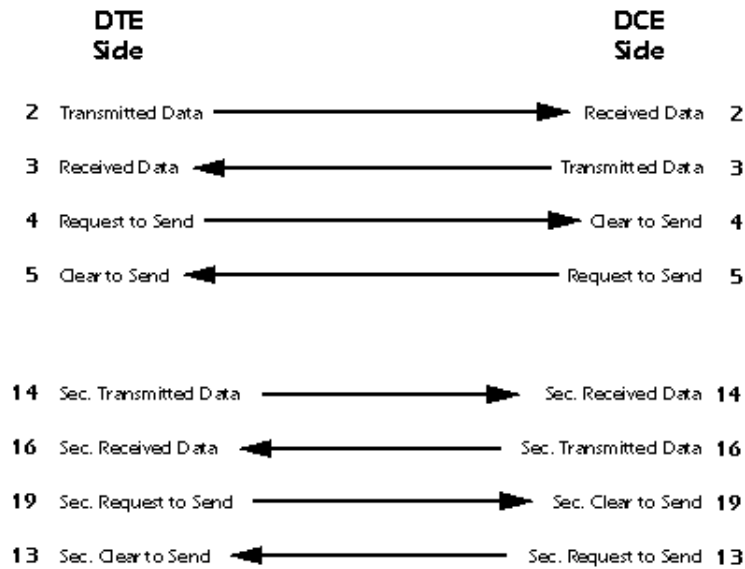
Many of the 22 signal lines in the EIA232 standard pertain to connections where the DCE device is a modem, and then are used only when the software protocol employs them. For any DCE device that is not a modem, or when two DTE devices are directly linked, far fewer signal lines are necessary.

You may have noticed in the pinout drawings that there is a secondary channel which includes a duplicate set of flow-control signals. This secondary channel provides for management of the remote modem, enabling baud rates to be changed on the fly, retransmission to be requested if a parity error is detected, and other control functions. This secondary channel, when used, is typically set to operate at a very low baud rate in comparison with the primary channel to ensure reliability in the control path. In addition, it may operate as either a simplex, half-duplex, or full-duplex channel, depending on the capabilities of the modem.

Transmitter and receiver timing signals (pins 15, 17, and 24) are used only for a synchronous

transmission protocol. For the standard asynchronous 8-bit protocol, external timing signals are unnecessary.

**IMPORTANT:** Signal names that imply a direction, such as Transmit Data and Receive Data, are named from the point of view of the DTE device. If the EIA232 standard were strictly followed, these signals would have the same name for the same pin number on the DCE side as well. Unfortunately, this is not done in practice by most engineers, probably because no one can keep straight which side is DTE and which is DCE. As a result, direction-sensitive signal names are changed at the DCE side to reflect their *drive direction* at DCE. The following list gives the conventional usage of signal names:




---

## Signal Definitions

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

Signal functions in the EIA232 standard can be subdivided into six categories. These categories are summarized below, after which each signal described.

- 1 - Signal ground and shield.
- 2 - Primary communications channel. This is used for data interchange, and includes flow control signals.
- 3 - Secondary communications channel. When implemented, this is used for control of the remote modem, requests for retransmission when errors occur, and governance over the setup of the primary channel.
- 4 - Modem status and control signals. These signals indicate modem status and provide intermediate checkpoints as the telephone voice channel is established.
- 5 - Transmitter and receiver timing signals. If a synchronous protocol is used, these signals



provide timing information for the transmitter and receiver, which may operate at different baud rates.

6 - Channel test signals. Before data is exchanged, the channel may be tested for its integrity, and the baud rate automatically adjusted to the maximum rate that the channel can support.

---

## Signal Ground and Shield

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 7, Pin 1**, and the **shell** are included in this category. Cables provide separate paths for each, but internal wiring often connects pin 1 and the cable shell/shield to signal ground on pin 7.

**Pin 7 - Ground** All signals are referenced to a common ground, as defined by the voltage on pin 7. This conductor may or may not be connected to protective ground inside the DCE device. The existence of a defined ground potential within the cable makes the EIA232 standard different from a balanced differential voltage standard, such as EIA530, which provides far greater noise immunity.

---

## Primary Communications Channel

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 2 - Transmitted Data (TxD)** This signal is active when data is transmitted from the DTE device to the DCE device. When no data is transmitted, the signal is held in the mark condition (logic '1', negative voltage).

*NOTE:* Pin 2 on the DCE device is commonly labeled "Received Data", although by the EIA232 standard it should still be called Transmitted Data because the data is thought to be destined for a remote DTE device.

**Pin 3 - Received Data (RxD)** This signal is active when the DTE device receives data from the DCE device. When no data is transmitted, the signal is held in the mark condition (logic '1', negative voltage).

*NOTE:* Pin 3 on the DCE device is commonly labeled "Transmitted Data", although by the EIA232 standard it should still be called Received Data because the data is thought to arrive from a remote DTE device.

**Pin 4 - Request to Send (RTS)** This signal is asserted (logic '0', positive voltage) to prepare the DCE device for accepting transmitted data from the DTE device. Such preparation might include enabling the receive circuits, or setting up the channel direction in half-duplex applications. When the DCE is ready, it acknowledges by asserting Clear to Send.

*NOTE:* Pin 4 on the DCE device is commonly labeled "Clear to Send", although by the

EIA232 standard it should still be called Request to Send because the request is thought to be destined for a remote DTE device.

**Pin 5 - Clear to Send (CTS)** This signal is asserted (logic '0', positive voltage) by the DCE device to inform the DTE device that transmission may begin. RTS and CTS are commonly used as handshaking signals to moderate the flow of data into the DCE device.

*NOTE:* Pin 5 on the DCE device is commonly labeled "Request to Send", although by the EIA232 standard it should still be called Clear to Send because the signal is thought to originate from a remote DTE device.

---

## Secondary Communications Channel

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 14 - Secondary Transmitted Data (STxD)**

**Pin 16 - Secondary Received Data (SRxD)**

**Pin 19 - Secondary Request to Send (SRTS)**

**Pin 13 - Secondary Clear to Send (SCTS)**

These signals are equivalent to the corresponding signals in the primary communications channel. The baud rate, however, is typically much slower in the secondary channel for increased reliability.

---

## Modem Status and Control Signals

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 6 - DCE Ready (DSR)** When originating from a modem, this signal is asserted (logic '0', positive voltage) when the following three conditions are all satisfied:

- 1 - The modem is connected to an active telephone line that is "off-hook";
- 2 - The modem is in data mode, not voice or dialing mode; and
- 3 - The modem has completed dialing or call setup functions and is generating an answer tone.

If the line goes "off-hook", a fault condition is detected, or a voice connection is established, the DCE Ready signal is deasserted (logic '1', negative voltage).

*IMPORTANT:* If DCE Ready originates from a device other than a modem, it may be

asserted to indicate that the device is turned on and ready to function, or it may not be used at all. If unused, DCE Ready should be permanently asserted (logic '0', positive voltage) within the DCE device or by use of a self-connect jumper in the cable. Alternatively, the DTE device may be programmed to ignore this signal.

**Pin 20 - DTE Ready (DTR)** This signal is asserted (logic '0', positive voltage) by the DTE device when it wishes to open a communications channel. If the DCE device is a modem, the assertion of DTE Ready prepares the modem to be connected to the telephone circuit, and, once connected, maintains the connection. When DTE Ready is deasserted (logic '1', negative voltage), the modem is switched to "on-hook" to terminate the connection.

*IMPORTANT:* If the DCE device is not a modem, it may require DTE Ready to be asserted before the device can be used, or it may ignore DTE Ready altogether. If the DCE device (for example, a printer) is not responding, confirm that DTE Ready is asserted before you search for other explanations.

**Pin 8 - Received Line Signal Detector (CD)** (also called carrier detect) This signal is relevant when the DCE device is a modem. It is asserted (logic '0', positive voltage) by the modem when the telephone line is "off-hook", a connection has been established, and an answer tone is being received from the remote modem. The signal is deasserted when no answer tone is being received, or when the answer tone is of inadequate quality to meet the local modem's requirements (perhaps due to a noisy channel).

**Pin 12 - Secondary Received Line Signal Detector (SCD)** This signal is equivalent to the Received Line Signal Detector (pin 8), but refers to the secondary channel.

**Pin 22 - Ring Indicator (RI)** This signal is relevant when the DCE device is a modem, and is asserted (logic '0', positive voltage) when a ringing signal is being received from the telephone line. The assertion time of this signal will approximately equal the duration of the ring signal, and it will be deasserted between rings or when no ringing is present.

**Pin 23 - Data Signal Rate Selector** This signal may originate either in the DTE or DCE devices (but not both), and is used to select one of two prearranged baud rates. The asserted condition (logic '0', positive voltage) selects the higher baud rate.

---

## Transmitter and Receiver Timing Signals

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 15 - Transmitter Signal Element Timing (TC)** (also called Transmitter Clock) This signal is relevant only when the DCE device is a modem and is operating with a synchronous protocol. The modem generates this clock signal to control exactly the rate at which data is sent on Transmitted Data (pin 2) from the DTE device to the DCE device. The logic '1' to logic '0' (negative voltage to positive voltage) transition on this line causes a corresponding transition to the next data element on the Transmitted Data line. The modem generates this signal continuously, except when it is performing internal diagnostic functions.

**Pin 17 - Receiver Signal Element Timing (RC)** (also called Receiver Clock) This signal is similar to TC described above, except that it provides timing information for the DTE receiver.

**Pin 24 - Transmitter Signal Element Timing (ETC)** (also called External Transmitter Clock) Timing signals are provided by the DTE device for use by a modem. This signal is used only when TC and RC (pins 15 and 17) are not in use. The logic '1' to logic '0' transition (negative voltage to positive voltage) indicates the time-center of the data element. Timing signals will be provided whenever the DTE is turned on, regardless of other signal conditions.

---

## Channel Test Signals

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

**Pin 18 - Local Loopback (LL)** This signal is generated by the DTE device and is used to place the modem into a test state. When Local Loopback is asserted (logic '0', positive voltage), the modem redirects its modulated output signal, which is normally fed into the telephone line, back into its receive circuitry. This enables data generated by the DTE to be echoed back through the local modem to check the condition of the modem circuitry. The modem asserts its Test Mode signal on Pin 25 to acknowledge that it has been placed in local loopback condition.

**Pin 21 - Remote Loopback (RL)** This signal is generated by the DTE device and is used to place the remote modem into a test state. When Remote Loopback is asserted (logic '0', positive voltage), the remote modem redirects its received data back to its transmitted data input, thereby remodulating the received data and returning it to its source. When the DTE initiates such a test, transmitted data is passed through the local modem, the telephone line, the remote modem, and back, to exercise the channel and confirm its integrity. The remote modem signals the local modem to assert Test Mode on pin 25 when the remote loopback test is underway.

**Pin 25 - Test Mode (TM)** This signal is relevant only when the DCE device is a modem. When asserted (logic '0', positive voltage), it indicates that the modem is in a Local Loopback or Remote Loopback condition. Other internal self-test conditions may also cause Test Mode to be asserted, and depend on the modem and the network to which it is attached.

---

## Electrical Standards

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

The EIA232 standard uses negative, bipolar logic in which a negative voltage signal represents logic '1', and positive voltage represents logic '0'. This probably originated with the pre-RS232 current loop standard used in 1950s-vintage teletype machines in which a flowing current (and hence a low voltage) represents logic '1'. Be aware that the negative logic assignment of EIA232 is the reverse of that found in most modern digital circuit designs. See the inside rear cover of the CableEye manual for a comparison.

---

## Common Signal Ground

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

The EIA232 standard includes a common ground reference on Pin 7, and is frequently joined to Pin 1 and a circular shield that surrounds all 25 cable conductors. Data, timing, and control signal voltages are measured with respect to this common ground. EIA232 cannot be used in applications where the equipment on opposite ends of the connection must be electrically isolated.

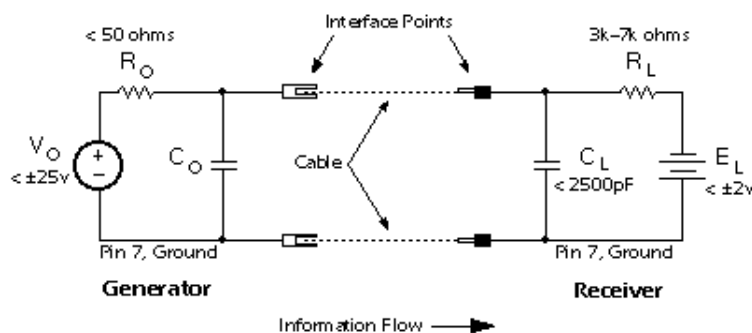
*NOTE:* optical isolators may be used to achieve ground isolation, however, this option is not mentioned or included in the EIA232 specification.

---

## Signal Characteristics

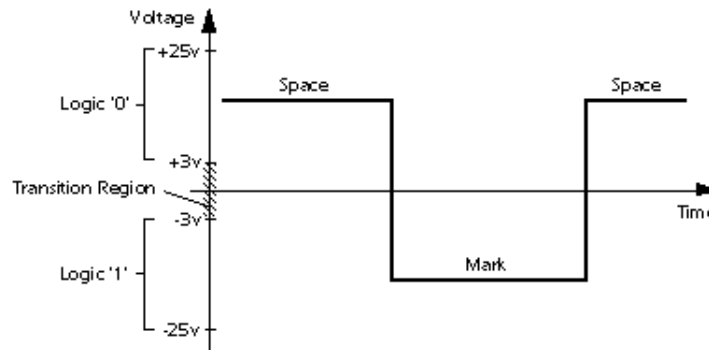
[Next Topic](#) | [Previous Topic](#) | [TOC](#)

Equivalent Circuit - All signal lines, regardless of whether they provide data, timing, or control information, may be represented by the electrical equivalent circuit shown here:



*This is the equivalent circuit for an EIA232 signal line and applies to signals originating at either the DTE or DCE side of the connection. "C<sub>o</sub>" is not specified in the standard, but is assumed to be small and to consist of parasitic elements only. "R<sub>o</sub>" and "V<sub>o</sub>" are chosen so that the short-circuit current does not exceed 500ma. The cable length is not specified in the standard; acceptable operation is experienced with cables that are less than 25 feet in length.*

**Signal State Voltage Assignments** - Voltages of -3v to -25v with respect to signal ground (pin 7) are considered logic '1' (the marking condition), whereas voltages of +3v to +25v are considered logic '0' (the spacing condition). The range of voltages between -3v and +3v is considered a transition region for which a signal state is not assigned.



*Logic states are assigned to the voltage ranges shown here. Note that this is a "negative logic" convention, which is the reverse of that used in most modern digital designs.*

Most contemporary applications will show an open-circuit signal voltage of -8 to -14 volts for logic '1' (mark), and +8 to +14 volts for logic '0' (space). Voltage magnitudes will be slightly less when the generator and receiver are connected (when the DTE and DCE devices are connected with a cable).

**IMPORTANT:** If you insert an LED signal tester in an EIA232 circuit to view signal states, the signal voltage may drop in magnitude to very near the minimum values of  $\pm 3$ v for logic '1', and +3v for logic '0'. Also note that some inexpensive EIA232 peripherals are powered directly from the signal lines to avoid using a power supply of their own. Although this usually works without problems, keep the cable short, and be aware that noise immunity will be reduced.

**Short-Circuit Tolerance** - The generator is designed to withstand an open-circuit (unconnected) condition, or short-circuit condition between its signal conductor and any other signal conductor, including ground, without sustaining damage to itself or causing damage to any associated circuitry. The receiver is also designed to accept any signal voltage within the range of  $\pm 25$  volts without sustaining damage.

**CAUTION:** Inductive loads or magnetically induced voltages resulting from long cables may cause the received voltage to exceed the  $\pm 25$ -volt range momentarily during turn-on transients or other abnormal conditions, possibly causing damage to the generator, receiver, or both. Keep the cable length as short as possible, and avoid running the cable near high-current switching loads like electric motors or relays.

**Fail-Safe Signals** - Four signals are intended to be fail-safe in that during power-off or cable-disconnected conditions, they default to logic '1' (negative voltage). They are:

Request to Send - Default condition is deasserted.

Sec. Request to Send - Default condition is deasserted.

DTE Ready - Default condition is DTE not ready.

DCE Ready - Default condition is DCE not ready.

Note specifically that if the cable is connected but the power is off in the generator side, or if the cable is disconnected, there should be adequate bias voltage in the receiver to keep the signal above +3v (logic '0') to ensure that the fail-safe requirement is met.

Schmitt triggers or other hysteresis devices may be used to enhance noise immunity in some designs, but should never be adjusted to compromise the fail-safe requirement.

---

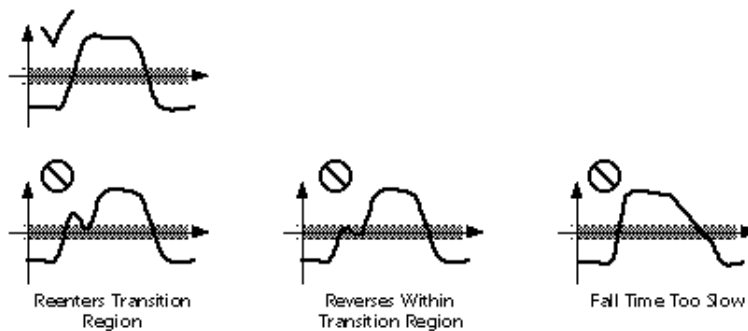
## Signal Timing

[Next Topic](#) | [Previous Topic](#) | [TOC](#)

The EIA232 standard is applicable to data rates of up to 20,000 bits per second (the usual upper limit is 19,200 baud). Fixed baud rates are not set by the EIA232 standard. However, the commonly used values are 300, 1200, 2400, 9600, and 19,200 baud. Other accepted values that are not often used are 110 (mechanical teletype machines), 600, and 4800 baud.

Changes in signal state from logic '1' to logic '0' or vice versa must abide by several requirements, as follows:

- 1 - Signals that enter the transition region during a change of state must move through the transition region to the opposite signal state without reversing direction or reentering.
- 2 - For control signals, the transit time through the transition region should be less than 1ms.
- 3 - For Data and Timing signals, the transit time through the transition region should be
  - a - less than 1ms for bit periods greater than 25ms,
  - b - 4% of the bit period for bit periods between 25ms and 125 $\mu$ s,
  - c - less than 5 $\mu$ s for bit periods less than 125 $\mu$ s.The rise and fall times of data and timing signals ideally should be equal, but in any case vary by no more than a factor of three.



*An acceptable pulse (top) moves through the transition region quickly and without hesitation or reversal. Defective pulses (bottom) could cause data errors.*

---

4 - The slope of the rising and falling edges of a transition should not exceed  $30\text{v}/\mu\text{S}$ . Rates higher than this may induce crosstalk in adjacent conductors of a cable.

Note that neither the ASCII alphabet nor the asynchronous serial protocol that defines the start bit, number of data bits, parity bit, and stop bit, is part of the EIA232 specification. For your reference, it is discussed in the Data Communications Basics section of this web site.

---

### **Accepted Simplifications of the Standard**

[Previous Topic](#) | [TOC](#)

The EIA232 document published by the Electronic Industries Association describes 14 permissible configurations of the original 22-signal standard. Each configuration uses a subset of the 22 defined signals, and serves a more limited communications requirement than that suggested by using all the available 22-signals. Applications for transmit-only, receive-only, half-duplex operation, and similar variations, are described. Unfortunately, connection to DCE devices other than modems is not considered. Because many current serial interface applications involve direct device-to-device connections, manufacturers do not have a standard reference when producing printers, plotters, print spoolers, or other common peripherals. Consequently, you must acquire the service manual for each peripheral device purchased to determine exactly which signals are utilized in its serial interface.

END

[Return to TOC](#)



**Description:**

*Protocol suite:* PPP.

*Type:* PPP network layer protocol.

*PPP protocol:* 0x0049

*Working groups:* pppext, Point-to-Point Protocol Extensions.

Serial Data Transport Protocol (SDTP) is used for synchronous serial data compression over a PPP link.

Before any SDTP packets may be communicated, PPP must reach the Network-Layer Protocol phase, and the SDTP Control Protocol must reach the Opened state.

The maximum length of the SDTP datagram transmitted over a PPP link is limited only by the negotiated Maximum-Frame-Size and the maximum length of the Information field of a PPP encapsulated packet. Note that if compression is used on the PPP link, this the maximum length of the SDTP datagram may be larger or smaller than the maximum length of the Information field of a PPP encapsulated packet, depending on the particular compression algorithm and protocol used.

RFC 1963, pages 1 - 3:

This document describes a new Network level protocol (from the PPP point of view), PPP Serial Data Transport Protocol, that provides encapsulation and an associated Serial Data Control Protocol (SDCP) for transporting serial data streams over a PPP link. This protocol was developed for the purpose of using PPP's many features to provide a standard method for synchronous data compression. The encapsulation uses a header structure based on that of the ITU-T Recommendation V.120.

This document is a product of the TR30.1 ad hoc committee on compression of synchronous data. It represents a component of a proposal to use PPP to provide compression of synchronous data in DSU/CSUs.

In addition to providing support for multi-protocol datagrams, the Point-to-Point Protocol (PPP) has defined an effective and robust negotiating mechanism that can be used on point to point links. When used in conjunction with the PPP Compression Control Protocol and one of the PPP Compression Protocols, PPP provides an interoperable method of employing data compression on a point-to- point link.

This document provides a PPP encapsulation for serial data, specifying a transport protocol, PPP Serial Data Transport Protocol (PPP-SDTP), and an associated control protocol, PPP Serial Data Control Protocol (PPP-SDCP). When these protocols are added to above mentioned PPP protocols, PPP can be used to provide compression of serial data on a point-to-point link.

This first edition of PPP-SDTP/SDCP covers HDLC-like synchronous serial data and asynchronous serial data. It does this by using a terminal adaption header based on that of ITU-T Recommendation V.120. Support may be added in the future for other synchronous protocols as the marketplace demands.

The V.120 terminal adaption header allows transported data frames to be split over several packets, supports the transport of DTE port idle and error information, and optionally supports the transport of DTE control state information.

In addition to the V.120 Header, fields can be added to the packet format through negotiation to provide support for features not included in the V.120 header. The extra fields are: a Length Field, which is used to distinguish packets in compound frames, and a Port field, which is used to provide multi-port multiplexing capability. The protocol also allows reserved bits in the V.120 header to be used to transport non-octet aligned frames and to provide a flow control mechanism.

To provide these features, PPP-SDTP permits a single frame format to be selected from several possible formats by using PPP-SDCP negotiation. The terminal adaption header can be either fixed length or variable length, to allow either simplicity or flexibility.

The default frame format places the terminal adaption header at the end of the packet. This permits optimal transmitter timelines when user frames are segmented and compression is also used in conjunction with this protocol.

---

### **Packet format:**

---

### **Glossary:**

#### **V.120.**

CCITT Recommendation V.120 (09/92), "Support by an ISDN of Data Terminal Equipment with V-Series Type Interfaces with Provision for Statistical Multiplexing", 1993.

---

### **RFCs:**

[RFC 1963] PPP Serial Data Transport Protocol (SDTP).

---

---

**Disclaimer:** This description is completely unofficial. Most of the information presented here is discovered by me, Eugene Crosser, while snooping the serial line and by trial and error. I never had an official protocol description, have never seen any related software source code, and have never done reverse engineering of any related software. This description may be incomplete, inaccurate or completely wrong. You are warned.

Some information is taken from 'camediaplay' package by Jun-ichiro Itoh <itojun@itojun.org>, from the findings of Thierry Bousch <bousch%linotte.uucp@topo.math.u-psud.fr> Tsuruzoh Tachibanaya <tsuruzoh@butaman.ne.jp> and from other (open) sources and not checked by me.

---

# Serial Protocol of Some Digital Cameras

Several models of digital cameras, namely Epson, Sanyo, Agfa and Olympus cameras, seem to use the same protocol for communication with the host. Follows the description of the high-level protocol they use over the serial line.

## Protocol Basics

The host and the camera exchange with data packets and individual bytes. Serial line parameters used are: 8bit, no parity. No flow control is used. All arithmetic data is transmitted least significant byte first ("little endian").

## Protocol Elements

The elementary units of the protocol are:

Initialization Byte	NUL	0x00
Action Complete Notification	ENQ	0x05
Positive Acknowledgement	ACK	0x06
Unable to Execute Command	DC1	0x11
Negative Acknowledgement, also Camera Signature	NAK	0x15
Packet	Variable length sequence of bytes	
Termination Byte		0xff

## Packet structure

The packet has the following structure:

Offset	Length	Meaning
0	1	Packet type
1	1	Packet subtype/sequence
2	2	Length of data
4	variable	Data
-2	2	checksum

Known packet types are:

Type	Description
0x02	Data packet that is not last in sequence
0x03	Data packet that is last in sequence
0x1b	Command packet

Data packets that are sent in response to a single command are numbered starting from zero. If all requested data fits in one packet, it has type 0x03 and sequence 0.

Command packet has subtype 0x43 or 0x53. Only the first command packet in a session has subtype 0x53.

Maximum length of data field in a packet is 2048 bytes, which yields in 2054 total packet length.

Checksum is a simple 16 bit arithmetic sum of all bytes in the data field. As already mentioned above, length and checksum values are transmitted least significant byte first.

## Flow of Control

A communication session flow is as follows:

Host	Camera
Port speed set to 19200 baud	
Host sends init byte 0x00	
	Camera responds with signature 0x15
Host sends command packet with subtype 0x53 and "set speed" command	
	Camera sends ACK 0x06
Port speed set to the new value	
Host sends command	
	Camera responds with either ACK plus optionally "action taken" notifier or data packet sequence
Host sends ACK to every data packet	
... Command - reply cycle repeated ...	
	Camera sends 0xff and resets after a few seconds (value is model-dependant) of inactivity

If the camera does not respond to a command in reasonable time, or responds with a NAK, the command can be resent. If the camera does not provide a complete data packet in reasonable time, or the data packet is corrupt (checksum does not match), the host can request resending of the packet by sending NAK instead of ACK.

## Command format and codes

Command is a sequence of bytes sent in the data field of a command packet. Command format is as follows:

Offset	Length	Description
0	1	Command code
1	1	Register number or subcode
2	variable	Optional argument

Five command codes are known:

Code	Argument	Description
0	int32	Set value of integer register
1	none	Read value of integer register
2	vdata	Take action unrelated to registers
3	vdata	Set value of vdata register
4	none	Read value of vdata register

Commands 0 and 3 are replied with a single ACK 0x06. Command 2 is replied with an ACK 0x06 followed by an "action complete" notifier 0x05. Commands 1 and 4 are replied with a sequence of data

packets, each of them must be ACK'ed by the host.

Command 0 must be issued with a 4 byte argument containing the new value for the register (bytes in "LSB first" order). Command 2 typically is issued with a single zero byte as an argument. Command 3 is issued with an argument of variable number of bytes. If this is a printable string, it should **not** include the trailing zero byte.

Camera replies to the command 1 with a single data packet containing 4 bytes of a 32bit integer (in "LSB first" order). Camera replies to the command 4 with a sequence of data packets with variable number of data bytes. Note that if a printable string is returned, it **is** terminated with a zero byte, and thus may be safely printed or otherwise treated as a normal C language character string.

## Registers

The following registers are known (read/writability info is inaccurate):

No.	Type	R/W	Description
1	int32	R/W	Resolution: 1 - Std, 2 - Hi, 3 - Ext, other values possible
2	int32	R/W	Clock in UNIX time_t format
3	int32	R/W	Shutter speed (microseconds), 0 - auto
4	int32	W	Current frame number (or animation number if hi order byte is 0xff)
5	int32	R/W	Aperture: 0 - Auto, 1 - Low, 2 - Med, 3 - ?, 4 - Hi
6	int32	R/W	Color mode: 1 - Color, 2 - B/W
7	int32	R/W	Flash mode: 0 - Auto, 1 - Force, 2 - Off, 3 - Anti RedEye, 4 - Slow sync
8	int32	R/W	Unknown (128)
9	int32	R/W	Unknown (128)
10	int32	R	No. of frames in current folder
11	int32	R	No. of frames left
12	int32	R	Length of current frame *
13	int32	R	Length of current thumbnail *
14	vdata	R	Current frame data *
15	vdata	R	Current thumbnail data *
16	int32	R	Battery capacity percentage
17	int32	R/W	Communication speed 1 - 9600 .. 5 - 115200, 6 - 230400, 256 - 9600 .. 264 - 911600 (sync?)
18	int32	R	Unknown (1)
19	int32	R/W	Bright/Contrast: 0 - Normal, 1 - Contrast+, 2 - Contrast-, 3 - Brightnes+, 4 - Brightnes-
20	int32	R/W	White balance: 0 - Auto, 1 - Sunny, 2 - Incandescent, 3 - Fluorescent, 5 - Flash, 6- White preset, 255 - Cloudy
21	vdata	R/W	Unused

22	vdata	R/W	Camera I.D.
23	int32	R/W	Autoshut on host timer (seconds)
24	int32	R/W	Autoshut in field timer (seconds)
25	vdata	R/W	Serial No. (string)
26	vdata	R	Version
27	vdata	R/W	Model
28	int32	R	Available memory left
29	vdata	R/W	Upload image data to this register
30	int32	W	LED: 0 - Off, 1 - On, 2 - Blink
31	vdata	R	Unknown ("\0")
32	int32	R	Put "magic spell" 0x0FEC000E here before uploading image data
33	int32	R/W	Focus mode: 1 - Macro, 2 - Normal, 3 - Infinity/fisheye
34	int32	R	Operation mode: 1 - Off, 2 - Record, 3-Play, 6-Thumbnail
35	int32	R/W	LCD brightness 1 to 7
36	int32	R	Unknown (3)
37	vdata	R	Unknown ("\0")
38	int32	R	LCD autoshut timer (seconds)
39	int32	R	Protection state of current frame *
40	int32	R	True No. of frames taken
41	int32	R/W	LCD date format: 1 - 'YY MM DD, 2 - DD MM 'HH
42	vdata	R	Unknown ("")
43	vdata	R	Audio data description block * 0: expanded .wav length 1: compressed .wav length 3: Unknown (0) 4: Unknown (0) 5: Unknown (0) 6: Unknown (0) 7: Unknown (0)
44	vdata	R	Audio data *
45	vdata	R	Unknown ("")
46	vdata	R	Camera summary data: 32 bytes with copies of 8 other registers 0: Reg 1 (Resolution) 1: Reg 35 (LCD brightness) or Reg 7 (Flash mode) 2: Reg 10 (Frames taken) or Unknown 3: Unknown (0) or Unknown 4: Unknown (0) or Reg 16 (Battery capacity) 5: Unknown (0) or Reg 10 (Frames taken) 6: Unknown (0) or Reg 11 (Frames left) 7: Number of animations taken

47	vdata	R	Picture summary data: 32 bytes or 8 int32's * 0: Hi order byte: unknown, next 3 bytes: Length of current image 1: Length of current thumbnail 2: Audio data length (expanded) 3: Resolution 4: Protection state 5: TimeDate 6: Unknown (0) 7: Animation type: 1 - 10ms, 2 - 20ms
48	vdata	R	Manufacturer
49	vdata	R	Unknown ("")
50	int32	R	Unknown (0)
51	int32	R/W	Card detected: 1 - No, 2 - Yes
52	vdata	R/W	Unknown ("")
53	int32	R/W	Language: 3 - english, 4 - french, 5 - german, 6 - italian, 8 - spanish, 10 - dutch
54-59	vdata	R	Unknown ("")
60	int32	R	True No. of frames taken
61-68	vdata	R	Unknown ("")
69	vdata	R	Exposure Compensation 8 bytes 0: compensation value -20 to +20 1: 0 2: 0 3: 0 4: 10 5: 0 6: 0 7: 0
70	int32	R/W	Exp. meter: 2 - Center weighted, 3 - Spot, 5 - Multi element matrix
71	vdata	R/W	Effective zoom in tenths of millimeters: 8 bytes 0: LSB 1: MSB 2: 0 3: 0 4: 10 5: 0 6: 0 7: 0
72	int32	R/W	Bitmap: 1 - AEL/WBL, 2 - Fisheye, 4 - Wide, 8 - Manual zoom, 16 - B/W, 256 - 1.25x, 512 - 1.6x, 768 - 2.0x, 1024 - 2.5x, 1280 - off
73-76	vdata	R	Unknown ("")
77	int32	W	Size of data packet from camera (default 0x800)
78	vdata	R	Unknown ("")



79	vdata	R	Filename of current frame *
80-81	vdata	R	Unknown ("")
82	int32	W	Unknown (enable folder features? Write 60 here)
83	int32	R/W	Folder navigation When read, return number of folders on the card. When written <b>without</b> data, reset folder system (?) Or select current folder by its number
84	vdata	R/W	Current folder name (may read or set)
85	vdata	R	Unknown ("")
86	int32	R/W	Digital zoom; 0 - 1X, otherwise zoom factor x 100 (i.e. in percent)
87-90	vdata	R	Unknown ("")
91	vdata	R	Current folder I.D. and name

\* **Note:** Marked registers only become useful for reading after setting register 4. If value of 0 assigned to register 4 after doing action 5, subsequent retrieval of picture data gives the "live preview".

For command 2, the second byte is action code not register number. The following action codes are known:

Code	Argument	Description
0	single zero byte	Erase last picture
1	single zero byte	Erase all pictures (but not animations)
2	single zero byte	Take picture
4	single zero byte	Finish session immediately
5	single zero byte	Take preview snapshot (retrievable as frame zero)
6	single byte	Calibration / testing. Arg value: 1 Calibrate autofocus 3 Test zoom/exposure 4-6 Store 0 in Reg 32 9 Load LCD Brightness (0-31) from Reg 32 10 Load LCD size (25 for Nikon Coolpix 950) from Reg 32 11 LCD Saturation (0-32) from Reg 32 13 LCD Red-Green (0-32) from Reg 32 14 LCD Blue (0-32) from Reg 32 15 Store -1 in Reg 32 16 Multi shot (locks up if lcd is on) 17 Take picture 18 Store -1 in Reg 32 20-23 locks up if lcd is on 24-255 Store -1 in Reg 32
7	single zero byte	Erase current frame *

8	single byte	Switch LCD mode. Arg value: 1 - Off 2 - Record 3 - Play (show current frame fullscreen) 4 - preview thumbnails (?) 5 - Thumbnail view (smaller?) 6 - Thumbnail view (larger?) 7 - Next 8 - Previous
9	single byte	Set protection state of current frame to the value of parameter (binary 0 or 1)*
11	single zero byte	Store freshly uploaded image into NVRAM (see appendix A)
12	single byte	LCD test. Arg value: 0 - white 1 - gray 2 - black 3 - red 4 - green 5 - blue 6 - test pattern

\* **Note:** actions 7 and 9 only useful after setting register 0x04.

## Appendix A

Date: Sun, 14 Jul 2002 01:28:39 +0200 (CEST)  
From: =?iso-8859-1?Q?Peter\_=C5strand?= <astrand(at)lysator.liu.se>  
To: allyn(at)fratkin.com, <wolfgang(at)charlotte.wsrcc.com>, <crosser(at)average.org>  
Subject: Upload on Olympus C-860L

FYI.

Tonight, I've been struggling with uploading arbitrary pictures to my Olympus C-860L. I've finally found out that for the camera to accept the picture, two conditions must be met:

- 1) The subsampling must be 2x1, 1x1, 1x1
- 2) The EXIF info must be just like the pictures the camera itself produces.

So, I've made a small script to fix this. Feel free to include it in FAQs and/or photopc dists.

/more/data/pics/olympus-reference-pic.jpg is just some picture taken with the camera.

photopc-upload-all:

```
#!/bin/sh
```

```
TMPFILE=`mktemp /tmp/photopc-upload.XXXXXX` || exit 1
```

```
for file in $@; do
  echo Converting $file...
  djpeg $file | cjpeg -sample 2x1 > $TMPFILE
  jhead -te /more/data/pics/olympus-reference-pic.jpg $TMPFILE
  echo Uploading $file...
  photopc upload $TMPFILE
  sleep 2;
done

rm -f $TMPFILE

--
/Peter Åstrand <astrand(at)lysator.liu.se>
```

---

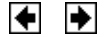
## Appendix B

Some Nikon models support an extension to the protocol described above, specifically designed for remote control units. This protocol allows to control zoom, emulate half-depress of the shutter release button, bulb operation and possibly more. <vladimir.vyskocil(at)wanadoo.fr> compiled a partial description of this protocol, available here.

---

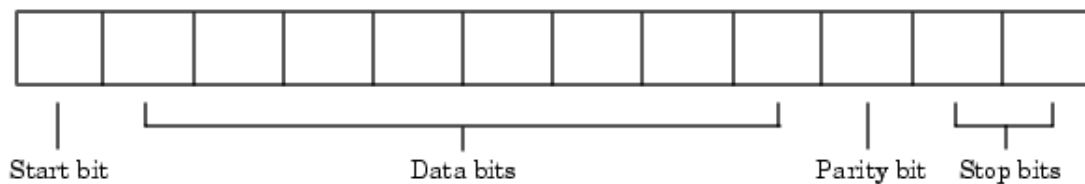
Please mail your corrections/additions to <crosser at average dot org>  
See <http://photopc.sourceforge.net/> for possible updates.

---



## Serial Data Format

The serial data format includes one start bit, between five and eight data bits, and one stop bit. A parity bit and an additional stop bit might be included in the format as well. The diagram below illustrates the serial data format.



The format for serial port data is often expressed using the following notation

- number of data bits - parity type - number of stop bits

For example, 8-N-1 is interpreted as eight data bits, no parity bit, and one stop bit, while 7-E-2 is interpreted as seven data bits, even parity, and two stop bits.

The data bits are often referred to as a *character* because these bits usually represent an ASCII character. The remaining bits are called *framing bits* because they frame the data bits.

## Bytes Versus Values

The collection of bits that comprise the serial data format is called a *byte*. At first, this term might seem inaccurate because a byte is 8 bits and the serial data format can range between 7 bits and 12 bits. However, when serial data is stored on your computer, the framing bits are stripped away, and only the data bits are retained. Moreover, eight data bits are always used regardless of the number of data bits specified for transmission, with the unused bits assigned a value of 0.

When reading or writing data, you might need to specify a *value*, which can consist of one or more bytes. For example, if you read one value from a device using the `int32` format, then that value consists of four bytes. For more information about reading and writing values, refer to [Writing and Reading Data](#).

## Synchronous and Asynchronous Communication

The RS-232 standard supports two types of communication protocols: synchronous and asynchronous.

Using the synchronous protocol, all transmitted bits are synchronized to a common clock signal. The two devices initially synchronize themselves to each other, and then continually send characters to stay synchronized. Even when actual data is not really being sent, a constant flow of bits allows each device to know where the other is at any given time. That is, each bit that is sent is either actual data or an idle character. Synchronous communications allows faster data transfer rates than asynchronous methods, because additional bits to mark the beginning and end of each data byte are not required.

Using the asynchronous protocol, each device uses its own internal clock resulting in bytes that are transferred at arbitrary times. So, instead of using time as a way to synchronize the bits, the data format is used.

In particular, the data transmission is synchronized using the start bit of the word, while one or more stop bits indicate the end of the word. The requirement to send these additional bits causes asynchronous communications to be slightly slower than synchronous. However, it has the advantage that the processor does not have to deal with the additional idle characters. Most serial ports operate asynchronously.

**Note** When used in this guide, the terms "synchronous" and "asynchronous" refer to whether read or write operations block access to the MATLAB command line. Refer to Controlling Access to the MATLAB Command Line for more information.

### How Are the Bits Transmitted?

By definition, serial data is transmitted one bit at a time. The order in which the bits are transmitted is given below:

1. The start bit is transmitted with a value of 0.
2. The data bits are transmitted. The first data bit corresponds to the least significant bit (LSB), while the last data bit corresponds to the most significant bit (MSB).
3. The parity bit (if defined) is transmitted.
4. One or two stop bits are transmitted, each with a value of 1.

The number of bits transferred per second is given by the *baud rate*. The transferred bits include the start bit, the data bits, the parity bit (if defined), and the stop bits.

### Start and Stop Bits

As described in Synchronous and Asynchronous Communication, most serial ports operate asynchronously. This means that the transmitted byte must be identified by start and stop bits. The start bit indicates when the data byte is about to begin and the stop bit(s) indicates when the data byte has been transferred. The process of identifying bytes with the serial data format follows these steps:

1. When a serial port pin is idle (not transmitting data), then it is in an "on" state.
2. When data is about to be transmitted, the serial port pin switches to an "off" state due to the start bit.
3. The serial port pin switches back to an "on" state due to the stop bit(s). This indicates the end of the byte.

## Data Bits

The data bits transferred through a serial port might represent device commands, sensor readings, error messages, and so on. The data can be transferred as either binary data or ASCII data.

Most serial ports use between five and eight data bits. Binary data is typically transmitted as eight bits. Text-based data is transmitted as either seven bits or eight bits. If the data is based on the ASCII character set, then a minimum of seven bits is required because there are  $2^7$  or 128 distinct characters. If an eighth bit is used, it must have a value of 0. If the data is based on the extended ASCII character set, then eight bits must be used because there are  $2^8$  or 256 distinct characters.

## The Parity Bit

The parity bit provides simple error (parity) checking for the transmitted data. The types of parity checking are given below.

**Table 9-2: Parity Types**

Parity Type	Description
Even	The data bits plus the parity bit result in an even number of 1's.
Mark	The parity bit is always 1.
Odd	The data bits plus the parity bit result in an odd number of 1's.
Space	The parity bit is always 0.

Mark and space parity checking are seldom used because they offer minimal error detection. You might choose to not use parity checking at all.

The parity checking process follows these steps:

1. The transmitting device sets the parity bit to 0 or to 1 depending on the data bit values and the type of parity checking selected.
2. The receiving device checks if the parity bit is consistent with the transmitted data. If it is, then the data bits are accepted. If it is not, then an error is returned.

**Note** Parity checking can detect only 1-bit errors. Multiple-bit errors can appear as valid data.

For example, suppose the data bits 01110001 are transmitted to your computer. If even parity is selected, then the parity bit is set to 0 by the transmitting device to produce an even number of 1's. If odd parity is selected, then the parity bit is set to 1 by the transmitting device to produce an odd number of 1's.

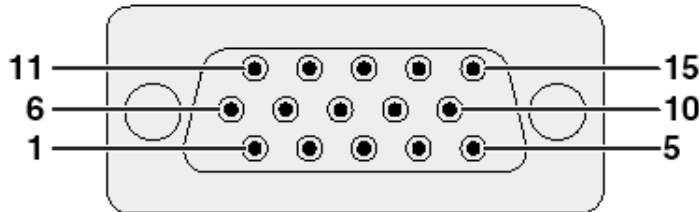


# Connectors

---

## RGB Signal Input Port:

15-pin Mini D-sub female connector



## RGB Input

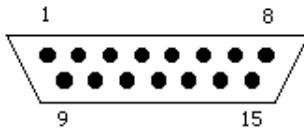
Analog

- |                                      |                            |
|--------------------------------------|----------------------------|
| 1. Video input (red)                 | 8. Earth (blue)            |
| 2. Video input (green/sync on green) | 9. Not connected           |
| 3. Video input (blue)                | 10. GND                    |
| 4. Not connected                     | 11. GND                    |
| 5. Composite sync                    | 12. Bi-directional data    |
| 6. Earth (red)                       | 13. Horizontal sync signal |
| 7. Earth (green/sync on green)       | 14. Vertical sync signal   |
|                                      | 15. Data clock             |

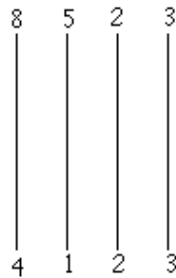
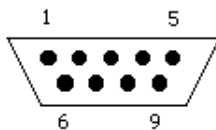
---

Computer - Pioneer DVD9300 player cable

DVD player DB-15 Male



Computer DB-9 female



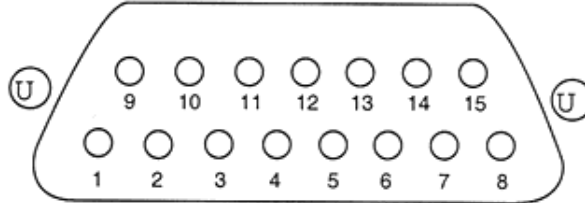


# Pioneer DVD 7300 (7400)

## 2.1 Interface Connector

A computer may be connected to the DVD-V7400 using a 15-pin D-Sub connector (e.g., a JAE DALC-J15SAF connector with suitable plug such as the JAE DA-15PF-N) to the RS-232C serial port or to the parallel port.

The pins are identified below:



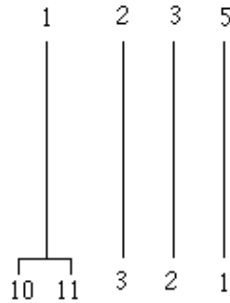
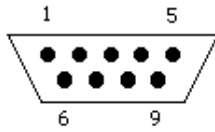
## 2.2 Serial Interface Pin Specification

Pin #	Terminal	Input/Output	Function
1	GND	--	ground
2	TxD	Output	send data
3	RxD	Input	receive data
4	DTR	Output	enable data receiving
5	POWER	Output	external power control
6	SW1	Input	
7	SW2	Input	
8	SW3	Input	
9	SW4	Input	
10	SW5	Input	
11	SW6	Input	
12	SW7	Input	
13	SW8	Input	
14	DLTST	Input	used only for servicing the unit – do not connect
15	V +8V	Output	used only for servicing the unit – do not connect

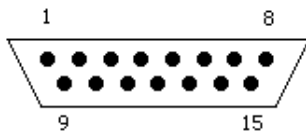
---

Serial controller - DVD player cable

Computer DB-9 male



DVD player DB-15 Male



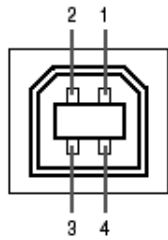
HT200/HT250

*RS-232C Control Port:*

<p>D-SUB 9-pin (female)</p>	Pin No	Signal	Definition
	1	N/A	Not used
	2	TD	Transmit data -
	3	RD	Receive data -
	4	N/A	Not used -
	5	GND	Ground -
	6	N/A	Not used
	7	N/A	Not used
	8	N/A	Not used
	9	N/A	Not used

Parameter	Value
Transfer Rate	19200 b.p.s.
Data Length	8 bits
Parity Bit	None
Stop Bit	1 bit
Flow Control	None

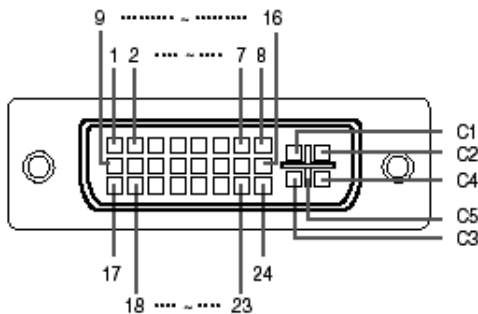
## 4-pin USB connector



### • USB connector: 4 pin B-type USB connector

Pin no.	Signal	Name
1	VCC	USB power
2	USB-	USB data-
3	USB+	USB data+
4	SG	Signal Ground

## DVI Digital / Analog INPUT 1 port : 29 pin connector



### • DVI Digital INPUT

Pin No.	Signal	Pin No.	Signal
1	T.M.D.S data 2-	16	Hot plug detection
2	T.M.D.S data 2+	17	T.M.D.S data 0-
3	T.M.D.S data 2 shield	18	T.M.D.S data 0+
4	Not connected	19	T.M.D.S data 0 shield
5	Not connected	20	Not connected
6	DDC clock	21	Not connected
7	DDC data	22	T.M.D.S clock shield
8	Not connected	23	T.M.D.S clock+
9	T.M.D.S data 1-	24	T.M.D.S clock-
10	T.M.D.S data 1+	C1	Not connected
11	T.M.D.S data 1 shield	C2	Not connected
12	Not connected	C3	Not connected
13	Not connected	C4	Not connected
14	+5V power	C5	Ground
15	Ground		

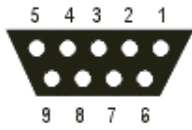
### • DVI Analog RGB Input

Pin No.	Signal	Pin No.	Signal
1	Not connected	16	Hot plug detection
2	Not connected	17	Not connected
3	Not connected	18	Not connected
4	Not connected	19	Not connected
5	Not connected	20	Not connected
6	DDC clock	21	Not connected
7	DDC data	22	Not connected
8	Vertical sync	23	Not connected
9	Not connected	24	Not connected
10	Not connected	C1	Analog input Red
11	Not connected	C2	Analog input Green
12	Not connected	C3	Analog input Blue
13	Not connected	C4	Horizontal sync
14	+5V power	C5	Ground
15	Ground		

### • DVI Analog Component Input

Pin No.	Signal	Pin No.	Signal
1	Not connected	16	Not connected
2	Not connected	17	Not connected
3	Not connected	18	Not connected
4	Not connected	19	Not connected
5	Not connected	20	Not connected
6	Not connected	21	Not connected
7	Not connected	22	Not connected
8	Not connected	23	Not connected
9	Not connected	24	Not connected
10	Not connected	C1	Analog input Pr/Cr
11	Not connected	C2	Analog input Y
12	Not connected	C3	Analog input Pb/Cb
13	Not connected	C4	Not connected
14	Not connected	C5	Ground
15	Ground		

## Serial mouse



Pin	Description
1	DCD Data carried detect
2	RD Receive data
3	TD Transmit data
4	DTR Data terminal ready
5	SG Signal ground
6	DSR Data set ready
7	RTS Request to send
8	CTS Clear to send
9	Ring

---

# Image Dimensions in Common Usage

Collated by Paul Bourke  
May 2000

---

Dimensions	Ratio	Comments
8x8	1	
16x16	1	Macintosh cursor size Supported by Windows icon format
32x32	1	Macintosh icon size Supported by Windows icon format
64x64	1	Supported by Windows icon format
88x31	2.8387	WWW micro banner
128x96	1 1/3	
160x120	1 1/3	NTSC 13" 1/16th CuSeeMe small image size Considered a "small" QuickTime movie
160x144	1.1111	
176x144	1.2222	
180x132	1.3636	
180x135	1 1/3	
192x144	1 1/3	PAL 1/16
234x60	3.9	WWW small banner
256x192	1 1/3	10"/12" 1/4
320x200	1.6	Called CGA, IBM PS/2
320x240	1 1/3	NTSC 13" 1/4 CuSeeMe image size Considered a "large" QuickTime movie
320x288	1.1111	
320x400	0.8	Amigo

352x288	1.2222	PAL Video CD
352x240	1.46666	NTSC Video CD
384x256	1.5	Photo CD
384x288	1 1/3	PAL 1/4
392x72	5.444444	WWW banner
400x300	1 1/3	
460x55	8.36363636	WWW banner
468x32	14.625	WWW banner
468x60	7.8	WWW banner
512x342	1.497	Original Macintosh screen
512x384	1 1/3	
544x372	1.4623	WebTV image size
640x350	1.82857	Called EGA, IBM
640x480	1 1/3	NTSC full Called PGA, IBM VGA
640x576	1.1111	
704x576	1.22222	Full image size from ASUS video capture with TNT2
720x350	2.05714286	Called MDA, IBM / VESA
720x400	1.8	Called MCGA, IBM/VESA
720x480	1.5	Format 480p. NTSC video format as used by DPS PVR video hardware. NTSC DV SDTV
720x483	1.49068323	SDTV
720x484	1.48760331	Alternative Media-100 NTSC, eg: Media-100
720x486	40/27	CCIR 601 NTSC
720x540	1 1/3	CCIR 601 NTSC Sq
720x576	1.25	CCIR 601 DV PAL and DV SECAM

729x348	2.094828	Hercules graphics
768x576 Subtract 75 from left/right and 55 top/bottom for the PAL "safe" area	1 1/3	CCIR 601 PAL full
800x600	1 1/3	Called SVGA resolution. Used in the first generation of LCD projectors (their native resolution).
832x624	1 1/3	Macintosh
856x480	1.78333333	
896x600	1.49333333	
960x720	1 1/3	Non standard digital TV format supported by some suppliers.
1024x768	1 1/3	XGA. Native resolution of many LCD projectors. Used by VisionStation and VisionStation 3.
1080x720	1.5	HDTV
1152x768	1.5	
1152x864	1 1/3	VESA
1152x870	1.3241	Macintosh
1152x900	1.28	Sun / SGI
1280x720	16/9	Format 720p. HDTV WXGA
1280x800	1.6	
1280x854	1.498829	Mac G4 15" laptop
1280x960	1 1/3	SXVGA
1280x992	1.29	Optimised on the PowerStorm 350 OpenGL card/drivers from Compaq
1280x1024	1.25	SXGA
1360x766	1.77545692	

1365x768	16:9 (nearly, 1.77734375)	NEC 61" plasma
1365x1024	1 1/3	VisionStation 3 with upgrade and VisionStation 5.
1400x1050	1.3671875	DELL Laptop
1440x900	1.6	Apple 17" G4 laptop
1520x856	1.77570093	
1600x900	16:9	
1600x1024	1.5625	
1600x1200	1 1/3	VESA UXGA
1792x1120	1.6	
1792x1344	1 1/3	
1824x1128	1.61702128	
1824x1368	1 1/3	
1856x1392	1 1/3	
1920x1080	16/9	1080i format. HDTV, known as 1K
1920x1200	1.6	
1920x1440	1 1/3	
2000x1280	1.5625	QXGA
2048x1152	16:9	
2048x1536	1 1/3	Feature film, known as 2K Also used by DOME display controllers
2048x2048	1	Tiger high resolution display
2500x1340	1.86567	
3072x2252	1.3641	Sometimes used for IMAX (?)
3600x2613	1.37772675	Sometimes used for IMAX (?)
4096x3072	1 1/3	Image size for IMAX 3D rendering, known as 4K
4096x3840	1.0666666	NCSA tiled wall (2001)



## Serial Mouse Data Formats

The Microsoft Serial Mouse format is the defacto standard for serial mice. The Microsoft mouse format allows for only two buttons. Three button mice working in Microsoft mode ignore the middle button.

The data packets are sent at 1200 baud with 1 stop bit and no parity. Each packet consists of 3 bytes. It is sent to the computer every time the mouse changes state (ie. the mouse is moved or the buttons are pressed/released).

	D6	D5	D4	D3	D2	D1	D0
1st byte	1	LB	RB	Y7	Y6	X7	X6
2nd byte	0	X5	X4	X3	X2	X1	X0
3rd byte	0	Y5	Y4	Y3	Y2	Y1	Y0

LB is the state of the left button, 1 = pressed, 0 = released.

RB is the state of the right button, 1 = pressed, 0 = released

X0-7 is movement of the mouse in the X direction since the last packet. Positive movement is toward the right.

Y0-7 is movement of the mouse in the Y direction since the last packet. Positive movement is back, toward the user.

The mouse driver software collects the X and Y movement bits from the different bytes in the packet. All moves are sent as two's complement binary numbers.

Although the Microsoft format only requires 7 data bits per byte, most mice actually send 8-bit data with the most significant bit set to 1. Since the most-significant-bit (D7) is last in the serial data stream, this is the same as sending two stop bits instead of one. The Joymouse sends data packets as shown below.

	D7	D6	D5	D4	D3	D2	D1	D0
1st byte	1	1	LB	RB	Y7	Y6	X7	X6
2nd byte	1	0	X5	X4	X3	X2	X1	X0
3rd byte	1	0	Y5	Y4	Y3	Y2	Y1	Y0

---

[ [Back to Data & Documentation](#) ] [ [Home](#) ]

---

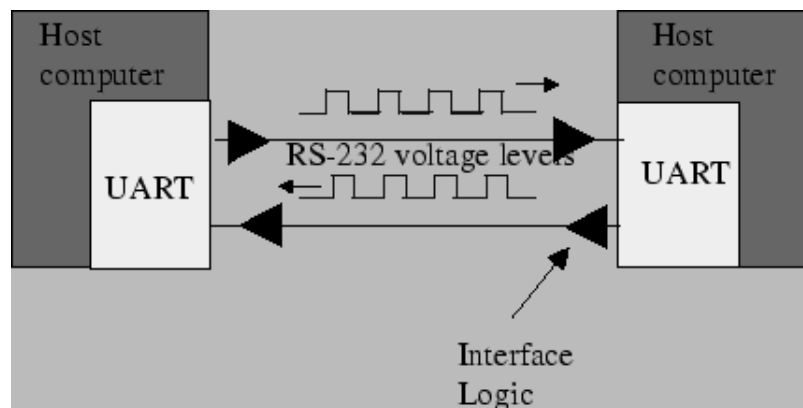


**Next:** Motorola 68HC11 SCI Interface **Up:** Serial Communication **Previous:** Asynchronous Serial Communication (SCI)

## RS-232 Serial Protocol

The RS-232 serial communication protocol is a standard protocol used in asynchronous serial communication. It is the primary protocol used over modem lines. It is the protocol used by the MicroStamp11 when it communicates with a host PC.

Figure 23 shows the relationship between the various components in a serial link. These components are the *UART*, the serial channel, and the interface logic. An interface chip known as the **universal asynchronous receiver/transmitter** or **UART** is used to implement serial data transmission. The UART sits between the host computer and the *serial channel*. The serial channel is the collection of wires over which the bits are transmitted. The output from the UART is a standard TTL/CMOS logic level of 0 or 5 volts. In order to improve bandwidth, remove noise, and increase range, this TTL logical level is converted to an RS-232 logic level of  $-12$  or  $+12$  volts before being sent out on the serial channel. This conversion is done by the interface logic shown in figure 23. In your system the interface logic is implemented by the comm stamp.



**Figure 23:** Asynchronous (RS-232) serial link

A **frame** is a complete and nondivisible packet of bits. A frame includes both *information* (e.g., data and characters) and *overhead* (e.g., start bit, error checking and stop bits). In asynchronous serial protocols such as RS-232, the frame consists of one start bit, seven or eight data bits, parity bits, and stop bits. A timing diagram for an RS-232 frame consisting of one start bit, 7 data bits, one parity bits and two stop bits is shown below in figure 24. Note that the exact structure of the frame must be agreed upon by both transmitter and receiver before the comm-link must be opened.



**Figure 24:** RS-232 Frame (1 start bit, 7 data bits, 1 parity bits,

and 2 stop bits)

Most of the bits in a frame are self-explanatory. The start bit is used to signal the beginning of a frame and the stop bit is used to signal the end of a frame. The only bit that probably needs a bit of explanation is the *parity* bit. Parity is used to detect transmission errors. For *even parity* checking, the number of 1's in the data plus the parity bit must equal an even number. For *odd parity*, this sum must be an odd number. Parity bits are used to detect errors in transmitted data. Before sending out a frame, the transmitter sets the parity bit so that the frame has either even or odd parity. The receiver and transmitter have already agreed upon which type of parity check (even or odd) is being used. When the frame is received, then the receiver checks the parity of the received frame. If the parity is wrong, then the receiver knows an error occurred in transmission and the receiver can request that the transmitter re-send the frame.

In cases where the probability of error is extremely small, then it is customary to ignore the parity bit. For communication between the MicroStamp11 and the host computer, this is usually the case and so we ignore the parity bit.

The *bit time* is the basic unit of time used in serial communication. It is the time between each bit. The transmitter outputs a bit, waits one bit time and then outputs the next bit. The start bit is used to synchronize the transmitter and receiver. After the receiver senses the true-false transition in the start bit, it waits one half bit time and then starts reading the serial line once every bit time after that. The *baud rate* is the total number of bits (information, overhead, and idle) per time that is transmitted over the serial link. So we can compute the baud rate as the reciprocal of the bit time.



**Next:** Motorola 68HC11 SCI Interface **Up:** Serial Communication **Previous:** Asynchronous Serial Communication (SCI)

*Bill Goodwine 2002-09-29*



Next: LCD display for the Up: Serial Communication Previous: RS-232 Serial Protocol

## Motorola 68HC11 SCI Interface

The Motorola 68HC11 supports one SCI. We'll discuss both transmitting and receiving ends of the SCI. The programmer controls the operation of the SCI interface through a set of hardware registers that are memory mapped into the processor's address space. There are 5 control registers shown below in figure 25. This figure also shows the logical names for individual bits in the registers. The BAUD register is used to set the serial link's baud rate. There are two control registers SCCR1 and SCCR2 that specify how the SCI should work. There is a status register, SCSR that the programmer can use to check whether the transmission/reception of a frame has been completed. Finally there is the data register SCDR that holds the transmitted or received information bits.

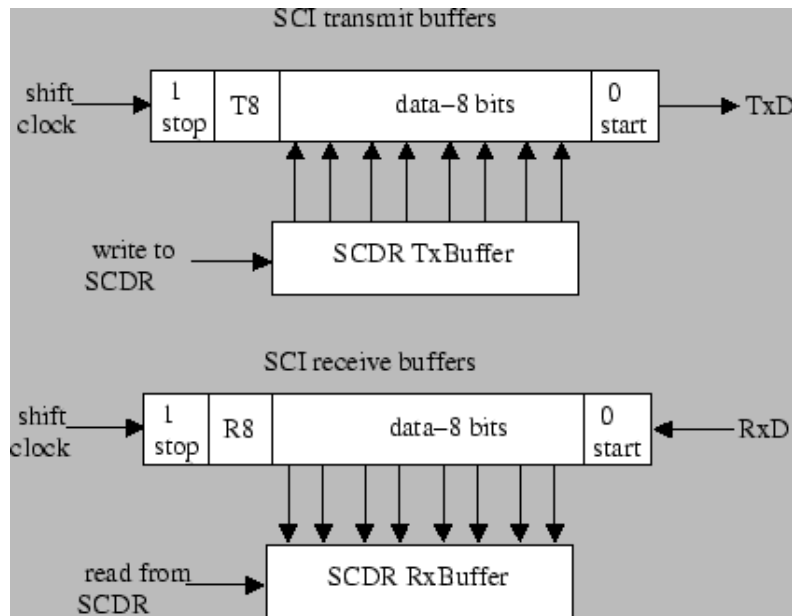
BAUD							
TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0
SCCR1							
R8	T8	0	M	WAK	0	0	0
SCCR2							
TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
SCSR							
TDRE	TC	RDRF	IDLE	OR	NF	FE	0
SCDR							
R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0

Figure 25: 68HC11 SCI Registers

From the number of control bits in the SCCR1 and SCCR2 registers you can see that the programmer has quite a bit of control over the SCI interface. Most of the situations we'll be using, however, have standard set-ups so we won't need to discuss the control register bits in detail. Instead, we'll provide standard functions that encapsulate the user's interface to SCI devices such as a personal computers and simply discuss how these functions work. The proper setup of the SCI subsystem is usually done in your program's `init()` initialization routine.

To understand how the SCI subsystem works, let's examine figure 26. Figure 26 shows how the programmer interacts with the SCI transmit and receive buffers. In order to transmit, the programmer first loads the 8-bit data register SCDR with the data to be sent. The SCI module automatically fills in the start bit, stop bit, and the extra T8 bit from control register SCCR1. The T8 bit can be used as a parity bit.

Once the `SCDR` register is loaded, the subsystem loads this data into the SCI module's transmit buffer. The SCI transmit buffer then holds a single frame and this frame is then clocked out of the transmit register one bit at a time at the rate specified in the `BAUD` register. Once all of the bits have been clocked out of the transmit buffer, the SCI module sets the `TDRE` (transmit data register empty) bit in the SCI's status register `SCSR`. This single bit can then be used to check whether or not the frame has been successfully transmitted.



**Figure 26:** SCI transmit and receive buffers

A similar set of steps can be used to check and see if the receive data register has been filled. Once initialized, the receive data component of the SCI subsystem will wait for the true-to-false transition on the input line signalling a start bit. After the start bit has been detected, the receive subsystem will shift in 10 or 11 bits into the receive data register. The start and stop bits are removed and 8 bits of data are loaded into the SCI data register `SCDR`. The ninth parity bit `R8` is put in the `SCCR1` control register. When the receive data register is full, then the SCI subsystem sets the `RDRF` (receive data register full) flag in the status register `SCSR`. The programmer can then check this status flag to see if a full frame has been received.

The following code segment from an `init()` function can be used to initialize the SCI module to transmit and receive at 38 kbaud. This setup was used in our earlier `kernel.c` functions to send characters back and forth between the MicroStamp11 and the PC.

```
void init(void){
    asm(" sei");
    CONFIG = 0x04;
    BAUD=BAUD38K;
    SCCR1 = 0x00;
    SCCR2 = 0x0C;
    asm(" cli");
}
```

The instructions in this function do the following. The first instruction `CONFIG = 0x04` turns off the

micro-controller's watchdog timer. The second instruction `BAUD=BAUD38K` sets the SCI subsystem's baud rate to 38 kilo-baud. The variable `BAUD38K` is a logical name whose actual value will be found in `hc11.h`. The next two lines set up the SCI subsystem's parameters. By zeroing `SCCR1`, we are ignoring the parity bit and creating a 10-bit frame. By setting `SCCR2=0x0C`, we've disabled all transmit and receive interrupts and we've enabled the transmit and receive modules in the SCI subsystem.

Note that the SCI module has hardware interrupts associated with the hardware events

- Transmission Complete (TC): which is set when the transmit shift register is empty.
- Transmit Data Register Empty (TDRE): which is set when the transmit data register is empty.
- Receive Data Register Full (RDRF): which is set when the receive data register is full.
- Idle line (ILIE): which is when the receive module detects an idle line.

These hardware interrupts can be used to do parity-bit processing on a transmitted or received frame. In our particular examples, however, we assume no parity checking so these interrupts have been disabled.

As specific examples of how the SCI interface can be used, we consider the two function `InChar()` and `OutChar()`. These functions are used to receive and transmit, respectively, a single byte (frame) of data.

```
void OutChar(char data){
    while((SCSR & TDRE) == 0);
    SCDR=data;
}

void InChar(void){
    while((SCSR & RDRF) == 0);
    return(SCDR);
}
```

The first function, `OutChar()` transmits a single byte. The function simply waits in a loop until the `TDRE` bit (transmit data register empty) is set. Once this is done, we know that any previously loaded bytes have been successfully shifted out of the transmit data register. The function then reloads the SCI's data register `SCDR` with the new data. The other function `InChar()` receives a single byte. The function waits in a while loop until the `RDRF` bit (receive data register full) is set, thereby indicating that 8-bits of have been shifted into the receive data register. Once this is done, the function returns a pointer to the `SCDR` data register.



**Next:** LCD display for the **Up:** Serial Communication **Previous:** RS-232 Serial Protocol

*Bill Goodwine 2002-09-29*

**Next:** Framing Error **Up:** What is a UART? **Previous:** What is a UART?

## Serial Data Format

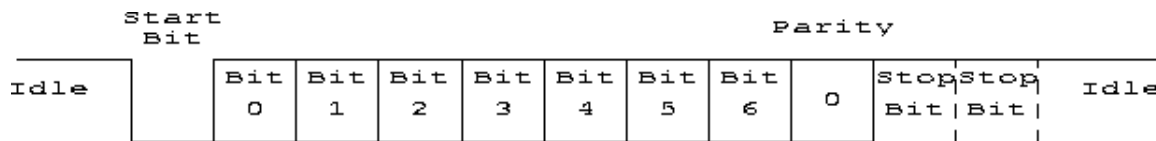
The serial data that we are interested in sending to and from the terminal is byte-wide ASCII data. ASCII is a standard code for sending alphanumeric data and is actually only 7-bits wide. The 8th bit is often used to indicate the parity of the 7-bit data word and used for error detection. For our circuit, the high-order bit will always be 0, but you should always send it anyway. So, each packet that is sent will consist of 8 bits, 7-bits of ASCII and one 0 in the high-order bit. A table of the ASCII code is shown in Figure 1.

$B_3B_2B_1B_0$	$B_6B_5B_4$							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	'	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	—
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

**Figure 1:** ASCII Character Codes

Once we agree to send ASCII, we should also agree on how that data should be sent as a serial data stream. The protocol we will use defines that the bits in the byte are passed least-significant bit first in a serial stream. So, send the bits one at a time, starting with the least-significant. How long each bit is asserted in this serial stream depends on how fast your baud rate is. At 9600 baud, for example, each bit will be asserted for 1/9600 of a second, or about 104  $\mu$  sec. Now the problem is how to decide when to look at the data wire in order to see the bit. The DCE and DTE will not be synchronized to a common clock, so in order to decide when to look at the data line to see new data being passed, they must synchronize with each new byte that is being passed. This is why the UART is “asynchronous” in operation. The data is being passed at a known frequency, but the starting time of each new byte is unknown. So, the receiving circuit must resynchronize at the start of each new byte.

In practice this is quite easy. You only need some sort of protocol that tells you when to expect new data. For our system (and most asynchronous serial protocols in general) the data line must be held in a 1 state (+5v in our case) until a byte is ready to be passed. When a byte is to be sent, the data line drops to 0 (gnd in our case) for one bit time to signal that a byte will follow. This is the *start bit* and its purpose is to wake up the receiver and alert it to the byte that is about to be sent. The 8 data bits then follow, least significant bit first, each asserted for one bit time (which depends on the baud rate). Finally, one or two *stop bits* are sent to indicate the end of the byte. Stop bits are 1-bits that are asserted for one bit time each. In our system the receiver will assume that only one stop bit is sent, and the sender will send two stop bits. This is the most general and safest solution. For example, if the receiver expects a single stop bit and two are sent, nothing bad happens except that some extra time elapses between that byte and the next due to the extra stop bit. On the other hand, if the sender sends only a single stop bit but the external receiver expects two, a mistake might be made. So, it's safer to expect only a single stop bit, but always send two (note that in practice, most terminals and other communication devices have settings to control how many stop bits are sent or expected.) A picture of this data protocol is shown in Figure 2.



**Figure 2:** Data Byte Transmission Format

---

[Next](#) [Up](#) [Previous](#)

**Next:** Framing Error **Up:** What is a UART? **Previous:** What is a UART?

*Erik Brunvand*

*Tue Apr 11 15:50:32 MDT 2000*



# PostScript Tutorial

Written by Paul Bourke  
Original November 1990. Last updated December 1998

---

## Introduction

Postscript is a programming language that was designed to specify the layout of the printed page. Postscript printers and postscript display software use an interpreter to convert the page description into the displayed graphics.

The following information is designed as a first tutorial to the postscript language. It will concentrate on how to use postscript to generate graphics rather than explore it as a programming language. By the end you should feel confident about writing simple postscript programs for drawing graphics and text. Further information and a complete specification of the language can be obtained from **The Postscript Language Reference Manual** from Adobe Systems Inc, published by Addison-Wesley, Reading, Massachusetts, 1985.

Why learn postscript, after all, many programs can generate it for you and postscript print drivers can print to a file? Some reasons might be:

- Having direct postscript output can often result in much more efficient postscript, postscript that prints faster than the more generic output from printer drivers.
- There are many cases where generating postscript directly can result in much better quality. For example when drawing many types of fractals where high resolution is necessary, being able to draw at the native high resolution of a postscript printer is desirable.
- It isn't uncommon for commercial packages to make errors with their postscript output. Being able to look at the postscript and make some sense of what is going on can sometimes give insight on how to fix the problem.

## The Basics

Postscript files are (generally) plain text files and as such they can easily be generated by hand or as the output of user written programs. As with most programming languages, postscript files (programs) are intended to be, at least partially, human-readable. As such, they are generally free format, that is, the text can be split across lines and indented to highlight the logical structure.

Comments can be inserted anywhere within a postscript file with the percent (%) symbol, the comment applies from the % until the end of the line.

While not part of the postscript specification the first line of a postscript file often starts as %!. This is so that spoolers and other printing software detect that the file is to be interpreted as postscript instead of a plain text file. The inline example below will not include this but the postscript files linked from this page will include it since they are design for direct printing.

The first postscript command to learn is **showpage**, it forces the printer to print a page with whatever is currently drawn on it. The examples given below print on single pages and therefore there is a showpage at the end of the file in each example, see the comments later regarding EPS.

## A Path

A path is a collection of, possibly disconnected, lines and areas describing the image. A path is itself not drawn, after it is specified it can be stroked (lines) or filled (areas) making the appropriate marks on the page. There is a special type of path called the clipping path, this is a path within which future drawing is constrained. By default the clipping path is a rectangle that matches the border of the paper, it will not be changed during this tutorial.

## The Stack

Postscript uses a stack, otherwise known as a LIFO (Last In First Out) stack to store programs and data. A postscript interpreter places the postscript program on the stack and executes it, instructions that require data will read that data from the stack. For example, there is an operator in postscript for multiplying two numbers, **mul**. it requires two arguments, namely the two numbers that are to be multiplied together. In postscript this might be specified as

```
10 20 mul
```

The interpreter would place 10 and then 20 onto the stack. The operator **mul** would remove 20 and then 10 from the stack, multiply them together and leave the result, 200, on the stack.

## Coordinate system

Postscript uses a coordinate system that is device independent, that is, it doesn't rely on the resolution, paper size, etc of the final output device. The initial coordinate system has the x axis to the right and y axis upwards, the origin is located at the bottom left hand corner of the page. The units are of "points" which are 1/72 of an inch long. In other words, if we draw a line from postscript coordinate (72,72) to (144,72) we will have a line starting one inch in from the left and right of the page, the line will be horizontal and be one inch long.

The coordinate system can be changed, that is, scaled, rotated, and translated. This is often done to form a more convenient system for the particular drawing being created.

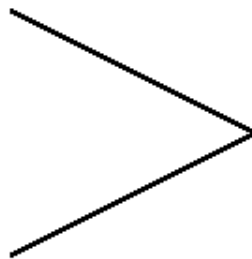
## Basic Drawing Commands

Time to draw something. The following consists of a number of operators and data, some operators like **newpath** don't need arguments, others like **lineto** take two arguments from the stack. All the examples in this text are shown as postscript on the left with the resulting image on the right. The text on the left also acts as a link to a printable form of the postscript file.

```

newpath
100 200 moveto
200 250 lineto
100 300 lineto
2 setlinewidth
stroke

```



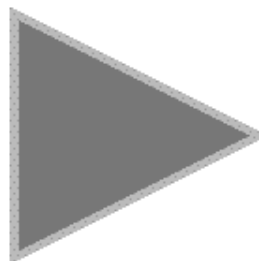
There are also a relative moveto and lineto commands, namely, **rmoveto** and **rlineto**.

In this next example a filled object will be drawn in a particular shade, both for the outline and the interior. Shades range from 0 (black) to 1 (white). Note the **closepath** that joins the first vertex of the path with the last.

```

newpath
100 200 moveto
200 250 lineto
100 300 lineto
closepath
gsave
0.5 setgray
fill
grestore
4 setlinewidth
0.75 setgray
stroke

```



The drawing commands such as **stroke** and **fill** destroy the current path, the way around this is to use **gsave** that saves the current path so that it can be reinstated with **grestore**.

## Text

Text is perhaps the most sophisticated and powerful aspect of postscript, as such only a fraction of its capabilities will be discussed here. One of the nice things is that the way characters are placed on the page is no different to any other graphic. The interpreter creates a path for the character and it is then either stroked or filled as per usual.

```

/Times-Roman findfont
12 scalefont
setfont
newpath
100 200 moveto
(Example 3) show

```

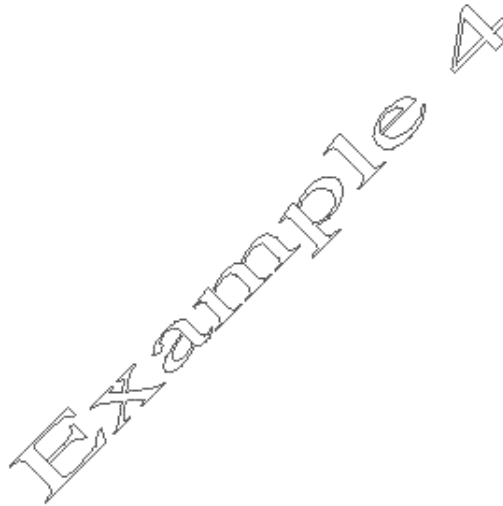
Example 3

As might be expected the position (100,200) above specifies the position of the bottom left corner of the text string. The first three lines in the above example are housekeeping that needs to be done the first time a font is used. By default the font size is 1 point, scalefont then sets the font size in units of points

(1/72 inch). The brackets around the words "Example 3" indicate that it is a string.

A slightly modified version of the above uses **charpath** to treat the characters in the string as a path which can be stroked or filled.

```
/Times-Roman findfont
32 scalefont
setfont
100 200 translate
45 rotate
2 1 scale
newpath
0 0 moveto
(Example 4) true charpath
0.5 setlinewidth
0.4 setgray
stroke
```

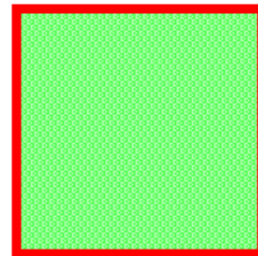


You should make sure you understand the order of the operators above and the resulting orientation and scale of the text, procedurally it draws the text, **scale** the y axis by a factor of 2, **rotate** counter clockwise about the origin, finally **translate** the coordinate system to (100,200).

## Colour

For those with colour LaserWriters the main instruction of interest that replaces the **setgray** is previous examples is **setrgbcolor**. It requires 3 arguments, the red-green-blue components of the colour each varying from 0 to 1.

```
newpath
100 100 moveto
0 100 rlineto
100 0 rlineto
0 -100 rlineto
-100 0 rlineto
closepath
gsave
0.5 1 0.5 setrgbcolor
fill
grestore
1 0 0 setrgbcolor
4 setlinewidth
stroke
```



## Programming

As mentioned in the introduction postscript is a programming language. The extend of this language will

not be covered here except to show some examples of procedures that can be useful to simplify postscript generation and make postscript files smaller.

Lets assume one needed to draw lots of squares with no border but filled with a particular colour. One could create the path repeatedly for each one, alternatively one could define something like the following.

```
/csquare {  
  newpath  
  0 0 moveto  
  0 1 rlineto  
  1 0 rlineto  
  0 -1 rlineto  
  closepath  
  setrgbcolor  
  fill  
} def  
  
20 20 scale  
  
5 5 translate  
1 0 0 csquare  
  
1 0 translate  
0 1 0 csquare  
  
1 0 translate  
0 0 1 csquare
```



This procedure draws three coloured squares next to each other, each 20/72 inches square, note the **scale** of 20 on the coordinate system. The procedure draws a unit square and it expects the RGB colour to be on the stack. This could be used as a method (albeit inefficient) of drawing a bitmap image.

Even if one is simply drawing lots of lines on the page, in order to reduce the file size it is common to define a procedure as shown below. It just defines a single character "l" to draw a line segment, one can then use commands like 100 200 200 200 l" to draw a line segment from (100,200) to (200,200).

```
/l { newpath moveto lineto stroke } def
```

## Some other useful Commands

The following are some other commonly used commands along with a brief description, again you should consult a reference manual for the entire set of commands.

- arc**            Draws an arc (including a circle). The arguments are xcenter, ycenter, radius, start angle, stop angle. The arc is drawn counterclockwise, the angles are in units of degrees.
- centerpoint**   This is an example of an instruction that takes no arguments but leaves numbers on the stack, namely the coordinates of the current point.
- setdash**        This sets the dash attribute of a line in terms of a mark-space array. Just as strings are denoted by round braces (), arrays are denoted by square braces []. For example the following command "[3 3] 0 setdash" would make any following lines have a 3 unit dash followed by a 3 unit space. The argument after the dash array is the offset for the start of

the first dash.

### **setlinecap**

This specifies what the ends of a stroked line look like. It takes one argument which may be 0 (butt caps), 1 (round caps), or 2 (extended butt caps). The radius of round caps and the extension of the butt caps is determined by the line thickness.

```
/LINE {
    newpath
    0 0 moveto
    100 0 lineto
    stroke
} def

100 200 translate
10 setlinewidth 0 setlinecap 0 setgray LINE
1 setlinewidth 1 setgray LINE

0 20 translate
10 setlinewidth 1 setlinecap 0 setgray LINE
1 setlinewidth 1 setgray LINE

0 20 translate
10 setlinewidth 2 setlinecap 0 setgray LINE
1 setlinewidth 1 setgray LINE
```



### **setlinejoin**

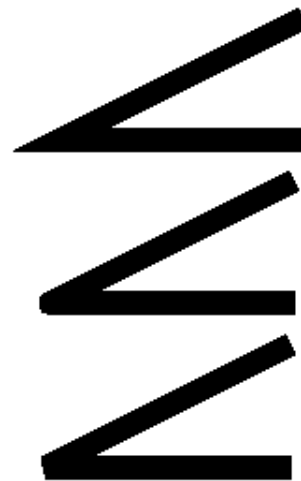
This determines the appearance of joining lines. It takes one argument which may be 0 (miter join), 1 (round join), or 2 (bevel join).

```
/ANGLE {
    newpath
    100 0 moveto
    0 0 lineto
    100 50 lineto
    stroke
} def

10 setlinewidth
0 setlinejoin
100 200 translate
ANGLE

1 setlinejoin
0 70 translate
ANGLE

2 setlinejoin
0 70 translate
ANGLE
```



### **curveto**

This draws a bezier curve through the three points given as arguments. The curve starts at the first point, end at the last point, and the tangents are given by the line between the first-second and second-third pair.

### **save and restore**

Instead of having to "undo" changes to the graphics state it is possible using **save** to push the entire graphics state onto the stack and then reinstate it later with a **restore**.



```
>}  
image
```

## 24 Bit RGB Colour

RGB images with 8 bits per pixel can be represented in postscript using the command **colorimage** which is very similar to the **image** command. In the following example the image is 32 pixels wide by 38 pixels tall.

```
100 200 translate  
32 38 scale  
32 38 8 [32 0 0 -38 0 38]  
{<  
1dfb0023fb002afb0031fb0037fb003ffb00  
66fb006cfb0073fb0079fb0081fb0086fb00  
adfb00b5fb00bbfb00c3fb00c8fb00cffb00  
23f5002af50031f50037f5003ff50044f500
```

...cut...

```
3807003f08004508004c0800520800590800  
8108008608008d07009308009a0700a20800  
c90800d00800d60800dd0800e40700ea0700  
>}  
false 3 colorimage
```



## What is EPS?

EPS (Encapsulated PostScript) is normal postscript with a few restrictions and a few comments in a specified format that provides more information about the postscript that follows. It was design to make it easier for applications to include postscript generated elsewhere within their own pages. The full specification can be obtained from Adobe but in order to make a postscript file DSC (Adobe's Document Structuring Convention) compliant the following must be true:

- There shouldn't be a **showpage**, since EPS is designed to be included inside other documents a **showpage** would obviously ruin the intended effect. In reality most programs that import EPS redefine **showpage** so that if it does exist it doesn't cause problems, a common definition is `"/showpage { } def"`
- The file should consist of just one page.
- The first line of the file should be `"%!PS-Adobe EPSF-3.0"`
- There must be a correctly formed BoundingBox comment, this looks like `%%BoundingBox: xmin ymin xmax ymax` and tells application that plans to include the postscript how large the image is.
- The file should not use any operators that change the global drawing state. In particular the following command may not be used:

banddevice	exitserver	initmatrix	setshared
clear	framedevice	quit	startjob
cleardictstack	grestoreall	renderbands	copypage
initclip	setglobal	initgraphics	setpagedevice
erasepage	nulldevice	sethalftone	setscreen



setgstate          setmatrix          settransfer          undefinefont

- The stack must be left EXACTLY in the same state at the end of the EPS file as it was at the start of the EPS file.
- The lines in EPS files cannot exceed 255 characters in length.

Perhaps most importantly, since usually an application that supports postscript file insertion doesn't have the full postscript interpreter, an EPS file generally has a preview image associated with it. The application dealing with the EPS file can display the preview in the user interface giving a better idea what will print. It should be noted that EPS previews are one of the more machine/OS dependent aspects of EPS.

## Frequently Used Comments

Comments can of course be added anywhere and they will be ignored by the interpreter. There are some standard comments the most common of which are listed below. The text within the square brackets should be replaced with the appropriate text for the file in which they appear (without the []).

```
%!PS-Adobe-3.0 EPSF-3.0
%%Creator: [generally the program that generated the postscript]
%%Title: [descriptive name or just the file name]
%%CreationDate: [date the file was created]
%%DocumentData: Clean7Bit
%%Origin: [eg: 0 0]
%%BoundingBox: xmin ymin xmax ymax
%%LanguageLevel: 2 [could be 1 2 or 3]
%%Pages: 1
%%Page: 1 1
%%EOF
```

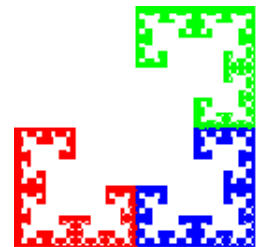
## Drawing "large" images

Due to line length and other restrictions, turning 'large' bitmaps into postscript requires a modification to the methods discussed earlier. The following will describe the most general case of representing a 24 bit RGB colour image as an EPS file. While inefficient this can also be used for greyscale and even black and white images. In the following code "width" and "height" should be replaced with the numbers appropriate to the image.

```
%!PS-Adobe-3.0 EPSF-3.0
%%Creator: someone or something
%%BoundingBox: 0 0 width height
%%LanguageLevel: 2
%%Pages: 1
%%DocumentData: Clean7Bit
width height scale
width height 8 [width 0 0 -height 0 height
{currentfile 3 width mul string readhexstring pop} bind
false 3 colorimage

...hexadecimal information cut...

%%EOF
```



The modifications for greyscale images are quite simple, change the line

```
{currentfile 3 width mul string readhexstring pop} bind
```

to

```
{currentfile width string readhexstring pop} bind
```

and of course only write one hexadecimal number (representing the grey level of the pixel) for each pixel of the image. This technique should work for images of any size.

## Paper sizes

Paper Size	Dimension (in points)
Comm #10 Envelope	297 x 684
C5 Envelope	461 x 648
DL Envelope	312 x 624
Folio	595 x 935
Executive	522 x 756
Letter	612 x 792
Legal	612 x 1008
Ledger	1224 x 792
Tabloid	792 x 1224
A0	2384 x 3370
A1	1684 x 2384
A2	1191 x 1684
A3	842 x 1191
A4	595 x 842
A5	420 x 595
A6	297 x 420
A7	210 x 297
A8	148 x 210
A9	105 x 148
B0	2920 x 4127
B1	2064 x 2920
B2	1460 x 2064
B3	1032 x 1460
B4	729 x 1032
B5	516 x 729
B6	363 x 516
B7	258 x 363
B8	181 x 258
B9	127 x 181
B10	91 x 127

# QD3D

## Apple QuickTime 3D Meta file format

Converted by Paul Bourke

---

### Metafile Header

**Full name:** 3DMF, 3DMetafile

**Drawable:** No

**Parent Class Heirarchy:** 3DMF

**Binary type:** 3DMF

**Ascii type:** 3DMetafile

**Binary size:** 16

**Parent Objects:** none

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

#### Description:

The metafile header is the first object to appear in any metafile.

Metafile versions of 1.x are expected to maintain some degree of compatibility.

Flags indicate to a general degree of how the file is structured or should be read.

A database file indicates that the metafile is a library, and all objects that are shared appear in the table of contents.

A stream file indicates that no references exist in the metafile, so that a parsing program may discard encountered data when it is through with it.

If the toc location (Table of Contents location) is NULL, the entire file must be parsed to find a Table Of Contents.

#### Data structure:

Uns16 majorVersion

Uns16 minorVersion

MetafileFlags flags

FilePointer tocLocation

As of this release:

majorVersion = 0

minorVersion = 8

The final release of the metafile will be:

```
majorVersion = 1
minorVersion = 0
```

MetafileFlags bitfield is:

```
Binary Text
0x00000000 Normal
0x00000001 Stream
0x00000002 Database
```

### **Text samples:**

```
3DMetafile (
  1 0 # version
  Normal
  toc>
)
...
toc: TableOfContents (
  ...
)
```

---

## **Begin Group**

**Full name:** 3DMF, BeginGroup

**Drawable:** No

**Parent Class Heirarchy:** 3DMF

**Binary type:** bgng

**Ascii type:** BeginGroup

**Binary size:** sizes of contained objects + (8 \* number of child objects)

**Parent Objects:** special

**Format:** Data Format

**Subobjects:**

**Inherited:** No

**Referencable:** No

### **Description:**

The begin group object is used similarly to the container object, except it is used as the starting delimiter for a group. This allows a naive parser to traverse a metafile without special casing the many types of groups that appear in the metafile spec. It also allows for a single mechanism that is used to declare a group.

Please note that all objects of type group **MUST** be contained in a begin group, to allow them to be identified as starting a group.

**Data structure:****Text samples:**

```
BeginGroup ( DisplayGroup ( ) )  
  Triangle ( 0 0 1 0 0 0 0 1 0 )  
  Translate ( 1 2 3 )  
  Sphere ( )  
EndGroup ( )
```

```
BeginGroup (  
  OrderedDisplayGroup ( )  
  DisplayGroupState ( DoNotDraw )  
)  
  Triangle ( 0 0 1 0 0 0 0 1 0 )  
  Translate ( 1 2 3 )  
  Sphere ( )  
EndGroup ( )
```

```
BeginGroup ( InfoGroup ( ) )  
  CString ( Copyright (c) 1995 )  
EndGroup ( )
```

---

## Container

**Full name:** 3DMF, Container

**Drawable:** No

**Parent Class Heirarchy:** 3DMF

**Binary type:** cntr

**Ascii type:** Container

**Binary size:** sizes of contained objects + (8 \* number of child objects)

**Parent Objects:** special

**Format:** Data Format

**Subobjects:** special

**Inherited:** No

**Referencable:** No

**Description:**

Used to bind objects together to form a single object.

Container objects always contain other objects.

The first object in the container is called the root object, and sets the scope of the remaining objects in the container, called subobjects.

In general, the root object instantiates the object with its default values, and subobjects append information to the original root object.

There is one exception to these encapsulation rules, which is group objects. Although group objects contain a list of other objects, they are delimited with another 3DMF object, the end group object.

**Data structure:**

## Text samples:

```
Container (
  Box ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 1 )
  )
)
```

---

## End Group

**Full name:** 3DMF, EndGroup

**Drawable:** No

**Parent Class Heirarchy:** 3DMF

**Binary type:** endg

**Ascii type:** EndGroup

**Binary size:** 0

**Parent Objects:** none

**Format:** No Data

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

This object is used as a delimiter for all group objects.

### Data structure:

Groups should be arranged into non-overlapping pairs of BeginGroup ( group type/data ) and an EndGroup object.

All groups must be arranged into DAGs. (no cycles are permitted)

## Text samples:

```
# Empty group
BeginGroup ( OrderedDisplayGroup ( ) )
EndGroup ( )
```

```
# Group containing 1 object
BeginGroup ( DisplayGroup ( ) )
  Translate ( 1 2 3 )
  Sphere ( )
EndGroup ( )
```

```
# Inline group referenced elsewhere
```

```
REDColor:
BeginGroup (
  DisplayGroup ( )
```

```

    DisplayGroupState ( IsInline )
)
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 0 )
  )
EndGroup ( )

BeginGroup ( DisplayGroup ( ) )
  Reference ( 1 ) # REDColor
  Cone ( ) # Cone is RED
EndGroup ( )
toc: TableOfContents (
  nextTOC> -1 2 0 12
  1
  1 REDColor>
)

```

---

## Junk

**Full name:** 3DMF, Junk

**Drawable:** No

**Parent Class Heirarchy:** 3DMF

**Binary type:** junk

**Ascii type:** Junk

**Binary size:** any size

**Parent Objects:** special

**Format:** Data Format

**Subobjects:** any

**Inherited:** No

**Referencable:** No

### Description:

The junk object contains garbage, and serves as a placeholder for deleted information in the metafile. Junk objects should always be skipped and never parsed.

### Data structure:

### Text samples:

```

Container (
  Box ( )
  Junk (
    AttributeSet ( )
    DiffuseColor ( 1 0 1 )
  )
)

```

is equivalent to:

```
Box ( )
```

---

## Reference

**Full name:** 3DMF, Reference

**Drawable:** No

**Parent Class Heirarchy:** 3DMF

**Binary type:** rfm

**Ascii type:** Reference

**Binary size:** 4

**Parent Objects:** may be substituted for any Shared object

**Format:** Data Format

**Subobjects:** 1 Storage object (optional)

**Inherited:** No

**Referencable:** No

### Description:

The reference object is used to instantiate an object multiple times in a metafile.

It may be substituted anywhere in the metafile for another Shared object. Only shared objects may be referenced.

References are resolved in the Table Of Contents. If a Storage object is specified as a subobject, it is assumed that the reference is external to the current metafile, and should be resolved in that external storages table of contents.

### Data structure:

Uns32 refID

if refID = 0, must contain subobjects

if refID > 0, a TOC must exist in current metafile that contains refIDs resolution

This refID is resolved in the current metafile unless a Storage subobject is found in the Reference

### Text samples:

```
Reference ( 23 ) # internal reference
```

```
...
toc: TableOfContents (
  nextTOC> 35 -1 0 12
  ...
  20 CarFrame>
  21 Axle>
  23 WheelOfCar>
  ...
)
```

```
Container ( # external reference
  Reference ( 23 )
```



```
UnixPath ( parts/car.eb )  
)
```

---

## Table Of Contents

**Full name:** 3DMF, TableOfContents

**Drawable:** No

**Parent Class Heirarchy:** 3DMF

**Binary type:** toc

**Ascii type:** TableOfContents

**Binary size:** 28 + (tocEntrySize \* nEntries)

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The table of contents provides a means of resolving references within a file. The nextTOC file pointer points to the next table of contents in the file, or is NULL if no other table of contents exists.

The reference seed indicates the next available reference id available for reference objects. It is an unsigned positive number that is incremented with each additional reference in a file. It is always one more than the maximum reference seed in a file.

The type seed indicates the next available type ID available for type objects. It is a negative number that is decremented with each additional type in a file. It is always one less than the minimum type seed in a file.

The tocEntryType and tocEntrySize are a set of paired values which indicate the size and type of information stored in a tocEntry.

The tocEntries are sorted by reference ID, in increasing order, to allow fast searching of the table of contents.

### Data structure:

FilePointer nextTOC

Uns32 refSeed

Int32 typeSeed

Uns32 tocEntryType

Uns32 tocEntrySize

Uns32 nEntries

TOCEntry tocEntries

refSeed > 0

typeSeed < 0

tocEntryType = 0 or 1  
tocEntrySize = 12 or 16, based upon tocEntryType  
the TOCEntry structure is:

- tocEntryType 0, tocEntrySize 12 is:

Uns32 refID  
FilePointer objLocation

- tocEntryType 1, tocEntrySize 16 is:

Uns32 refID  
FilePointer objLocation  
ObjectType objType

### **Text samples:**

```
3DMetafile (
  1 0
  Normal
  toc>
)
box23:
Mesh (
  45 # nVertices
  ...
)
Reference ( 1 )
Arrows:
BeginGroup ( DisplayGroup ( ) )
  Cone ( )
  Scale ( 0.2 0.1 0.2 )
  Cylinder ( )
EndGroup ( )
Reference ( 2 )
Reference ( 4 )
...
Type ( -1 Joes Garage:RepairHistory )
...

-1 ( Jim Fixed lug nut 0.23 0.2 1.2 )

toc:
TableOfContents (
  nextTOC>
  5 # refSeed
  -2 # typeSeed
  0 12 # tocEntry Type/Size
  3 # nEntries
  1 box23>
  2 Arrows>
  4 Geom34>
)
```

---

# Type

**Full name:** 3DMF, Type

**Drawable:** No

**Parent Class Heirarchy:** 3DMF

**Binary type:** type

**Ascii type:** Type

**Binary size:** 4 + sizeof(String)

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

## Description:

A type definition is used to declare a custom data type. A type definition may appear anywhere in a file, however, the custom type must be encountered before the custom object of that type is encountered..

All custom types in the metafile are negative numbers, and the typeID field begins at -1 and is decremented for each additional type. Only 2147483648 (or  $2^{31}$ ) custom types are permitted in a single metafile.

The owner string is an ISO 9070 registered owner string. Owner strings are unique globally for each type of custom data.

In the binary and text metafile, the typeID is used as the object type later in the file.

## Data structure:

Int32 typeID

String owner

typeID < 0

owner string

## Text samples:

```
Type (  
  -1  
  Joes Garage:BoltData  
)  
  
...  
  
-1 (  
  -2.3 34 # Stress (kPA/area)  
)
```

---

# Face Attribute Set List

**Full name:** Data, AttributeSetList, FaceAttributeSetList

**Drawable:** No

**Parent Class Heirarchy:** Data, AttributeSetList

**Binary type:** fasl

**Ascii type:** FaceAttributeSetList

**Binary size:**  $12 + nIndices * sizeof(Uns) + padding$

**Parent Objects:** ALWAYS: Box, GeneralPolygon, Mesh, TriGrid

**Format:** Data Format

**Subobjects:** many AttributeSet (order-dependent)

**Inherited:** No

**Referencable:** No

## Description:

The face attribute set list specifies a list of attributes to be attached to a set of faces determined by the parents topology.

nObjects indicates the total number of objects being mapped to.

packing indicates how AttributeSet objects are mapped to indices. Include packing lists the face indices, in sequential order, of those faces to be assigned face attribute sets. Exclude packing lists the face indices, in sequential order, of those faces to NOT be assigned face attribute sets.

So, for example, supposing nObjects was 5, Include packing with a list of 3 indices after it means that there are 3 subobjects, each assigned to the indices in their order. Exclude packing with a list of 3 indices after it means there are 2 attribute sets subobjects, assigned to the indices NOT in the exclude list, in order.

The face attribute set list is padded to the nearest long word.

The values in indices always appear in increasing order.

If a packing value other than Include or Exclude is found, this object and its subobjects should be ignored.

## Data structure:

Uns32 nObjects

PackingEnum packing

Uns32 nIndices

Uns32 indices[nIndices]

nObjects must match parent values

PackingEnum is:

Binary Text

0x00000000 Include  
0x00000001 Exclude

0 indices < nObjects

### Text samples:

```
Container (  
  Box ( )  
  Container (  
    FaceAttributeSetList (  
      6 Include 2  
      0 1  
    )  
    Container ( # assigned to 0  
      AttributeSet ( )  
      DiffuseColor ( 1 0 0 )  
    )  
    Container ( # assigned to 1  
      AttributeSet ( )  
      DiffuseColor ( 0 0 1 )  
    )  
  )  
)
```

```
Container (  
  Box ( )  
  Container (  
    FaceAttributeSetList (  
      6 Exclude 2  
      2 4  
    )  
    Container ( # assigned to 0  
      AttributeSet ( )  
      DiffuseColor ( 1 0 0 )  
    )  
    Container ( # assigned to 1  
      AttributeSet ( )  
      DiffuseColor ( 1 1 0 )  
    )  
    Container ( # assigned to 3  
      AttributeSet ( )  
      DiffuseColor ( 1 0 1 )  
    )  
    Container ( # assigned to 5  
      AttributeSet ( )  
      DiffuseColor ( 0 0 1 )  
    )  
  )  
)
```

---

## Geometry Attribute Set List

**Full name:** Data, AttributeSetList, GeometryAttributeSetList

**Drawable:** No

**Parent Class Heirarchy:** Data, AttributeSetList

**Binary type:** gasl

**Ascii type:** GeometryAttributeSetList

**Binary size:** 12 + nIndices \* 4 + padding

**Parent Objects:** ALWAYS: PolyLine

**Format:** Data Format

**Subobjects:** many AttributeSet (order-dependent)

**Inherited:** No

**Referencable:** No

**Description:**

The geometry attribute set list specifies a list of attributes to be attached to a set of geometric entities determined by the parents topology.

Currently, only the PolyLine primitive uses this object. Each attribute set is mapped to a line segment in the PolyLine.

Packing for this object is identical to the other attribute set lists.

**Data structure:**

Uns32 nObjects

PackingEnum packing

Uns32 nIndices

Uns32 indices[nIndices]

nObjects must match parent values

PackingEnum described in FaceAttributeSetList

**Text samples:**

```
Container (
  PolyLine (
    3
    10 2 3
    0 0 0
    2 8.5 3
  )
  Container (
    GeometryAttributeSetList (
      3 Exclude 1 1
    )
    Container ( # segment 0
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container ( # segment 2
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

---

## Vertex Attribute Set List

**Full name:** Data, AttributeSetList, VertexAttributeSetList

**Drawable:** No

**Parent Class Heirarchy:** Data, AttributeSetList

**Binary type:** vasl

**Ascii type:** VertexAttributeSetList

**Binary size:** 12 + nIndices \* sizeof(Uns) + padding

**Parent Objects:** ALWAYS: GeneralPolygon, Line, Mesh, Polygon, PolyLine, Triangle, TriGrid

**Format:** Data Format

**Subobjects:** many AttributeSet (order-dependent)br> **Inherited:** No

**Referencable:** No

### Description:

The vertex attribute set list specifies a list of attributes to be attached to a set of vertices determined by the parents topology.

Packing for this object is identical to the other attribute set lists.

### Data structure:

Uns32 nObjects

PackingEnum packing

Uns32 nIndices

Uns32 indices[nIndices]

nObjects must match parent values

PackingEnum described in FaceAttributeSetList

### Text samples:

```
Container (
  Triangle (
    0 0 0
    0 2 0
    0 0 2
  )
  Container (
    VertexAttributeSetList (
      3 Exclude 0
    )
    Container ( # vertex 0
      AttributeSet ( )
      DiffuseColor ( 0 0 0 )
    )
    Container ( # vertex 0
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

```
    Container ( # vertex 0
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
    )
  )
)
```

---

## Camera Placement

**Full name:** Data, CameraData, CameraPlacement

**Drawable:** No

**Parent Class Heirarchy:** Data, CameraData

**Binary type:** cimpl

**Ascii type:** CameraPlacement

**Binary size:** 36

**Parent Objects:** ALWAYS: Camera objects: ViewAngleAspectCamera, ViewPlaneCamera, OrthographicCamera

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The camera placement specifies the location and orientation of the camera in space, by a camera location, a point of interest, and an up vector. This placement locates and orients the camera, and defines a space in which the rest of the parameters are interpreted.

If the up vector is not of unit length upon reading, it should be normalized by the reading program.

The camera placement is affected by the current transformation state in a hierarchy. The location and point of interest are multiplied by the current transformation directly, and the up vector is multiplied by the current transformation minus any translation component of the transform, and unitized.

The camera vector is defined as:

camera vector = (pointOfInterest - location)

### Data structure:

Point3D location

Point3D pointOfInterest

Vector3D upVector

$\text{upVector} \wedge (\text{pointOfInterest} - \text{location})$

$|\text{upVector}| = 1.0$

Default Values:

0 0 1 # location



```
0 0 0 # pointOfInterest
0 1 0 # upVector
```

### Text samples:

```
Container (
  OrthographicCamera (
    -1 -1 1 1
  )
  CameraPlacement (
    10 0 0 # located along X axis
    0 0 0 # point of interest is origin
    0 1 0 # Y is up
  )
)
```

---

## Camera Range

**Full name:** Data, CameraData, CameraRange

**Drawable:** No

**Parent Class Heirarchy:** Data, CameraData

**Binary type:** cmrg

**Ascii type:** CameraRange

**Binary size:** 8

**Parent Objects:** ALWAYS: Camera objects: ViewAngleAspectCamera, ViewPlaneCamera, OrthographicCamera

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The camera range affects the clipping of the viewing frustum.

This is used to bound the range of the set of objects of interest.

Hither is the frontmost clipping plane (sometimes referred to as near), yon is the backmost clipping plane (sometimes referred to as far).

Each of these distances is measured along the camera vector, described in the Camera Placement object.

### Data structure:

Float32 hither

Float32 yon

$0 < \text{hither} < \text{yon}$

default is:

hither e  
yon

**Text samples:**

```
Container (  
  OrthographicCamera (  
    -1 -1 1 1  
  )  
  CameraRange (  
    0.1 2 # hither, yon  
  )  
)
```

## Camera ViewPort

**Full name:** Data, CameraData, CameraViewPort

**Drawable:** No

**Parent Class Heirarchy:** Data, CameraData

**Binary type:** cmvp

**Ascii type:** CameraViewPort

**Binary size:** 16

**Parent Objects:** ALWAYS: any Camera object: ViewAngleAspectCamera, ViewPlaneCamera, OrthographicCamera

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

The camera viewport specifies a rectangular region of the viewing frustum to which the image is clipped. Effectively the view port may be used to zoom in on a particular feature of an image.

The view port uses the cartesian coordinate system, with Y towards the top of the screen, X to the right, and Z coming towards the viewer, as shown in the diagram.

**Data structure:**

Point2D origin

Float32 width

Float32 height

-1 origin.x 1

-1 origin.y 1

0 < width 2

0 < height 2

Default is:  
-1 1 # origin  
2 # width  
2 # height

### Text samples:

```
Container (  
  OrthographicCamera (  
    -1 -1 1 1  
  )  
  CameraViewPort ( # zoom to 200%  
    -0.5 0.5 1 1  
  )  
)
```

---

## Bottom Cap Attribute Set

**Full name:** Data, CapData, BottomCapAttributeSet

**Drawable:** No

**Parent Class Heirarchy:** Data, CapData

**Binary type:** bcas

**Ascii type:** BottomCapAttributeSet

**Binary size:** 0

**Parent Objects:** ALWAYS: Cone, Cylinder

**Format:** No Data

**Subobjects:** 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** No

### Description:

This object simply allows the attributes associated with the bottom cap of a Cone or Cylinder to be encapsulated.

Presence of a bottom cap attribute set does not necessarily mean the bottom cap is drawn.

The Caps object determines whether the Cone and Cylinder caps are drawn or not.

### Data structure:

### Text samples:

```
3DMetafile ( 1 0 Normal toc> )  
Container (  
  Cone ( )  
  Caps ( Bottom )  
  Container (  
    BottomCapAttributeSet ( )  
  )  
)
```

```

    capColor: Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
  )
)
Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
    Reference (1)
  )
)
...
toc: TableOfContents (
  ...
  1 capColor>
)

```

---

## Caps

**Full name:** Data, CapData, Caps

**Drawable:** No

**Parent Class Heirarchy:** Data, CapData

**Binary type:** caps

**Ascii type:** Caps

**Binary size:** 4

**Parent Objects:** ALWAYS: Cone, Cylinder

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

In the binary file, the upper 28 bits of the caps bitfield should be ignored. In the text file, unknown bitfield strings should be skipped. The default caps value is 0, or None.

The Top cap bit (label) is ignored in the Cone.

### Data structure:

CapsFlags caps

CapsFlags is defined as:

Binary Text

0x00000000 None

0x00000001 Bottom

0x00000002 Top

Default is:  
None

### Text samples:

```
Container (
  Cylinder ( )
  Caps ( Bottom | Top )
)

Container ( # Cone with a blue bottom
  Cone ( )
  Caps ( Bottom )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)
```

---

## Face Cap Attribute Set

**Full name:** Data, CapData, FaceCapAttributeSet

**Drawable:** No

**Parent Class Heirarchy:** Data, CapData

**Binary type:** fcas

**Ascii type:** FaceCapAttributeSet

**Binary size:** 0

**Parent Objects:** ALWAYS: Cone, Cylinder

**Format:** No Data

**Subobjects:** 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** No

### Description:

Attaches a set of attributes to the face cap of the cone and cylinder primitives. For the cone, its indicated in the diagram.

### Data structure:

### Text samples:

```
Container (
  Cone ( )
  Caps ( Bottom )
  Container (
    FaceCapAttributeSet ( )
    Container (
```

```
    AttributeSet ( )
    DiffuseColor ( 0.2 0.9 0.4 )
  )
)
```

---

## Top Cap Attribute Set

**Full name:** Data, CapData, TopCapAttributeSet

**Drawable:** No

**Parent Class Heirarchy:** Data, CapData

**Binary type:** tcas

**Ascii type:** TopCapAttributeSet

**Binary size:** 0

**Parent Objects:** ALWAYS: Cylinder

**Format:** No Data

**Subobjects:** 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** No

### Description:

Attaches a set of attributes to the top cap of the cylinder primitive.

Presence of a top cap attribute set does not necessarily mean the top cap is drawn.

The Caps object determines whether the Cylinder caps are drawn or not.

### Data structure:

### Text samples:

```
Container (
  Cylinder ( )
  Caps ( Top )
  Container (
    TopCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0.2 0.9 0.4 )
    )
  )
)
```

---

## Display Group State

**Full name:** Data, DisplayGroupState

**Drawable:** No

**Parent Class Heirarchy:** Data

**Binary type:** dgst

**Ascii type:** DisplayGroupState

**Binary size:** 4

**Parent Objects:** ALWAYS: DisplayGroup, OrderedDisplayGroup

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

This piece of data is a subobject only to objects of type display group. It affects how a display group is traversed. These flags allow any display group to have the following characteristics:

To have invisible objects in a scene which may act as user interface items, or may aid in bounding complex geometries

To have non-user interface items which may serve only as decoration and should not be picked.

To have a group of shaders/attributes which affects the state as an inline group so it may be instantiated and inherited in many parts of a hierarchy.

**Data structure:**

DisplayGroupStateFlags traversalFlags

DisplayGroupStateFlags is:

Binary Text

0x00000000 None

0x00000001 Inline

0x00000002 DoNotDraw

0x00000004 NoBoundingBox

0x00000008 NoBoundingSphere

0x00000010 DoNotPick

default is:

Binary Text

0x00000000 None

**Text samples:**

# to pick a chess piece by a box around it

```
BeginGroup ( DisplayGroup ( ) )
  PickIDStyle ( 1 )
  BeginGroup (
    DisplayGroup ( )
    DisplayGroupState ( DoNotDraw )
  )
  Scale ( 2 4 2 )
```

```
    Box ( )
EndGroup ( )

Container (
    DisplayGroup ( )
    DisplayGroupState ( DoNotPick )
)
Mesh ( # chess piece
    56 # nVertices
    0.2 0.3 0.5
    ...
)
EndGroup ( )
EndGroup ( )
```

---

## General Polygon Hint

**Full name:** Data, GeneralPolygonHint

**Drawable:** No

**Parent Class Heirarchy:** Data

**Binary type:** gplh

**Ascii type:** GeneralPolygonHint

**Binary size:** 4

**Parent Objects:** ALWAYS: GeneralPolygon

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The GeneralPolygonHint gives a reading application some hint of what shape a general polygon is.

A Complex general polygon may contain intersecting, concave, or convex polygons.

A Concave general polygon contains no intersecting polygons, but contains 1 or more concave polygons.

A Convex general polygon indicates that all contained polygons are convex and non-intersecting.

### Data structure:

GeneralPolygonHintEnum shapeHint

GeneralPolygonHintEnum is:

Binary Text

0x00000000 Complex

0x00000001 Concave

0x00000002 Convex



default is:  
Complex

### Text samples:

```
Container (  
  GeneralPolygon (  
    1  
    3  
    0 2 3  
    0 2 1  
    2 0 0  
  )  
  GeneralPolygonHint ( Convex )  
)
```

---

## Light Data

**Full name:** Data, LightData

**Drawable:** No

**Parent Class Heirarchy:** Data

**Binary type:** lght

**Ascii type:** LightData

**Binary size:** 20

**Parent Objects:** ALWAYS: any Light: SpotLight, AmbientLight, PointLight, DirectionalLight

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The light data object affects information about a light that is common among all lights.

A light may be on or off, may vary in intensity, or may have different colors.

### Data structure:

Boolean isOn

Float32 intensity

ColorRGB color

0 intensity 1

Default is:

True # isOn

1.0 # intensity

1 1 1 # color

## Text samples:

```
Container (  
  AmbientLight ( )  
  LightData (  
    True  
    0.4  
    1 0 0  
  )  
)
```

---

## Mesh Corners

**Full name:** Data, MeshCorners

**Drawable:** No

**Parent Class Hierarchy:** Data

**Binary type:** crnr

**Ascii type:** MeshCorners

**Binary size:**  $4 + \text{sizeof}(\text{corners}[0..\text{nCorners}-1]) \text{sizeof}(\text{MeshCorner}) = 8 + \text{nFaces} * 4$

**Parent Objects:** ALWAYS: Mesh

**Format:** Data Format

**Subobjects:** nCorners AttributeSets (order-dependent)

**Inherited:** No

**Referencable:** No

### Description:

Mesh Corners allow you to attach AttributeSets to a mesh vertex, to allow for attributes to be associated with a particular face-vertex pair. This may be used to allow sharp corners in an object (diagram above), to set different shading parameters for adjacent faces, etc.

Mesh corners supplies a vertex index, a list of face indices, and a vertex attribute set for each corner.

The mesh corners object most often appears inside a container, and always has AttributeSet subobjects. The first corner in the mesh corners data is mapped to the first attribute set subobject, the second corner to the second attribute set, etc.

### Data structure:

Uns32 nCorners

MeshCorner corners[nCorners]

$0 < \text{nCorners}$

where MeshCorner is:

Uns32 vertexIndex

Uns32 nFaces

Uns32 faces[nFaces]

$0 < nFaces$

### Text samples:

```
Container (
  Mesh (
    ...
  )
  Container (
    MeshCorners (
      2 # numCorners

      # Corner 0
      5 # vertexIndex
      2 # faces
      25 26 # face indices

      # Corner 1
      5 # vertexIndex
      2 # faces
      23 24 # face indices
    )
    Container (
      AttributeSet ( )
      Normal ( -0.2 0.8 0.3 )
    )
    Container (
      AttributeSet ( )
      Normal ( -0.7 -0.1 0.4 )
    )
  )
)
```

---

## Mesh Edges

**Full name:** Data, MeshEdges

**Drawable:** No

**Parent Class Heirarchy:** Data

**Binary type:** edge

**Ascii type:** MeshEdges

**Binary size:**  $4 + \text{sizeof}(\text{corners}[0..n\text{Corners}-1]) \text{sizeof}(\text{MeshEdges}) = 2 * \text{sizeof}(\text{Uns})$

**Parent Objects:** ALWAYS: Mesh

**Format:** Data Format

**Subobjects:** nCorners AttributeSets (order-dependent)

**Inherited:** No

**Referencable:** No

### Description:

Mesh Edges allow you to attach AttributeSets to a mesh edge.

You may attach mesh edges to any edge in the mesh that corresponds to a face edge. To specify and

edge that should have an attribute set attached to it, include it as the nth edge the list of edges, and specify the attribute set as the nth attribute set subobject.

**Data structure:**

Uns32 nEdges  
MeshEdge edges[nEdges]

0 < nEdges  
where MeshEdge is:

Uns32 vertexIndex1  
Uns32 vertexIndex2

**Text samples:**

```
Container (
  Mesh (
    ...
  )
  Container (
    MeshEdges (
      2 # numEdges
      0 1 # 1st edge vertexIndices
      1 2 # 2nd edge vertexIndices
    )
    Container ( /* 1st edge attribute set */
      AttributeSet ( )
      DiffuseColor ( 0.2 0.8 0.3 )
    )
    Container ( /* 2nd edge attribute set */
      AttributeSet ( )
      DiffuseColor ( 0.8 0.2 0.3 )
    )
  )
)
```

## NURB Curve 2D

**Full name:** Data, NURBCurve2D

**Drawable:** No

**Parent Class Heirarchy:** Data

**Binary type:** nb2c

**Ascii type:** NURBCurve2D

**Binary size:** 8 + 12 \* nPoints + 4 \* (order + nPoints)

**Parent Objects:** ALWAYS: TrimCurves

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

The NURB Curve 2D is a subobject of the TrimCurves object, and supplies a 2 dimensional curve to trim NURB Patches.

**Data structure:**

Uns32 order

Uns32 nPoints

RationalPoint3D points[nPoints]

Float32 knots[order + nPoints]

2 order

2 nPoints

0 < points[...].w (weights of points)

**Text samples:**

## Shader Data

**Full name:** Data, ShaderData

**Drawable:** No

**Parent Class Heirarchy:** Data

**Binary type:** shdr

**Ascii type:** ShaderData

**Binary size:** 8

**Parent Objects:** ALWAYS: any Shader

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

The shader data initializes boundary wrapping conditions for a shader.

**Data structure:**

ShaderUVBoundaryEnum uBounds

ShaderUVBoundaryEnum vBounds

ShaderUVBoundaryEnum is:

Binary Text

0x00000000 Wrap

0x00000001 Clamp

default is:

Wrap Wrap

## Text samples:

```
Container (
  CustomShader ( ... )
  ShaderData ( Wrap Clamp )
)
```

---

# Shader Transform

**Full name:** Data, ShaderTransform

**Drawable:** No

**Parent Class Heirarchy:** Data

**Binary type:** sdxif

**Ascii type:** ShaderTransform

**Binary size:** 64

**Parent Objects:** ALWAYS: any Shader

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

## Description:

This transforms a shaded object into another world space coordinate system. It does not affect how the object is drawn, or the current state of the hierarchy.

## Data structure:

Matrix4x4 shaderTransform

## Text samples:

```
Container (
  3DMarbleShader ( )
  ShaderTransform (
    1 0 0 0
    0 1 0 0
    0 0 1 0
    2 3 4 1
  )
)
...
Type ( -3 Apple:ATG:3DMarbleShader )
Container (
  -3 ( 2.3 1.0 -10 )
  ShaderTransform (
    1 0 0 0
    0 1 0 0
    0 0 1 0
    2 3 4 1
  )
)
```

---

## Shader UV Transform

**Full name:** Data, ShaderUVTransform

**Drawable:** No

**Parent Class Heirarchy:** Data

**Binary type:** sduv

**Ascii type:** ShaderUVTransform

**Binary size:** 36

**Parent Objects:** ALWAYS: any Shader

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The Shader UV transform allows the uvs on a geometric object to be transformed before shading occurs.

This allows you to rotate a texture map, for example.

### Data structure:

Matrix3x3 matrix

### Text samples:

```
Container (  
  TextureShader ( )  
  ShaderUVTransform (  
    1 0 0  
    0 1 0  
    0.2 0.3 1  
  )  
  PixmapTexture (  
    ...  
  )  
)
```

---

## Trim Curves

**Full name:** Data, TrimLoop

**Drawable:** No

**Parent Class Heirarchy:** Data

**Binary type:** trml

**Ascii type:** TrimLoop

**Binary size:** 0

**Parent Objects:** ALWAYS: NURBPatch  
**Format:** No Data  
**Subobjects:** many NURBCurve2D (order-dependent)  
**Inherited:** No  
**Referencable:** No

**Description:**

The Trim Loop subobject allows users to attach trimming loops to a NURB Patch. The Trim Loop object contains no data, and serves only as an encapsulation of various 2-dimensional curves used for trimming.

The Trim loop object contains a sequence of 2 dimensional curves which are concatenated together to form a loop. The subobjects are order-dependent. Each trim loop subobject should contain loops that are geometrically continuous, meaning the first trim curves end point ends at the next trim curves starting point.

In the metafile version 1.0, the only 2-dimensional curve allowed is a NURBCurve2D.

In future releases of the metafile, we expect to add additional types of 2d trim curves for trimming NURBS.

**Data structure:**

**Text samples:**

```
Container (  
  NURBPatch (  
    4 4 4 4 # u,v order, num M,N points  
    -2 2 0 1   -1 2 0 1   1 2 0 1   2 2 0 1  
    -2 2 0 1   -1 2 0 1   1 0 5 1   2 2 0 1  
    -2 -2 0 1  -1 -2 0 1   1 -2 0 1   2 -2 0 1  
    -2 -2 0 1  -1 -2 0 1   1 -2 0 1   2 -2 0 1  
    0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 # knots  
  )  
  Container (  
    TrimLoop (  
      NURBCurve2D (  
        ...  
      )  
      NURBCurve2D (  
        ...  
      )  
    )  
  )  
)
```

---

## Image Clear Color

**Full name:** Data, ViewHintsData, ImageClearColor  
**Drawable:** No



**Parent Class Heirarchy:** Data, ViewHintsData

**Binary type:** imcc

**Ascii type:** ImageClearColor

**Binary size:** 12

**Parent Objects:** ALWAYS: ViewHints

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

This specifies the preferred rgb color with should be used to clear the drawing areas background.

**Data structure:**

ColorRGB clearColor

**Text samples:**

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  ImageClearColor ( 1 1 1 ) # white
)
Box ( )
```

## Image Dimensions

**Full name:** Data, ViewHintsData, ImageDimensions

**Drawable:** No

**Parent Class Heirarchy:** Data, ViewHintsData

**Binary type:** imdm

**Ascii type:** ImageDimensions

**Binary size:** 8

**Parent Objects:** ALWAYS: ViewHints

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

The image dimensions specifies the preferred image width and height in bits. It is a subobject of the view hints, which aids an application in determining how to display an image.

**Data structure:**

Uns32 width

Uns32 height

0 < width  
0 < height

### Text samples:

```
3DMetafile ( 1 0 Normal toc> )  
Container (   
  ViewHints ( )   
  ImageDimensions ( 32 32 )   
  ImageClearColor ( 1 1 1 )   
 )   
Rotate ( X 0.75 )   
Rotate ( Y 0.75 )   
Container (   
  AttributeSet ( )   
  DiffuseColor ( 1 0 0 )   
 )   
Box ( )
```

---

## Image Mask

**Full name:** Data, ViewHintsData, ImageMask

**Drawable:** No

**Parent Class Heirarchy:** Data, ViewHintsData

**Binary type:** immk

**Ascii type:** ImageMask

**Binary size:** 12 + (rowBytes \* height) + padding

**Parent Objects:** ALWAYS: ViewHints

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The image mask is a bitmap that specifies how an images rendered pixels should be clipped. The origin of the bitmap (the upper-left) is aligned with the origin (upper left) of the drawing area. Generally, the image mask and the image dimensions are used simultaneously to specify an image which is partially clipped.

The example to the right specifies a mask to clip a 32x32 image. The application using this data uses this clip mask to only render to a clipped portion of a custom document icon in this case, the bitmap will only draw inside of a document icon, providing a small preview in the Finder with a black document icon. The image mask to the right was used to render the example above.

### Data structure:

Uns32 width

Uns32 height

Uns32 rowBytes



**Parent Objects:** ALWAYS: AttributeSet

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

The ambient coefficient describes the intensity of the ambient light that is reflected by a surface.

**Data structure:**

Float32 ambientCoefficient

0 ambientCoefficient 1.0

**Text samples:**

```
Container (  
  AttributeSet ( )  
  AmbientCoefficient ( 0.7 )  
)
```

---

## Diffuse Color

**Full name:** Element, Attribute, DiffuseColor

**Drawable:** No

**Parent Class Heirarchy:** Element, Attribute

**Binary type:** kdif

**Ascii type:** DiffuseColor

**Binary size:** 12

**Parent Objects:** ALWAYS: AttributeSet

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

The diffuse color indicates the amount of diffuse light reflected by a surface.

**Data structure:**

ColorRGB diffuseColor

**Text samples:**

```
Container (  
  AttributeSet ( )  
  DiffuseColor ( 1 0 0 ) # red  
)
```

---

## Highlight State

**Full name:** Element, Attribute, HighlightState

**Drawable:** No

**Parent Class Heirarchy:** Element, Attribute

**Binary type:** hlst

**Ascii type:** HighlightState

**Binary size:** 4

**Parent Objects:** ALWAYS: AttributeSet

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The highlight state attribute, when true, indicates that the current attribute state is overridden with the current highlight styles attribute set. The highlight state attribute allows various portions of a geometry object to be highlighted for user interface, etc. while retaining the integrity of a geometrys attribute set.

### Data structure:

Boolean highlighted

### Text samples:

```
Container (
  HighlightStyle ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 0 ) # RED
  )
)
...
Container (
  Polygon (
    3
    0 1 2
    0 0 0
    0 -1 2
  )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0 1 2 )
    HighlightState ( True )
    # polygon is drawn RED
  )
)
```

---

# Normal

**Full name:** Element, Attribute, Normal  
**Drawable:** No  
**Parent Class Heirarchy:** Element, Attribute  
**Binary type:** nrml  
**Ascii type:** Normal  
**Binary size:** 12  
**Parent Objects:** ALWAYS: AttributeSet  
**Format:** Data Format  
**Subobjects:** none  
**Inherited:** No  
**Referencable:** No

## Description:

If normal is not of unit length upon reading, it should be normalized. (npi)

The normal indicates the surface normal at a vertex.

## Data structure:

Vector3D normal

$|\text{normal}| = 1$

## Text samples:

```
Container (
  Polygon (
    5
    0.23423 0.56434 0.2312
    ...
  )
  Container (
    VertexAttributeSetList ( 5 Exclude 0 )
    Container (
      AttributeSet ( )
      Normal ( 0.8 -0.1 -0.1 )
    )
  )
)
```

---

# Shading UV

**Full name:** Element, Attribute, ShadingUV  
**Drawable:** No  
**Parent Class Heirarchy:** Element, Attribute  
**Binary type:** shuv  
**Ascii type:** ShadingUV

**Binary size:** 8

**Parent Objects:** ALWAYS: AttributeSet

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

The shading UV indicates an alternate UV to the Surface UV for shading purposes.

Shading UVs are generally used by shaders that affect appearance information, such as texture maps, which alter the color on a geometric surface.

Surface UVs are generally used for trimming.

**Data structure:**

Param2D shadingUV

Any UV parametrization is allowed, however, shading generally occurs with the following values.

0 shadingUV.u 1

0 shadingUV.v 1

**Text samples:**

```
Container (  
    AttributeSet ( )  
    ShadingUV ( 0 0 )  
)
```

---

## Specular Color

**Full name:** Element, Attribute, SpecularColor

**Drawable:** No

**Parent Class Heirarchy:** Element, Attribute

**Binary type:** kspc

**Ascii type:** SpecularColor

**Binary size:** 12

**Parent Objects:** ALWAYS: AttributeSet

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

The specular color indicates the color of specular highlights on a surface.

**Data structure:**

ColorRGB specularColor

**Text samples:**

```
Container (
  AttributeSet ( )
  DiffuseColor ( 0.1 0.1 0.1 ) # near-black
  SpecularColor ( 1 1 1 ) # white
)
Sphere (
  0 0 0
  0 1 0
  1 0 0
  0 0 1
)
```

---

## Specular Control

**Full name:** Element, Attribute, SpecularControl

**Drawable:** No

**Parent Class Heirarchy:** Element, Attribute

**Binary type:** cspc

**Ascii type:** SpecularControl

**Binary size:** 4

**Parent Objects:** ALWAYS: AttributeSet

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

**Description:**

The specular control attribute indicates the power to which the specular component of lighting computations is raised.

**Data structure:**

Float32 specularControl

0 specularControl

**Text samples:**

```
Container (
  AttributeSet ( )
  DiffuseColor ( 0.5 0.5 0.5 ) # near-black
  SpecularColor ( 0.5 ) # white highlights
  SpecularControl ( 1 ) # larger highlight area
)
```



Sphere ( )

---

## Surface Tangent

**Full name:** Element, Attribute, SurfaceTangent

**Drawable:** No

**Parent Class Heirarchy:** Element, Attribute

**Binary type:** srtn

**Ascii type:** SurfaceTangent

**Binary size:** 24

**Parent Objects:** ALWAYS: AttributeSet

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The surface tangent attribute indicates the direction of changing U and V on a surface.

### Data structure:

Vector3D paramU

Vector3D paramV

### Text samples:

```
Container (
  Mesh (
    ...
  )
  Container (
    VertexAttributeSetList (
      ...
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 0.1 0.293 )
      SurfaceTangent (
        1 0 0
        0 1 0
      )
    )
  )
)
```

---

## Surface UV

**Full name:** Element, Attribute, SurfaceUV

**Drawable:** No  
**Parent Class Heirarchy:** Element, Attribute  
**Binary type:** sruv  
**Ascii type:** SurfaceUV  
**Binary size:** 8  
**Parent Objects:** ALWAYS: AttributeSet  
**Format:** Data Format  
**Subobjects:** none  
**Inherited:** No  
**Referencable:** No

**Description:**

The surface UV indicates an alternate UV to the shading UV for shading purposes.

Surface UVs are generally used for trim shaders.

Shading UVs are generally used by shaders that affect appearance information, such as texture maps, which alter the color on a geometric surface.

**Data structure:**

Param2D surfaceUV

Any UV parametrization is allowed, however, shading generally occurs with the following values.

0 surfaceUV 1  
0 surfaceUV 1

**Text samples:**

```
Container (
  Mesh (
    ...
  )
  Container (
    VertexAttributeSetList (
      200 Include 4 10 21 22 11
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 0 0 )
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 0 1 )
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 1 1 )
    )
    Container (
      AttributeSet ( )
      SurfaceUV ( 1 0 )
    )
  )
)
```

)  
)

---

## Transparency Color

**Full name:** Element, Attribute, TransparencyColor

**Drawable:** No

**Parent Class Heirarchy:** Element, Attribute

**Binary type:** kxpr

**Ascii type:** TransparencyColor

**Binary size:** 12

**Parent Objects:** ALWAYS: AttributeSet

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** No

### Description:

The transparency color indicates the degree of light allowed to pass though the various channels (r,g,b) of a surface.

A color of (1, 1, 1) indicates complete transparency (meaning 100% of the light behind an object is allowed to pass through), a color of (0, 0, 0) indicates complete opacity (meaning no light passes through an object.)

### Data structure:

ColorRGB transparency

### Text samples:

```
Container (  
  Polygon (  
    ...  
  )  
  Container (  
    AttributeSet (  
      TransparencyColor ( 1 0 0 )  
    )  
  )  
)
```

---

## Generic Renderer

**Full name:** Shared, Renderer, GenericRenderrer

**Drawable:** No

**Parent Class Heirarchy:** Shared, Renderer

**Binary type:** gnrr  
**Ascii type:** GenericRenderer  
**Binary size:** 0  
**Parent Objects:** SOMETIMES: ViewHints  
**Format:** No Data  
**Subobjects:** none  
**Inherited:** No  
**Referencable:** Yes

**Description:**

A renderer that doesnt do anything, but may be used to accumulate state or for picking.

**Data structure:**

**Text samples:**

```
Container (  
  ViewHints ( )  
  GenericRenderer ( )  
  ViewAngleAspectCamera (  
    ...  
  )  
  AmbientLight ( )  
  Container (  
    AttributeSet ( )  
    DiffuseColor ( 0.2 0.2 0.2 )  
  )  
)
```

---

## Interactive Renderer

**Full name:** Shared, Renderer, InteractiveRenderer  
**Drawable:** No  
**Parent Class Heirarchy:** Shared, Renderer  
**Binary type:** ctwn  
**Ascii type:** InteractiveRenderer  
**Binary size:** 0  
**Parent Objects:** SOMETIMES: ViewHints  
**Format:** No Data  
**Subobjects:** none  
**Inherited:** No  
**Referencable:** Yes

**Description:**

The interactive renderer.

This will be renamed later when the corresponding product is named.

**Data structure:****Text samples:**

```
Container (
  ViewHints ( )
  InteractiveRenderer ( )
  ViewAngleAspectCamera (
    ...
  )
  AmbientLight ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
  )
)
```

---

## Wire Frame Renderer

**Full name:** Shared, Renderer, WireFrame

**Drawable:** No

**Parent Class Heirarchy:** Shared, Renderer

**Binary type:** wrfr

**Ascii type:** WireFrame

**Binary size:** 0

**Parent Objects:** SOMETIMES: ViewHints

**Format:** No Data

**Subobjects:** none

**Inherited:** No

**Referencable:** Yes

**Description:**

A wireframe renderer.

**Data structure:****Text samples:**

```
Container (
  ViewHints ( )
  Wireframe ( )
  ViewAngleAspectCamera (
    ...
  )
  AmbientLight ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.2 0.2 0.2 )
  )
)
```

---

# Attribute Set

**Full name:** Shared, Set, AttributeSet

**Drawable:** No

**Parent Class Heirarchy:** Shared, Set

**Binary type:** attr

**Ascii type:** AttributeSet

**Binary size:** 0

**Parent Objects:** any AttributeSetList, any Geometry, any Group, any CapAttributeSet

**Format:** No Data

**Subobjects:** 1 AmbientCoefficient (optional) 1 DiffuseColor (optional) 1 HighlightState (optional) 1 Normal (optional) 1 ShadingUV (optional) 1 SpecularColor (optional) 1 SpecularControl (optional) 1 SurfaceTangent (optional) 1 SurfaceUV (optional) 1 TransparencyColor (optional) 1 SurfaceShader (optional)

**Inherited:** No

**Referencable:** Yes

## Description:

A attribute set groups sets of unique attributes together and is associated with a vertex, face, or an entire geometry. Any object that is an Element may be placed in an attribute set.

An attribute set also may be placed in a group. The various attributes in an attribute set are inherited to nodes lower than it in a hierarchy.

## Data structure:

## Text samples:

```
Container (  
  Mesh (  
    ...  
  )  
  Container (  
    VertexAttributeSetList (  
      30 Exclude 2  
      29 30  
    )  
    ...  
    Container (  
      AttributeSet ( )  
      DiffuseColor ( 0 1 0 )  
      SurfaceUV ( 0.87 0.57 )  
    )  
    ...  
  )  
)
```

---

# Orthographic Camera

**Full name:** Shared, Shape, Camera, OrthographicCamera

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Camera

**Binary type:** orth

**Ascii type:** OrthographicCamera

**Binary size:** 16

**Parent Objects:** SOMETIMES: ViewHints

**Format:** Data Format

**Subobjects:** 1 CameraPlacement (optional, default) 1 CameraViewPort (optional, default) 1 CameraRange (optional, default)

**Inherited:** No

**Referencable:** Yes

## Description:

The lens characteristics are set with the dimensions of a rectangular view port in the frame of the camera.

## Data structure:

Float32 left

Float32 top

Float32 right

Float32 bottom

left < right

bottom < top

## Text samples:

```
OrthographicCamera (  
  -1 -1 1 1  
)
```

```
Container (  
  OrthographicCamera (  
    -1 -1 1 1  
  )  
  CameraPlacement (  
    0 0 20  
    0 0 0  
    1 0 0  
  )  
  CameraRange (  
    1 25  
  )  
)
```

---

# View Angle Aspect Camera

**Full name:** Shared, Shape, Camera, ViewAngleAspectCamera

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Camera

**Binary type:** vana

**Ascii type:** ViewAngleAspectCamera

**Binary size:** 8

**Parent Objects:** SOMETIMES: ViewHints

**Format:** Data Format

**Subjects:** 1 CameraPlacement (optional, default)1 CameraViewPort (optional, default)1 CameraRange (optional, default)

**Inherited:** No

**Referencable:** Yes

## Description:

A perspective camera specified in terms of the minimum view angle and the aspect ratio of X to Y.

## Data structure:

Float32 fieldOfView

Float32 aspectRatioXtoY

$0 < \text{fieldOfView}$

$0 < \text{aspectRatioXtoY}$

## Text samples:

```
ViewAngleAspectCamera (  
  1.7 1.0  
)
```

```
Container (  
  ViewAngleAspectCamera (  
    1.7 1.0  
  )  
  CameraPlacement (  
    0 0 20  
    0 0 0  
    1 0 0  
  )  
  CameraRange (  
    1 25  
  )  
)
```

---

# View Plane Camera

**Full name:** Shared, Shape, Camera, ViewPlaneCamera



**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Camera

**Binary type:** vwpl

**Ascii type:** ViewPlaneCamera

**Binary size:** 20

**Parent Objects:** SOMETIMES: ViewHints

**Format:** Data Format

**Subobjects:** 1 CameraPlacement (optional, default)1 CameraViewPort (optional, default)1

CameraRange (optional, default)

**Inherited:** No

**Referencable:** Yes

**Description:**

A view plane camera is a view angle aspect camera specified in terms of an arbitrary view plane. This is most useful when setting the camera to look at a particular object.

The viewPlane is set to distance from the camera to the object.

The halfWidth is set to half the width of the cross section of the object, and the halfHeight equal to the halfWidth divided by the aspect ratio of the viewPort.

This is the only perspective camera with specifications for off-axis viewing, which is desirable for scrolling.

**Data structure:**

Float32 viewPlane

Float32 halfWidthAtViewPlane

Float32 halfHeightAtViewPlane

Float32 centerXOnViewPlane

Float32 centerYOnViewPlane

0 < viewPlane

0 < halfWidthAtViewPlane

0 < halfHeightAtViewPlane

centerXOnViewPlane, centerYOnViewPlane may be any value

**Text samples:**

```
ViewPlaneCamera (  
    ...  
)
```

```
Container (  
    ViewPlaneCamera (  
        20  
        15.0 15.0  
        18 29  
    )  
    CameraPlacement (  
        0 0 20
```

```
    0 0 0
    1 0 0
  )
  CameraRange (
    1 25
  )
)
```

---

## Box

**Full name:** Shared, Shape, Geometry, Box

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** box

**Ascii type:** Box

**Binary size:** 0 or 48

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 FaceAttributeSetList (optional, nObjects = 6)1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

This is a rectangular parallelepiped

A size of zero indicates the default values, helpful in instantiating a unit-cube.

The Face Attribute Set List subobject assigns color to the following faces:

Face ^ orientation at origin + orientation

Face ^ orientation at origin

Face ^ majorAxis at origin + majorAxis

Face ^ majorAxis at origin

Face ^ minorAxis at origin + minorAxis

Face ^ minorAxis at origin

Basically, the faces perpendicular to the orientation direction are assigned first, then the majorAxis, then the minorAxis.

### Data structure:

Vector3D orientation

Vector3D majorAxis

Vector3D minorAxis

Point3D origin

For 0-sized objects, default is:

1 0 0 # orientation  
0 1 0 # majorAxis  
0 0 1 # minorAxis  
0 0 0 # origin

### Text samples:

Box ( )

```
Box (  
  2 0 0  
  0 1 1  
  2 3 0  
  0 0 0  
)
```

```
Container (  
  Box (  
    Container (  
      FaceAttributeSetList (  
        6 Exclude 0  
      )  
      Container (  
        AttributeSet (  
          DiffuseColor ( 1 0 0 )  
        )  
      )  
      Container (  
        AttributeSet (  
          DiffuseColor ( 0 1 1 )  
        )  
      )  
      Container (  
        AttributeSet (  
          DiffuseColor ( 0 1 0 )  
        )  
      )  
      Container (  
        AttributeSet (  
          DiffuseColor ( 1 0 1 )  
        )  
      )  
      Container (  
        AttributeSet (  
          DiffuseColor ( 0 0 1 )  
        )  
      )  
      Container (  
        AttributeSet (  
          DiffuseColor ( 1 1 0 )  
        )  
      )  
    )  
  )  
)
```

---

## Cone

**Full name:** Shared, Shape, Geometry, Cone

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** cone

**Ascii type:** Cone

**Binary size:** 0 or 48

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 Caps (optional, default) 1 FaceCapAttributeSet (optional) 1 BottomCapAttributeSet (optional) 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

**Description:**

A cone may have a cap, and may have attributes assigned to the entire geometry, to the face cap, or to the bottom cap.

The default parametrization is shown in the diagram.

**Data structure:**

Vector3D orientation

Vector3D majorAxis

Vector3D minorAxis

Point3D origin

For 0-sized objects, default is:

1 0 0 # orientation

0 1 0 # majorAxis

0 0 1 # minorAxis

0 0 0 # origin

**Text samples:**

```
Cone ( )
```

```
Cone (  
  2 0 0  
  0 1 1  
  2 3 0  
  0 0 0  
)
```

```
Container (  
  Cone ( )  
  Caps ( Bottom )  
  Container (  
    BottomCapAttributeSet ( )  
    Container (  
      AttributeSet ( )  
      DiffuseColor ( 1 0 0 )  
    )  
  )  
  Container (  
    FaceCapAttributeSet ( )
```

```
Container (
  AttributeSet ( )
  DiffuseColor ( 1 1 0 )
)
)
```

---

## Cylinder

**Full name:** Shared, Shape, Geometry, Cylinder

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** cyln

**Ascii type:** Cylinder

**Binary size:** 0 or 48

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 Caps (optional, default) 1 TopCapAttributeSet (optional) 1 FaceCapAttributeSet (optional) 1 BottomCapAttributeSet (optional) 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

A cylinder may have either top or bottom caps, and may have attributes assigned to the entire geometry, to the face cap, the bottom cap, or the top cap.

The default parametrization is shown in the diagram.

### Data structure:

Vector3D orientation

Vector3D majorRadius

Vector3D minorRadius

Point3D origin

For 0-sized objects, default is:

1 0 0 # orientation

0 1 0 # majorAxis

0 0 1 # minorAxis

0 0 0 # origin

### Text samples:

```
Cylinder ( )
```

```
Cylinder (
  2 0 0
  0 1 1
```

```

    2 3 0
    0 0 0
)
Container (
  Cylinder ( )
  Caps ( Bottom | Top )
  Container (
    BottomCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
    )
  )
  Container (
    FaceCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 1 )
    )
  )
  Container (
    TopCapAttributeSet ( )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 1 0 )
    )
  )
)
)

```

---

## Disk

**Full name:** Shared, Shape, Geometry, Disk

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** disk

**Ascii type:** Disk

**Binary size:** 0 or 36

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

This is an elliptical disk at the given origin with two vectors specifying the dimensions.

The default parametrization is shown in the diagram.

### Data structure:

Vector3D majorRadius

Vector3D minorRadius  
Point3D origin

For 0-sized objects, default is:

1 0 0 # majorRadius  
0 1 0 # minorRadius  
0 0 0 # origin

### Text samples:

```
Disk ( )
```

```
Disk (  
  2 0 0  
  0 1 1  
  0 0 0  
)
```

```
Container (  
  Cylinder ( )  
  Caps ( Bottom | Top )  
  Container (  
    BottomCapAttributeSet ( )  
    Container (  
      AttributeSet ( )  
      DiffuseColor ( 1 0 1 )  
    )  
  )  
  Container (  
    FaceCapAttributeSet ( )  
    Container (  
      AttributeSet ( )  
      DiffuseColor ( 1 1 0 )  
    )  
  )  
  Container (  
    AttributeSet ( )  
    DiffuseColor ( 1 1 0 )  
  )  
)
```

---

## Ellipse

**Full name:** Shared, Shape, Geometry, Ellipse

**Drawable:** Yes

**Parent Class Hierarchy:** Shared, Shape, Geometry

**Binary type:** elps

**Ascii type:** Ellipse

**Binary size:** 0 or 36

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

**Description:**

This is an ellipse at the given origin with two vectors specifying its dimensions.

There is no default parametrization for an ellipse.

**Data structure:**

Vector3D majorAxis

Vector3D minorAxis

Point3D origin

For 0-sized objects, default is:

1 0 0 # majorAxis

0 1 0 # minorAxis

0 0 0 # origin

**Text samples:**

```
Ellipse ( )
```

```
Ellipse (  
  2 0 0  
  0 1 1  
  0 0 0  
)
```

```
Container (  
  Ellipse ( )  
  Container (  
    AttributeSet ( )  
    DiffuseColor ( 1 1 0 )  
  )  
)
```

---

## Ellipsoid

**Full name:** Shared, Shape, Geometry, Ellipsoid

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** elpd

**Ascii type:** Ellipsoid

**Binary size:** 0 or 48

**Parent Objects:**

**Format:** Data Format



**Subobjects:** 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

**Description:**

An ellipsoid may have an attribute set attached to it.

The default parametrization is shown in the diagram. V is zero to the left of majorRadius, and is 1 to the right. U is zero at the orientation vector, and 0 at the bottom.

**Data structure:**

Vector3D orientation

Vector3D majorRadius

Vector3D minorRadius

Point3D origin

For 0-sized objects, default is:

1 0 0 # orientation

0 1 0 # majorRadius

0 0 1 # minorRadius

0 0 0 # origin

**Text samples:**

```
Sphere ( )
```

```
Sphere (  
  2 0 0  
  0 1 1  
  2 3 0  
  0 0 0  
)
```

```
Container (  
  Sphere ( )  
  Container (  
    AttributeSet ( )  
    DiffuseColor ( 1 1 0 )  
  )  
)
```

---

## General Polygon

**Full name:** Shared, Shape, Geometry, GeneralPolygon

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** gpgn

**Ascii type:** GeneralPolygon

**Binary size:** 4 + sizeof(polygons[0..nContours-1]) sizeof(PolygonData) = 4 + nVertices \* 12

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 VertexAttributeSetList (optional, nObjects = nVertices[0] + ... + nVertices[nContours-1])

1 AttributeSet (optional) 1 GeneralPolygonHint (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

A general polygon is a polygon that may be convex or may contain holes. A general polygon also assumes that all faces are planar within floating point tolerances.

Holes are indicated by specifying a contour of the generalPolygon in clockwise order.

Polygons that cross use the even-odd rule to specify holes (see diagram).

You may specify the complexity of a GeneralPolygon by adding a viewHints object.

### Data structure:

Uns32 nContours

PolygonData polygons[nContours]

where PolygonData is:

Uns32 nVertices

Point3D vertices[nVertices]

0 < nContours

2 < nVertices

### Text samples:

```
Container (
  GeneralPolygon (
    2 # nContours
    3 # nVertices
    -1 0 0
    1 0 0
    0 1.7 0
    3 # nVertices
    -1 0.4 0
    1 0.4 0
    0 2.1 0
  )
  Container (
    VertexAttributeSetList ( 6 Exclude 2 0 4 )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
    Container (
```

```

    AttributeSet ( )
    DiffuseColor ( 0 1 1 )
)
Container (
    AttributeSet ( )
    DiffuseColor ( 1 0 1 )
)
Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
)
)
Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 1 )
)
)

```

---

## Line

**Full name:** Shared, Shape, Geometry, Line

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** line

**Ascii type:** Line

**Binary size:** 24

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 VertexAttributeSetList (optional, nObjects = 2) 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

Our basic line primitive is a line segment, a simple line drawn between two vertices.

Optional vertex attributes may be attached using a VertexAttributeSetList.

A set of attributes may be applied to the entire line segment by attaching an attribute set.

### Data structure:

Point3D start

Point3D end

### Text samples:

```
Line ( 0 0 0 1 0 0 )
```

```
Container (
  Line (
    0 0 0
```

```

    1 0 0
  )
  Container (
    VertexAttributeSetList ( 2 Exclude 0 )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
)

```

---

## Marker

**Full name:** Shared, Shape, Geometry, Marker

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** mrkr

**Ascii type:** Marker

**Binary size:** 32 + (rowBytes \* height) + padding

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

The marker is used to rasterize bitmaps parallel to the viewing plane. They are used for annotation of an image.

### Data structure:

Point3D location

Int32 xOffset

Int32 yOffset

Uns32 width

Uns32 height

Uns32 rowBytes

EndianEnum bitEndian

RawData data[height \* rowBytes]

0 < width

0 < height

$((\text{width} / 8) + ((\text{width} \& 7) > 0)) \text{ rowBytes}$

EndianEnum is:

Binary Text  
0x00000000 BigEndian  
0x00000001 LittleEndian

### Text samples:

```
Container (  
  Marker (  
    0.5 0.5 0.5 # origin  
    -28 # xOffset  
    -3 # yOffset  
    56 # width  
    6 # height  
    7 # rowBytes  
    BigEndian # bitOrder  
    0x7E3C3C667E7C18606066666066187C3C  
    0x607E7C661860066066607C1860066666  
    0x6066007E3C3C667E6618  
  )  
  Container (  
    AttributeSet ( )  
    DiffuseColor ( 0.8 0.2 0.6 )  
  )  
)
```

---

## Mesh

**Full name:** Shared, Shape, Geometry, Mesh

**Drawable:** Yes

**Parent Class Hierarchy:** Shared, Shape, Geometry

**Binary type:** mesh

**Ascii type:** Mesh

**Binary size:**  $4 + nVertices * 12 + 8 + (nFaces + nContours) * \text{sizeof}(\text{faces}[0..nFaces + nContours - 1])$

$\text{sizeof}(\text{MeshFace}) = \text{sizeof}(\text{Int}) + \text{sizeof}(\text{Uns}) * nFaceVertexIndices$

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 FaceAttributeSetList (optional, nObjects = nFaces) 1 VertexAttributeSetList (optional, nObjects = nVertices) 1 MeshCorners (optional) 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

The mesh is used for representing complex topological objects. It contains enough information to determine which polygonal faces are adjacent to each other without numerical ambiguity. This metafile object contains topological as well as geometrical information.

A contour (hole) in a face is indicated by supplying a negative number for the number of vertices, and adds a hole to the previous face that was not a contour.

The size of nFaceVertexIndices and faceVertexIndices is based on the value of nVertices.

We introduce a special subobject used only with the mesh, called MeshCorners. This object allows multiple attribute sets to be attached to a single vertex, where each attribute set is bound to a set of vertex-face pairs. It can be used to place a sharp edge in the mesh (if the attribute set contains a normal, for instance).

**Data structure:**

Uns32 nVertices  
Point3D vertices[nVertices]  
Uns32 nFaces  
Uns32 nContours  
MeshFace faces[nFaces + nContours]

where MeshFace is:

Int32 nFaceVertexIndices  
Uns32 faceVertexIndices[nFaceVertexIndices]

3 nVertices  
3 nFaceVertexIndices

**Text samples:**

```
Mesh (  
  10 # nVertices  
  -1 1 1  
  -1 1 -1  
  1 1 -1  
  1 -1 -1  
  1 -1 1  
  0 -1 1  
  -1 -1 0  
  -1 -1 -1  
  1 1 1  
  -1 0 1  
  7 # nFaces  
  0 # nContours  
  3 6 5 9  
  5 7 6 9 0 1  
  4 2 3 7 1  
  4 2 8 4 3  
  4 1 0 8 2  
  5 4 8 0 9 5  
  5 3 4 5 6 7  
)
```

---

## NURB Curve

**Full name:** Shared, Shape, Geometry, NURBCurve

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** nrbc

**Ascii type:** NURBCurve

**Binary size:**  $8 + (nPoints * 12) + ((nPoints + order) * 4)$

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** Yes

**Description:**

NURB curves are Non-Uniform Rational B-spline curves. A rational B-spline curve is a curve in 4D space, which has been projected down to 3D space. Thus, the control points for a 3D rational curve have four components - x, y, z, and w (usually known as the weight). For such a point, the corresponding point in 3D space is (x/w, y/w, z/w)

Weights (w) are always positive.

**Data structure:**

Uns32 order

Uns32 nPoints

RationalPoint4D points[nPoints]

Float32 knots[order + nPoints]

2 order

2 nPoints

$0 < points[...].w$  (weights of points)

**Text samples:**

```
NURBCurve (  
  4 7 # order, nPoints  
  0 0 0 1 # points  
  1 1 0 1  
  2 0 0 1  
  3 1 0 1  
  4 0 0 1  
  5 1 0 1  
  6 0 0 1  
  0 0 0 0 0.25 0.5 0.75 1 1 1 1 # knots  
)
```

## NURB Patch

**Full name:** Shared, Shape, Geometry, NURBPatch

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** nrbp

**Ascii type:** NURBPatch

**Binary size:**  $16 + (16 * \text{numColumns} * \text{numRows}) + ((\text{uOrder} + \text{numColumns}) * 4) + ((\text{vOrder} + \text{numRows}) * 4)$

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 TrimCurves (optional)

**Inherited:** No

**Referencable:** Yes

**Description:**

Non-Uniform Rational B-Spline (NURB) Patches are closed under projective transformations, can represent quadrics exactly, and can be refined locally to allow additional detail.

The default parametrization is given by the knot vectors.

Weights (w) are always positive.

**Data structure:**

Uns32 uOrder

Uns32 vOrder

Uns32 numColumns

Uns32 numRows

RationalPoint4D points[numMPoints\*numNPoints]

Float32 uKnots[uOrder + numColumns]

Float32 vKnots[vOrder + numRows]

2 numColumns

2 numRows

2 uOrder

2 vOrder

0 < points[...].w (weights of points)

**Text samples:**

```
NURBPatch (  
  4 4 4 4 # u,v order, num M,N points  
  -2 2 0 1   -1 2 0 1   1 2 0 1   2 2 0 1  
  -2 2 0 1   -1 2 0 1   1 0 5 1   2 2 0 1  
  -2 -2 0 1  -1 -2 0 1   1 -2 0 1   2 -2 0 1  
  -2 -2 0 1  -1 -2 0 1   1 -2 0 1   2 -2 0 1  
  0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 # knots  
)
```

---

## Point

**Full name:** Shared, Shape, Geometry, Point



**Drawable:** Yes  
**Parent Class Hierarchy:** Shared, Shape, Geometry  
**Binary type:** pnt  
**Ascii type:** Point  
**Binary size:** 12  
**Parent Objects:**  
**Format:** Data Format  
**Subobjects:** 1 AttributeSet (optional)  
**Inherited:** No  
**Referencable:** Yes

**Description:**

The basic point primitive is an infinitesimally small point in space. It is specified as a 3D point plus an optional attribute set.

A 3D point has no default parametrization.

**Data structure:**

Point3D point

**Text samples:**

Point ( 0 1 2 )

---

## Polygon

**Full name:** Shared, Shape, Geometry, Polygon  
**Drawable:** Yes  
**Parent Class Hierarchy:** Shared, Shape, Geometry  
**Binary type:** plyg  
**Ascii type:** Polygon  
**Binary size:**  $4 + nVertices * 12$   
**Parent Objects:**  
**Format:** Data Format  
**Subobjects:** 1 VertexAttributeSetList (optional, nObjects = nVertices) 1 AttributeSet (optional)  
**Inherited:** No  
**Referencable:** Yes

**Description:**

The polygon is convex with no holes. To describe concave polygons or polygons with holes, use the general polygon primitive.

The points that make up a polygons face are assumed to be planar within floating point tolerances.

**Data structure:**

Uns32 nVertices  
Point3D vertices[nVertices]

2 nVertices

**Text samples:**

```
Polygon (  
  4  
  0 1 1  
  0 -1 1  
  0 -1 -1  
  0 1 -1  
)
```

---

## Poly Line

**Full name:** Shared, Shape, Geometry, PolyLine

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** pyl

**Ascii type:** PolyLine

**Binary size:** 4 + nVertices \* 12

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 VertexAttributeSetList (optional, nObjects = nVertices) 1 GeometryAttributeSetList (optional, nObjects = nVertices - 1) 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

**Description:**

An extension of the basic line primitive is a polyline, where simple lines are drawn between adjacent points in a point list

A polyline is NOT closed, and the last point is never connected to the first point.

A polyline has no default parametrization.

**Data structure:**

Uns32 nVertices  
Point3D vertices[nVertices]

2 nVertices

**Text samples:**

```
Container (
```

```
PolyLine (
  4
  -1 -0.5 -0.25
  -0.5 1.5 0.45
  0 0 0
  1.5 1.5 1
)
Container (
  AttributeSet ( )
  DiffuseColor ( 0.4 0.2 0.9 )
)
)
```

---

## Torus

**Full name:** Shared, Shape, Geometry, Torus

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** tors

**Ascii type:** Torus

**Binary size:** 0 or 52

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

The orientation length specifies the radius of the circular along the orientation vector of the torus cross-section.

The major and minor axes are vectors to the center of the torus cross-section (as in the diagram).

The ratio is the change in the orientation length in the axial direction. A ratio of 2, for example, creates a fatter torus cross-section along the major and minor axes, a ratio of 0.5 creates a fatter cross-section along the orientation.

As far as anyone knows, the torus is useful for drawing donuts and bagels, and makes a great demo.

The default parametrization is shown in the diagram.

### Data structure:

Vector3D orientation

Vector3D majorAxis

Vector3D minorAxis

Point3D origin

Float32 ratio

For 0-sized objects, default is:

```
1 0 0 # orientation
0 1 0 # majorAxis
0 0 1 # minorAxis
0 0 0 # origin
1 # ratio
```

### Text samples:

```
Torus ( )
```

```
Torus (
  2 0 0
  0 1 1
  2 3 0
  0 0 0
  1
)
```

```
Container (
  Torus ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 1 1 0 )
  )
)
```

---

## Triangle

**Full name:** Shared, Shape, Geometry, Triangle

**Drawable:** Yes

**Parent Class Hierarchy:** Shared, Shape, Geometry

**Binary type:** trng

**Ascii type:** Triangle

**Binary size:** 36

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 VertexAttributeSetList (optional, nObjects = 3) 1 AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

The most basic polygon is a triangle, which contains 3 points.

A VertexAttributeSetList may be used to attach attribute sets to the vertices (containing three vertex attribute sets) or an optional AttributeSet may be added to attach to the face.

There is no default parametrization for a triangle.

**Data structure:**

Point3D vertices[3]

**Text samples:**

```
Container (
  Triangle (
    -1 -0.5 -0.25
    0 0 0
    -0.5 1.5 0.45
  )
  Container (
    VertexAttributeSetList ( 3 Exclude 0 )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 1 0 )
    )
    Container (
      AttributeSet ( )
      DiffuseColor ( 0 0 1 )
    )
  )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.8 0.5 0.2 )
  )
)
```

---

## Tri Grid

**Full name:** Shared, Shape, Geometry, TriGrid

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Geometry

**Binary type:** trig

**Ascii type:** TriGrid

**Binary size:** 8 + (nColumns \* nRows \* 12)

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 FaceAttributeSetList (optional, nObjects = (numNVertices - 1) \* (numMVertices - 1) \* 2)

1 VertexAttributeSetList (optional, nObjects = numNVertices \* numMVertices attribute sets) 1

AttributeSet (optional)

**Inherited:** No

**Referencable:** Yes

**Description:**

Points specified are given in row major order.

You may add a FaceAttributeSetList to attach a set of attributes for each of the triangles generated by this primitive.

You may also add a VertexAttributeSetList to attach attributes to each vertex.

**Data structure:**

Uns32 nColumns

Uns32 nRows

Point3D points[numMVertices \* numNVertices]

2 nColumns

2 nRows

**Text samples:**

```
Container (
  TriGrid (
    3 4 # nUVertices nVVertices
    -1 1 1      -0.5 1 0  0 1 0
    0.7 1 0.5  -1 0 0    -0.5 0 0.3
    0 0.2 0    0.5 0 0   -1 -1 0
    -0.5 -1 0  0 -1 0.1  0.2 -1.3 0.2
  )
  Container (
    FaceAttributeSetList ( 12 Include 1 5 )
    Container (
      AttributeSet ( )
      DiffuseColor ( 1 0 0.5 )
    )
  )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0.8 0.7 0.3 )
  )
)
```

## Group

**Full name:** Shared, Shape, Group

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape

**Binary type:** grup

**Ascii type:** Group

**Binary size:** 0

**Parent Objects:** none

**Format:** No Data

**Subobjects:** none

**Inherited:** No

**Referencable:** Yes

**Description:**

The group is useful for grouping any type of shared objects together.

It is delimited by an end group object.

**Data structure:**

**Text samples:**

```
BeginGroup ( Group ( ) )  
  CString ( This is the first day of the rest of your life. )  
  Torus ( )  
EndGroup ( )
```

---

## Display Group

**Full name:** Shared, Shape, Group, DisplayGroup

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Group

**Binary type:** dspg

**Ascii type:** DisplayGroup

**Binary size:** 0

**Parent Objects:**

**Format:** No Data

**Subobjects:** 1 DisplayGroupState (optional)

**Inherited:** No

**Referencable:** Yes

**Description:**

A display group contains only objects that are drawable.

A display group adds the ability to be traversed for various operations via the DisplayGroupState subobject.

It is delimited by an end group object.

**Data structure:**

**Text samples:**

---

## IO Proxy Display Group

**Full name:** Shared, Shape, Group, DisplayGroup, IOProxyDisplayGroup

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Group, DisplayGroup

**Binary type:** iopx

**Ascii type:** IOProxyDisplayGroup

**Binary size:** 0

**Parent Objects:**

**Format:** No Data

**Subobjects:** 1 DisplayGroupState (optional, default)

**Inherited:** No

**Referencable:** Yes

### **Description:**

The IO proxy display group contains drawable objects that are similar representations of the same object in different formats. For example, if it is known that a particular application does not understand NURBPatches, the writing application may write the NURBPatch in an IO proxy group along with a mesh which is the tessellated NURBPatch.

The objects in a IO proxy display group appear in their preferential order. The first object is the most preferred representation, the last object the least. The first object that is understood by a reading application should be used.

You may specify a group of objects inside a IOProxyDisplayGroup, as a group (up to its EndGroup) delimiter is a single object.

It is understood that ONLY the first understood object in an IO proxy display group is traversed while drawing, bounding, or picking.

In other words, if an IO proxy display group contains many objects, only one of them will be drawn when it comes time to render an image, etc.

### **Data structure:**

### **Text samples:**

```
BeginGroup ( IOProxyDisplayGroup ( ) )
  Mesh (
    8
    0 0 0
    0 0 1
    0 1 0
    1 0 0
    1 1 0
    0 1 1
    1 0 1
    1 1 1
    ... etc.
  )
  Box ( )
EndGroup ( )
```



```
BeginGroup ( IOProxyDisplayGroup ( ) )
  NURBPatch (          # preferred object
    ...
  )
  DisplayGroup ( ) # 2nd choice object
  Translate ( 1 2 3 )
  Box ( )
EndGroup ( )
EndGroup ( )
```

---

## Ordered Display Group

**Full name:** Shared, Shape, Group, DisplayGroup, OrderedDisplayGroup

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Group, DisplayGroup

**Binary type:** ordg

**Ascii type:** OrderedDisplayGroup

**Binary size:** 0

**Parent Objects:**

**Format:** No Data

**Subobjects:** 1 DisplayGroupState (optional, default)

**Inherited:** No

**Referencable:** Yes

### Description:

The ordered display group is simply a display group except that objects are sorted by type. Objects always appear in an ordered group in the following order:

Transforms

Styles

AttributeSets

Shaders

Geometries

DisplayGroups

It is delimited by an end group object.

**Data structure:**

**Text samples:**

---

## Info Group

**Full name:** Shared, Shape, Group, InfoGroup

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Group

**Binary type:** info

**Ascii type:** InfoGroup

**Binary size:** 0

**Parent Objects:** none

**Format:** No Data

**Subobjects:** none

**Inherited:** No

**Referencable:** Yes

**Description:**

An info group contains nothing but String objects. It is used to add human-readable information pertaining to a files origin or history. A use that comes to mind is copyright notices.

The info group object should be preserved by a reading application, and appended with additional information if a file is re-written.

It is delimited by an end group object.

**Data structure:**

**Text samples:**

```
BeginGroup ( InfoGroup ( ) )
  CString (
    Copyright 1995 Apple Computer, Inc. )
  CString (
    Author: Bonanza Jellybean )
EndGroup ( )
```

---

## Light Group

**Full name:** Shared, Shape, Group, LightGroup

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Group

**Binary type:** lghg

**Ascii type:** LightGroup

**Binary size:** 0

**Parent Objects:** none

**Format:** No Data

**Subobjects:** none

**Inherited:** No

**Referencable:** Yes

**Description:**

A light group contains nothing but lights.

It is delimited by an end group object.

**Data structure:**

**Text samples:**

```
BeginGroup ( LightGroup ( ) )  
  AmbientLight ( )  
  DirectionalLight ( 1 0 0 False )  
EndGroup ( )
```

---

## Ambient Light

**Full name:** Shared, Shape, Light, AmbientLight

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Light

**Binary type:** ambn

**Ascii type:** AmbientLight

**Binary size:** 0

**Parent Objects:**

**Format:** No Data

**Subobjects:** 1 LightData (optional, default)

**Inherited:** No

**Referencable:** Yes

**Description:**

An ambient light supplies light that comes from secondary reflections.

In lieu of other light sources, the ambient light illuminates the scene with a flat, uniform light.

**Data structure:**

**Text samples:**

```
AmbientLight ( )  
  
Container (  
  AmbientLight ( )  
  LightData (  
    EcTrue # isOn  
    1.0 # intensity  
    1 0 0 # red color  
  )  
)
```

---

## Directional Light

**Full name:** Shared, Shape, Light, DirectionalLight

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Light

**Binary type:** drct

**Ascii type:** DirectionalLight

**Binary size:**

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 LightData (optional, defaults)

**Inherited:** No

**Referencable:** Yes

### **Description:**

A directional light is far enough away from the scene that we may treat it as though it were infinitely far away. This produces shading results faster than any other type of light (except ambient).

It is specified with a vector pointing in the same direction as the light rays, an attenuation and a boolean value indicating whether this light casts shadows or not.

### **Data structure:**

Vector3D direction

Boolean castsShadows

$|\text{direction}| = 1.0$

### **Text samples:**

```
DirectionalLight ( 1 0 0 True )
```

```
Container (
  DirectionalLight ( 1 0 0 True )
  LightData (
    True
    0.4
    1 0 0
  )
)
```

---

## **Point Light**

**Full name:** Shared, Shape, Light, PointLight

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Light

**Binary type:** pntl

**Ascii type:** PointLight

**Binary size:**

**Parent Objects:****Format:** Data Format**Subobjects:** 1 LightData (optional, defaults)**Inherited:** No**Referencable:** Yes**Description:**

A point light is a light at an infinitesimally small point in space. It may be attenuated or it may cast shadows.

**Data structure:**

Point3D location

Attenuation attenuation

Boolean castsShadows

where Attenuation is the structure:

Float32 c0

Float32 c1

Float32 c2

attenuation is computed, using d as the distance from location:

$$\frac{1}{c_0 + c_1 * d + c_2 * d^2}$$

$$0 < c_0$$

$$0 < c_1$$

$$0 < c_2$$

attenuation is not clamped to [0,1] to allow for lighting washout (such as in a nuclear explosion)

**Text samples:**

```
PointLight (
  12 23 2
  0 0 1 # InverseDistanceSquared
  True
)

Container (
  PointLight (
    12 23 2
    0 0 1 # InverseDistanceSquared
    True
  )
  LightData (
    True
    0.4
    1 0 0
  )
)
```

---

## Spot Light

**Full name:** Shared, Shape, Light, SpotLight

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Light

**Binary type:** spot

**Ascii type:** SpotLight

**Binary size:**

**Parent Objects:**

**Format:** Data Format

**Subobjects:** 1 LightData (optional, defaults)

**Inherited:** No

**Referencable:** Yes

### Description:

A spot light radiates with a circular cone of light that tapers toward the edge of the cone.

The hotSpotAngle is the angle (in radians) from the axis of the spot light for which the spot light has maximum, constant intensity. The outer angle is the angle for which the light falls to zero. Between these two, the light intensity tapers to zero according to the FallOff enumerated type.

### Data structure:

Point3D location

Vector3D orientation

Boolean castsShadows

Attenuation attenuation

Float32 hotAngle

Float32 outerAngle

FallOffEnum fallOff

|orientation| = 1

Attenuation is described in the Point Light

$0 < \text{hotAngle} < \text{outerAngle}$

FallOffEnum is:

Binary Text

0x00000000 None

0x00000001 Linear

0x00000002 Exponential

0x00000003 Cosine

### Text samples:

```
SpotLight (
  12 0 0
  0 1 0
  True
  0 0 1 # InverseDistanceSquared
  0.7 # hotAngle
  0.8 # outerAngle
  Cosine
)

Container (
  SpotLight (
    12 0 0
    0 1 0
    True
    0 0 1 # InverseDistanceSquared
    0.7 # hotAngle
    0.8 # outerAngle
    Cosine
  )
  LightData (
    True
    0.4
    1 0 1
  )
)
```

---

## Lambert Illumination

**Full name:** Shared, Shape, Shader, IlluminationShader, LambertIllumination

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Shader, IlluminationShader

**Binary type:** lmil

**Ascii type:** LambertIllumination

**Binary size:** 0

**Parent Objects:**

**Format:** No Data

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

### Description:

The lambertian illumination model.

### Data structure:

### Text samples:

```
LambertIllumination ( )
```

---

# Phong Illumination

**Full name:** Shared, Shape, Shader, IlluminationShader, PhongIllumination

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Shader, IlluminationShader

**Binary type:** phil

**Ascii type:** PhongIllumination

**Binary size:** 0

**Parent Objects:**

**Format:** No Data

**Subobjects:**

**Inherited:** Yes

**Referencable:** Yes

## Description:

The phong illumination model.

## Data structure:

## Text samples:

PhongIllumination ( )

---

# Texture Shader

**Full name:** Shared, Shape, Shader, SurfaceShader, TextureShader

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Shader, SurfaceShader

**Binary type:** txsu

**Ascii type:** TextureShader

**Binary size:** 0

**Parent Objects:**

**Format:** No Data

**Subobjects:** 1 PixmapTexture (required)

**Inherited:** Yes

**Referencable:** Yes

## Description:

The texture shader is used to perform shading using a texture (in this case, a PixmapTexture).

## Data structure:

## Text samples:



```
Container (
  TextureShader ( )
  PixmapTexture (
    ...
  )
)
```

---

## Backfacing Style

**Full name:** Shared, Shape, Style, BackfacingStyle

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Style

**Binary type:** bckf

**Ascii type:** BackfacingStyle

**Binary size:**

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

### Description:

The backfacing style tells a renderer how to clip backfacing polygons while rendering.

### Data structure:

BackfacingEnum backfacing

where BackfacingEnum is:

Text Binary

0x00000000 Both

0x00000001 Culled

0x00000002 Flipped

### Text samples:

BackfacingStyle ( Culled )

---

## Fill Style

**Full name:** Shared, Shape, Style, FillStyle

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Style

**Binary type:** fist  
**Ascii type:** FillStyle  
**Binary size:** 4  
**Parent Objects:**  
**Format:** Data Format  
**Subobjects:** none  
**Inherited:** Yes  
**Referencable:** Yes

**Description:**  
The fill style tells a renderer what parts of a polygon to draw.

**Data structure:**  
FillStyleEnum fillStyle

where FillStyleEnum is:

Text Binary  
0x00000000 Filled  
0x00000001 Edges  
0x00000002 Points  
0x00000003 Empty

**Text samples:**

FillStyle ( Edges )

---

## Highlight Style

**Full name:** Shared, Shape, Style, HighlightStyle  
**Drawable:** Yes  
**Parent Class Heirarchy:** Shared, Shape, Style  
**Binary type:** high  
**Ascii type:** HighlightStyle  
**Binary size:** 0  
**Parent Objects:**  
**Format:** No Data  
**Subobjects:** 1 AttributeSet (required)  
**Inherited:** Yes  
**Referencable:** Yes

**Description:**  
The highlight style sets the binding for highlighting features of a geometry via the HighlightState attribute. The attribute set subobject sets the highlight attribute set.

**Data structure:****Text samples:**

```
Container (
  HighlightStyle ( )
  Container (
    AttributeSet ( )
    DiffuseColor ( 0 0 1 )
  )
)
```

---

## Interpolation Style

**Full name:** Shared, Shape, Style, InterpolationStyle

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Style

**Binary type:** intp

**Ascii type:** InterpolationStyle

**Binary size:** 4

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

The interpolation style tells a renderer how to interpolate shading values on a polygon.

**Data structure:**

InterpolationStyleEnum interpolationStyle

where InterpolationStyleEnum is:

Binary Text

0x00000000 None

0x00000001 Vertex

0x00000002 Pixel

**Text samples:**

```
InterpolationStyle ( Vertex )
```

---

## Orientation Style

**Full name:** Shared, Shape, Style, OrientationStyle

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Style

**Binary type:** ornt

**Ascii type:** OrientationStyle

**Binary size:** 4

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

The Orientation style is used to change the orientation of polygons.

**Data structure:**

OrientationEnum orientation

where OrientationEnum is:

Binary Text

0x00000000 CounterClockwise

0x00000001 Clockwise

**Text samples:**

OrientationStyle ( Clockwise )

---

## Pick ID Style

**Full name:** Shared, Shape, Style, PickIDStyle

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Style

**Binary type:** pkid

**Ascii type:** PickIDStyle

**Binary size:** 4

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

The pick ID style is used to allow the user to insert ids within a hierarchy to aid in picking a hierarchy.

**Data structure:**

Uns32 id

**Text samples:**

PickIDStyle ( 23 )

---

## Pick Parts Style

**Full name:** Shared, Shape, Style, PickPartsStyle

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Style

**Binary type:** pkpt

**Ascii type:** PickPartsStyle

**Binary size:**

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

The pick parts style determines the level of granularity for picking.

**Data structure:**

PickPartsFlags pickParts

where PickPartsFlags is:

Text Binary

0x00000000 Object

0x00000001 Face

0x00000002 Edge

0x00000004 Vertex

default is:

Object

**Text samples:**

PickPartsStyle ( Object | Vertex )

---

## Receive Shadows Style

**Full name:** Shared, Shape, Style, ReceiveShadowsStyle

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Style

**Binary type:** rcsh

**Ascii type:** ReceiveShadowsStyle

**Binary size:** 4

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

The receive shadows style determines whether a geometry receives shadows when rendering. It is coupled with the casts shadows field in all lights, excluding the ambient light.

**Data structure:**

Boolean receiveShadows

**Text samples:**

ReceiveShadowsStyle ( True )

---

## Subdivision Style

**Full name:** Shared, Shape, Style, SubdivisionStyle

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Style

**Binary type:** sbdv

**Ascii type:** SubdivisionStyle

**Binary size:** (subdivisionMethod == Constant) ? 12 : 8

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

The subdivision style tells a geometric decomposition the coarseness of a geometric primitive tessellation. There are three methods of subdivision: constant, world space, and screen space subdivision.

Constant subdivision supplies 2 integral values, which indicate the number of sections the u and v axes of a decomposition should be divided into.

The Screen Space value indicates average size of a single polygon in a tessellation in screen space.

The world space value indicates the average size of a single polygon in a tessellation in world space.

**Data structure:**

This object has two forms, based on the subdivision method field:

for subdivisionMethod == WorldSpace or ScreenSpace the structure is:

```
SubdivisionMethodEnum subdivisionMethod  
Float32 value1
```

for subdivisionMethod == Constant, the values are integral:

```
SubdivisionMethodEnum subdivisionMethod  
Uns32 value1  
Uns32 value2
```

where SubdivisionMethodEnum is:

```
Binary Text  
0x00000000 Constant  
0x00000001 WorldSpace  
0x00000002 ScreenSpace
```

**Text samples:**

```
SubdivisionStyle (  
    Constant 12 12  
)
```

```
SubdivisionStyle (  
    WorldSpace 50  
)
```

```
SubdivisionStyle (  
    ScreenSpace 50  
)
```

---

## Matrix Transform

**Full name:** Shared, Shape, Transform, Matrix

**Drawable:** Yes

**Parent Class Hierarchy:** Shared, Shape, Transform

**Binary type:** mtrx

**Ascii type:** Matrix

**Binary size:** 64

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

A custom, invertible matrix transform.

**Data structure:**

Matrix4x4 matrix

matrix is invertible

**Text samples:**

---

## Quaternion Transform

**Full name:** Shared, Shape, Transform, Quaternion

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Transform

**Binary type:** qtrn

**Ascii type:** Quaternion

**Binary size:** 16

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

The quaternion specifies three axes of rotation and a twist value.

Useful for user interface.

**Data structure:**

Float32 w

Float32 x

Float32 y

Float32 z

**Text samples:**

Quaternion ( 0.2 0.7 0.2 1.57 )

---



# Rotate Transform

**Full name:** Shared, Shape, Transform, Rotate

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Transform

**Binary type:** rott

**Ascii type:** Rotate

**Binary size:**

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

## Description:

Rotate about the X, Y, or Z axes.

## Data structure:

AxisEnum axis

Float32 radians

AxisEnum is:

Binary Text

0x00000000 X

0x00000001 Y

0x00000002 Z

## Text samples:

Rotate ( X 1.57 )

---

# Rotate About Axis Transform

**Full name:** Shared, Shape, Transform, RotateAboutAxis

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Transform

**Binary type:** rtaa

**Ascii type:** RotateAboutAxis

**Binary size:** 28

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

Rotate about an arbitrary axis in space.

**Data structure:**

Point3D origin  
Vector3D orientation  
Float32 radians

|orientation| = 1

**Text samples:**

```
RotateAboutAxis (  
    20 0 0 # origin  
    0 1 0 # orientation  
    1.57 # radians  
)
```

---

## Rotate About Point Transform

**Full name:** Shared, Shape, Transform, RotateAboutPoint

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape, Transform

**Binary type:** rtap

**Ascii type:** RotateAboutPoint

**Binary size:** 20

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** Yes

**Referencable:** Yes

**Description:**

To rotate about the X, Y, or Z axes at an arbitrary point in space.

**Data structure:**

AxisEnum axis  
Float32 radians  
Point3D origin

AxisEnum is:

Binary Text

0x00000000 X

0x00000001 Y  
0x00000002 Z

**Text samples:**

---

## Scale Transform

**Full name:** Shared, Shape, Transform, Scale  
**Drawable:** Yes  
**Parent Class Heirarchy:** Shared, Shape, Transform  
**Binary type:** scal  
**Ascii type:** Scale  
**Binary size:**  
**Parent Objects:**  
**Format:** Data Format  
**Subobjects:** none  
**Inherited:** Yes  
**Referencable:** Yes

**Description:**  
A scale transform.

**Data structure:**  
Vector3D scale

scale.x 0.0  
scale.y 0.0  
scale.z 0.0

**Text samples:**

Scale ( 1 1 2 )

---

## Translate Transform

**Full name:** Shared, Shape, Transform, Translate  
**Drawable:** Yes  
**Parent Class Heirarchy:** Shared, Shape, Transform  
**Binary type:** trns  
**Ascii type:** Translate  
**Binary size:** 12  
**Parent Objects:**  
**Format:** Data Format

**Subobjects:** none  
**Inherited:** Yes  
**Referencable:** Yes

**Description:**  
A translate transform.

**Data structure:**  
Vector3D translate

**Text samples:**

Translate ( 1 2 100 )

---

## Unknown Binary

**Full name:** Shared, Shape, UnknownBinary  
**Drawable:** Yes  
**Parent Class Hierarchy:** Shared, Shape  
**Binary type:** ukbn  
**Ascii type:** UnknownBinary  
**Binary size:** 12 +  
**Parent Objects:**  
**Format:** Data Format  
**Subobjects:**  
**Inherited:** No  
**Referencable:** Yes

### Description:

The unknown binary object is a way of transporting unknown data found in a binary file. It is an encapsulated replica of the original data found in a binary metafile, containing the object type (an Int32), the object size (in bytes), the byte order of the original file, and the data itself. The byte order is needed if unknown data is transported across different processors, and allows for parsing endian-specific primitives within the raw data block.

Unknown binary objects may be written in either the text or binary files.

When an unknown binary object is encountered in a metafile, it is up to the reading program to either:  
transport the data around  
validate it and convert it to a known object  
discard the data

Unknown objects are inherently dirty, meaning you may assume the unknown binary object may contain out-of-sync (bogus) information, as the original object may have been removed from its original context.

**Data structure:**

Int32 objectType  
Uns32 objectSize  
EndianEnum byteOrder  
RawData objectData[objectSize]

**Text samples:**

```
UnknownBinary (  
    1701605476  
    4  
    BigEndian  
    0x0AB2  
)
```

---

## Unknown Text

**Full name:** Shared, Shape, UnknownText

**Drawable:** Yes

**Parent Class Heirarchy:** Shared, Shape

**Binary type:** uktx

**Ascii type:** UnknownText

**Binary size:** sizeof(name) + sizeof(data)

**Parent Objects:** any

**Format:** Data Format

**Subobjects:**

**Inherited:** No

**Referencable:** Yes

**Description:**

The unknown text object is a way of transporting unknown data found in a text file. It is an encapsulated replica of the original data found in a text metafile, containing the object type (a String), and a text string containing the original data. In some cases, white space and comments may have been stripped from the contents field.

Unknown text objects may be written in either the text or binary files.

When an unknown text object is encountered in a metafile, it is up to the reading program to either:

- transport the data around
- validate it and convert it to a known object
- discard the data

Unknown objects are inherently dirty, meaning you may assume the unknown text object may contain out-of-sync (bogus) information, as the original object may have been removed from its original context.

**Data structure:**

String asciiName

String contents

**Text samples:**

```
UnknownText (
  Ellipsoid
)
```

---

## Macintosh Path

**Full name:** Shared, Storage, MacintoshPath

**Drawable:** No

**Parent Class Heirarchy:** Shared, Storage

**Binary type:** macp

**Ascii type:** MacintoshPath

**Binary size:** sizeof(String)

**Parent Objects:** ALWAYS: Reference

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** Yes

**Description:**

The Macintosh path specifies the pathname of an external file reference using the pathname specification found in the Inside Macintosh volumes. (essentially, a colon-based separator)

**Data structure:**

String pathName

**Text samples:**

```
Container (
  Reference ( 43 )
  MacintoshPath ( :::Foo:Bar:Models:Cheryl )
)
```

---

## Unix Path

**Full name:** Shared, Storage, UnixPath

**Drawable:** No

**Parent Class Heirarchy:** Shared, Storage

**Binary type:** unix

**Ascii type:** UnixPath

**Binary size:** sizeof(String)  
**Parent Objects:** ALWAYS: Reference  
**Format:** Data Format  
**Subobjects:** none  
**Inherited:** No  
**Referencable:** Yes

**Description:**

The unix path object serves as a way to reference files on a unix file system.

The path should obey naming standards for unix operating systems.

**Data structure:**

String unixPath

**Text samples:**

```
Container (  
  Reference ( 23 )  
  UnixPath ( ./shaders.eb )  
)
```

---

## C String

**Full name:** Shared, String, CString  
**Drawable:** No  
**Parent Class Heirarchy:** Shared, String  
**Binary type:** strc  
**Ascii type:** CString  
**Binary size:** sizeof(String)  
**Parent Objects:**  
**Format:** Data Format  
**Subobjects:** none  
**Inherited:** No  
**Referencable:** Yes

**Description:**

The CString is a way of embedding text in a metafile.

Other string types allow for internationalization.

The only allowable characters in a CString are 7-bit ASCII numbers.

The following characters may be escaped with the \ character:

'a', 'b', 'f', 'n', 'r', 't', 'v', '\', '\\'

**Data structure:**

String cString

**Text samples:**

```
CString (  
    Copyright (c) 1994 Apple Computer, Inc.  
)
```

---

## Unicode

**Full name:** Shared, String, Unicode

**Drawable:** No

**Parent Class Heirarchy:** Shared, String

**Binary type:** uncd

**Ascii type:** Unicode

**Binary size:** 4 + length \* 2

**Parent Objects:**

**Format:** Data Format

**Subobjects:** none

**Inherited:** No

**Referencable:** Yes

**Description:**

The unicode object is another way of embedding text in a metafile.

See UNICODE reference for details.

**Data structure:**

Uns32 length

RawData unicode[length \* 2]

**Text samples:**

```
Unicode (  
    6  
    0x457363686572  
)
```

---

## Pixmap Texture

**Full name:** Shared, Texture, PixmapTexture

**Drawable:** No

**Parent Class Heirarchy:** Shared, Texture



**Binary type:** txpm  
**Ascii type:** PixmapTexture  
**Binary size:** 28 + rowBytes \* height + padding  
**Parent Objects:** SOMETIMES: TextureShader  
**Format:** Data Format  
**Subobjects:** none  
**Inherited:** No  
**Referencable:** Yes

**Description:**

A generic means of transferring pixmap data. Used in the Texture Shader.

**Data structure:**

Uns32 width  
Uns32 height  
Uns32 rowBytes  
Uns32 pixelSize  
PixelFormatEnum pixelType  
EndianEnum bitOrder  
EndianEnum byteOrder  
RawData image[rowBytes \* height]

0 < width  
0 < height  
0 < pixelSize < 32  
width \* pixelSize rowBytes  
PixelFormatEnum is:

Binary Text  
0x00000000 RGB8  
0x00000001 RGB16  
0x00000002 RGB24  
0x00000003 RGB32

EndianEnum is:

Binary Text  
0x00000000 BigEndian  
0x00000001 LittleEndian

**Text samples:**

```
PixmapTexture (  
    256 256 # width/height  
    128 # rowBytes  
    32 # pixelSize  
    RGB24  
    BigEndian BigEndian  
    0x00123232...
```

0x...  
)

---

## View Hints

**Full name:** Shared, ViewHints

**Drawable:** No

**Parent Class Heirarchy:** Shared

**Binary type:** vwhn

**Ascii type:** ViewHints

**Binary size:** 0

**Parent Objects:** none

**Format:** No Data

**Subobjects:** 1 Renderer (optional) 1 Camera (optional) many Lights (optional) 1 AttributeSet (optional)  
1 ImageDimensions (optional) 1 ImageMask (optional) 1 ImageClearColor (optional)

**Inherited:** No

**Referencable:** Yes

### Description:

The subobjects of the view hints object specifies the preferences supplied by a writing application when rendering a scene.

The semantic to be followed when a view hints object is encountered in the metafile is that the view hints is specified previous to a list of objects to be rendered to that particular view hints preference. The subobjects of the view hints object are inherited from the previous view hints in a metafile.

For example, if a modelling application contains 10 camera locations for viewing various portions of a scene, it would first store the default view as the first object in a metafile, then the group representing the scene, then a view containing the second camera position, then a reference to the scene, etc.

### Data structure:

### Text samples:

```
3DMetafile ( 1 0 Normal toc> )
Container (
  ViewHints ( )
  Container (
    ViewAngleAspect ( 0.73 1.0 )
    CameraPlacement (
      0 0 30
      0 0 0
      0 1 0
    )
  )
)
DirectionalLight ( -0.7 -0.7 -0.65 )
Container (
  AttributeSet ( )
  DiffuseColor ( 0.2 0.2 0.2 )
)
```

```
    SpecularControl ( 3 )
  )
  ImageDimensions ( 200 200 )
)
refl:
BeginGroup ( DisplayGroup ( ) )
...
EndGroup ( )
Container (
  ViewHints ( )
  Container (
    ViewAngleAspect ( 0.73 1.0 )
    CameraPlacement (
      0 10 0
      0 0 0
      0 1 0
    )
  )
)
)
Reference ( 1 )
```

---

# How to use your RS232 or IRDA port for Remote Control

**Update History - skip to Introduction if this is your first visit**

**26/02/2001**

**FAQ page added**

**28/02/2001**

**FAQ No.2 updated to include extra information on component selection**

**26/08/2002**

**Theory updated**

**Hardware schematic and text updated**

**Software Section text and code updated (winsamp now version 1.3)**

**FAQ updated**

**\*\*\*\*\* This was a big update - please let me know if I've broken anything (that used to work!)**

**\*\*\*\*\***

## **Introduction**

**Firstly, an apology. There is a lot to read on this page and I'm struggling to make it even slightly visually appealing. If you feel that the subject is worthy of your attention, then I suggest you print this page out or at least save it to disk, snag the zip file mentioned in the software section (not a big download) then study it all offline.**

**Unlike some of my other stuff, this one is fairly serious. It is intended to introduce a concept and demonstrate its application rather than present the definitive finished product, but having said that, everything needed to get it working properly on a PC is included, with enough information and flexibility to enable the software as supplied to be incorporated into much more elaborate Windows front-ends if desired. The idea is not limited to PC platforms, however - most devices with serial or IRDA ports are possible candidates for the technique as described, and samples made on one platform should transfer easily to other platforms.**

**A number of people have done good work in the field of turning a PC into a 'learning' universal infrared remote control, for various appliances such as TVs, video recorders, satellite decoders etc. The common methods appear to use either an IR detector and transmitter circuit attached to the parallel port, or an 'intelligent' device accepting high-level commands attached to a serial port. A quick online search will turn up schematics, construction details, driver software and some very attractive Windows front-ends for this purpose.**

**Following on from my Furby experiments ( <http://www.veg.nildram.co.uk/furby.htm>), I set out to prove that it is possible to control the TV etc using either an IRDA port or a standard RS232 port with extremely minimalist hardware. The reasons were threefold. First, the challenge. Second, the convenience or geek value - many PCs, PDAs and other devices already come with IRDA as standard, wouldn't it be nice to use it for something different. Third, **DEVICE INDEPENDENCE AND PORTABILITY OF SAMPLES** - serial ports of one type or another are fairly ubiquitous,**

more so than parallel ports anyway, and a drawback with sampling and playback through the parallel port is the heavy dependence on processor speed and environment - samples made on one speed of machine may not work reliably on a different speed of machine (no criticism intended, but this is what I found when I tried it), so to have a good probability of success you have to sample your remotes on the machine you intend to use for playing them back - consequently, you can't simply sample your Sony CD remote and send the file off to your pal in Greenland when he loses his [ahaaa - until now :-)]. Also, most systems have more than one COM port, so it's no great hardship tying one of them up permanently for remote control. The next bit is a disclaimer of sorts, then we'll get on with the theory.

### **Disclaimer - Please read this - Important**

What I am about to describe is my own original idea, my own software, my own project. As far as I am aware, nobody else does it, or has done it, this way, but if they have, what can I say? Great minds think alike? I am not intentionally ripping off anyone else's work, and have worked long and hard to get it into its present state, for my own amusement, with no prior knowledge that it can or cannot be done or indeed that it has been attempted in this fashion. The ideas and software have been put here because I thought you might be interested in how I did it. It all works as far as I'm concerned, but I'm not asking or telling you to use it in any way, shape or form. The software (should you choose to try it) can write small text and binary files containing sample information to your disk, or whatever you run it from. Also, the Windows demo application keeps a couple of settings in the Registry to enable it to remember preferences from one session to the next. None of this should cause you any distress, but as with all things, it could always go horribly wrong, so you have been warned. If you don't want to risk using my software, you should be able to write your own using the information in the theory section. Also be advised that there is a class of IR remote control (so-called 'flash' controls) which is not supported by the software in its present form, but the project is still in a state of flux and I may add support later - the issue here is one of trying to keep it simple enough for anyone to use when they don't have the use of a proper 'scope - I'm sure you don't wish to become an expert on remotes (not that I am, but a certain amount of it inevitably rubs off).

Anyway, with the software and hardware described, I routinely control a Toshiba TV, JVC video recorder and Maspro satellite receiver either using a laptop irda port or a normal 232 port on my desktop PC, using samples made from the original remotes and without having to know details of the three different protocols actually employed by the devices. There is a good chance that most other appliances will work equally well ('flash' devices being known exceptions). If you can't get a device to work, the sampler software will at least provide information to help identify the problem if you wish to look deeper into it.

Finally, if you would like me to investigate the implementation of this technique on other platforms, you should send me non-returnable hardware and development software for the experiments, and if you make a fortune based on the idea, I certainly wouldn't mind you giving me a share :-)

Blank Frank 24th November, 1999.

## Theory - Updated 26th August, 2002

Ok. Now to the good bit. Without getting too technical, here is an idea of what's going on and why. Though there is a mind-bogglingly large selection of remote control protocols at the bit level, based on quite a number of types of protocol at the packet level, the vast majority of ir remote controls function by sending data at relatively low rates by transmitting bursts of (carrier) pulses of various lengths with periods of silence inbetween, also of various lengths. The data is encoded in the lengths of the bursts and silences, and receivers demodulate the carrier to recover the baseband data. The carrier frequency is typically somewhere in the range 36 - 40 kHz, and receivers are designed to allow signals of this frequency range through (+/- a few kHz) while rejecting signals outwith this range in order to reject such things as strip lights.

You may wish to brush up on your serial port theory for this - lots of info online, maybe enough on the Furby page. I make the assumption that most receivers will be happy with combs of pulses at 38.4kHz, since they are designed round R-C filters and diode pumps (at least conceptually, though most use one integrated device). I can achieve this by transmitting suitable characters at 115200 baud out of any serial port, so long as there are no gaps between characters. The precise pattern for a comb is the character value 0xdb, at 115k2/8/e/2 (0x5b at 115k2/7/n/1 would also be possible but would require data to be loaded into the uart more frequently). There is sufficient tolerance in most receivers to cope with the effective 4-pulse granularity (or 3-pulse if 7/n/1) in the comb. The next assumption I make is that out-of-frequency-band pulses will be rejected by the receiver, so if I transmit 0xfe, still at 115k2/8/e/2, this will look like silence to the receiver. So that's the transmission part covered - by sending correctly-sized groups of characters with values 0xdb and 0xfe, I can effectively construct an infrared stream which looks like it came from a 'real' remote control from the receiver's point of view, so long as I keep the uart continuously stuffed with data. The timing is derived from the uart, not the raw speed of the PC or other host device, and it is fairly easy to keep up.

Sampling works as follows. Using the very simple hardware described below, (or your own equivalent - doesn't need to be a handshake line, could be a joystick or printer input, for example), I transmit a junk character (0xff as it happens, but it doesn't matter) continuously out the serial port with no gaps between characters. While waiting for each character to be transmitted, I sample the IR input as frequently as possible (in my case a handshake line) and if I ever see an 'active' condition on it (ie if IR is ever detected) I assume that there is a comb (36 - 40kHz) of data being received from a remote control during this period, and if I don't see activity I assume this is a silent period. I grab a big array, with one bucket for each character time, and the buckets will either be marked as active or silent. After a preset number of characters have been sent, I stop grabbing and process the data. What I end up with is another array of buckets which contain alternating active counts and silent counts, stored as numbers of character times, adjusted to compensate for the inherent tendency of this method to round up comb lengths and round down silence lengths. Arrays like this provide a compact method of sample storage, and are very easy to play back when I want to transmit the sample as described above, and again, the timing is absolute, based on the uart, not the processor raw speed, so it is accurate and repeatable. By understanding the actual protocol used, it would be easy to compute the packets rather than store samples as such and make the storage space requirement much smaller, but I'm trying to avoid this to keep things simple and flexible. There are a few details to do with timeouts, sample sizes,

waiting for the data to start, suspending interrupts etc which will be explained elsewhere, but the basic principle is very simple and apparently rock solid.

It is not possible to sample remotes through the IRDA port - at least with my software, but probably not at all on a PC - due to the design of the port hardware. Samples have to be made using a 232 port using the hardware described below, or equivalent, but this should not be a problem. Playback through the IRDA port works fine so long as you are aware of the potential pitfalls (again covered to some extent on my Furby page, and in particular relating to operation under Windows).

I have run the same software on an 8MHz 286 (yes, I managed to find one that still works) and a K6-2 333 and got the same results. Samples made on both extremes of machine look the same and can be exchanged without problems. This suggests that the technique will work well on a wide range of devices with serial ports, not just PCs. The actual sample-grabbing process is designed to scream along, even on a slow machine, though the post-grab processing can take as much time as it needs since it is not real-time critical.

**Update 26th August, 2002**

Things move on. Since writing the above section, PCs have got considerably faster. I am happy to report that the system has now been tested unchanged on Athlon 1000 boxes and above, without problems, under Windows 95 / 98SE, and, from the feedback I've had, the project has been constructed and run on a variety of PCs, and implemented on and some other platforms, in various parts of the world.

Now the interesting news. I have figured out a way of retaining all the uart-derived timing properties of transmission described above while achieving 'real' silence during the silent periods rather than the combs of out-of-frequency-band pulses which have been used up to now. It works on all the PCs I can get my hands on. The latest version of the DOS program (version 1.3) implements this and no longer has the option to switch in software timing loops to achieve (with some calibration effort) the same effect. It takes advantage of some particular features of the PC uart hardware, so I wouldn't class it as a new method, rather as an optimisation aimed at PCs. Since the vast majority of users are PC users, this new version should be of general interest, though I stand by the original method as described above as the more general solution. I have to say that I personally don't have a problem using the original silence method with the receivers I'm controlling, but this reworked version may be of some help with borderline cases, and this is now the 'official version' - I've moved over to it anyway, because it has a couple of other features I wanted, which are described elsewhere in the docs.

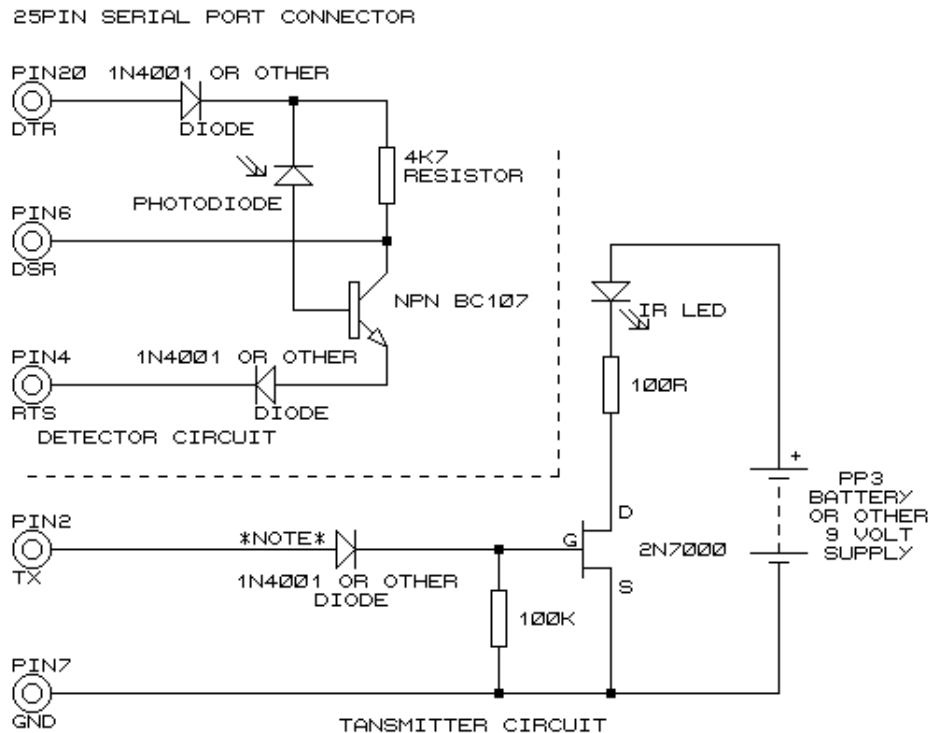
If you are a first-time user of this project, don't worry, just use the latest version of the software.

If you are an existing user of the system, there are several reasons why you might wish to install the latest version. Firstly, real silence rather than silence combs should work with a larger selection of equipment - the original silence combs might somewhat saturate the IR receiver or indeed totally confuse some poorly-filtered equipment (always one of the known drawbacks of the method), so the new method will hopefully improve the IR on/off contrast ratio and thus improve the operating range or make it work properly with your target equipment for the first time.

Secondly, the data stream will be much easier to observe on a proper oscilloscope than it used to be. Thirdly, you can do fun things like transmit a sample from one PC to another and get a close match (I know, you could do it more easily using a floppy or whatever). Fourthly, the new software has some helpful features in DOS-screen mode which make it easier to experiment, diagnose problems and manipulate data sets. Fifthly, the average power consumption of transmission is reduced, prolonging your battery life, saving the planet and so on. The main thing here is I can see no downside. All you need to do is replace the DOS program - the change is invisible to the Windows program, and any samples you already have remain totally compatible. At worst, you get the benefit of the new features; at best you may also get better performance.

## Hardware - Section updated 26th August, 2002

Here is a schematic for the hardware I use. This has been updated on 26th August, 2002 to include the diode explained below. If you built yours before this date, and you think your LED might be glowing slightly (in an IR sense) when it should be off, you should consider adding the diode even if everything seems to work fine. It may give you improved performance. You can check to see if you have the problem using a video camera or by measuring the current through the LED in the idle state - don't bother changing it if you don't see the problem. Having said that, it only hit me when I started playing with substitute transistors which were intended to be near-equivalents.



There are really two separate circuits, and it would be ok just to build the receiver section for sampling and use the IRDA port for transmission. The receiver section takes the power it needs from handshake lines (so no external source required in this configuration). The receiver range is



only a couple of cm, so the remote has to be close to work, but there is no particular reason why a proper circuit shouldn't be built which would operate over several metres, and indeed software could be written to let the PC decode the signal properly, though it would be a non-starter as a mouse substitute under Windows or whatever. There are additional notes about the driver transistor (2N7000) on my Furby page ([here](#)).

The diode marked \*NOTE\* between the TX pin and the gate of the FET has been added because I occasionally observed the LED to be slightly on when the TX line was low and the LED should have been off, with the result that the on/off IR contrast ratio was reduced (and consequently maybe the useful transmission range). There are a couple of possible explanations for the LED being on, which I didn't investigate further, opting rather to kill the problem once and for all by including the diode. Shame really, because the tolerance to large positive and negative Vgs values was one of the reasons I chose the FET in the first place. Anyway, the diode effectively prevents the gate from going more negative than the source and the problem is gone. If you suspect that the FET is not turning off fast enough with the 100k pulldown (due to extremely high gate / track capacitance or whatever) you could safely reduce the resistor to 10k to see if that helps, but I have no reason to think the change is needed.

The FAQ page has been updated, and now includes a couple of (untested) suggestions for increasing the output of the transmitter, and the simple change needed in order to replace the FET with a normal NPN transistor.

## **Software - section updated 26th August, 2002**

There are now two versions of the software package available for download, and unless you have a pressing desire to do comparisons between the original version (winsamp version 1.1) and the current one (V1.3), you should ignore REMOTE11.ZIP (37k) and go for REMOTE13.ZIP (39k). The packages contain versions of the same set of 5 files - MANYBUTT.EXE, BUTT1.BUT and BUTT18.BUT are unchanged, WINSAMP.EXE and README.TXT have been updated in the newer (remote13.zip) version. If you are updating your installation from 1.1 to 1.3, only the two changed files need to be replaced (obviously). Just for information, V1.2 was an intermediate version which was never published because I didn't want to have to do two page and docs revisions in quick succession, but you're not missing anything because 1.3 contains all of 1.2 plus extra features. At risk of repeating myself, the upgrade will not require you to do anything to your existing sample files, does not change the command line interface, does not modify the settings of the GUI (manybutt.exe), and only requires you to replace winsamp.exe (and the readme file, if you want to keep things in step - and then probably only if you actually want to read it). Note that the remainder of this section is virtually the same as what was here before, so there is no need to read it again if you're already familiar with the contents.

The software here is enough to get you going both in dos and Windows95/98, but is not pretty, clever or feature-laden. It goes some way beyond proving the point that the method works, but falls well short of some of the applications which have been written around the parallel-port system. It does everything I need it to do as far as controlling multiple devices and producing platform-independent samples is concerned, so I have drawn the line there for the time being, but I hope I will have provided enough information to allow anyone with an inclination towards

programming to write their own much nicer versions or to port the method to other platforms. I have in mind some particular features I might add to the VB side to address some specific future project requirements, but these are so far off the normal track that there would be little point in trying to build them in to a general app, and anyway they still depend on utilising the same dos-based core program.

### How to get started

Click here to download REMOTE13.ZIP (39k) , a .zip file which contains five files - WINSAMP.EXE, MANYBUTT.EXE, README.TXT, BUTT1.BUT and BUTT18.BUT. Create a directory (wherever you want and whatever you want to call it) and put these into it. WINSAMP.EXE is the main dos app, and is all you really need to get started. MANYBUTT.EXE is my VB5 demo for Win95/98 which calls the dos program as required. The \*.BUT files are example buttons used by MANYBUTT.EXE. README.TXT is all the explanation and detail I didn't want to put on this page, written in plain text, and I only suggest that it be in the same directory so it doesn't get lost or overwrite any other file of the same name - it overlaps with this file, but you would be expected to have both. This html file (REMOTE.HTM) and README.TXT are expected to evolve as I find problems, develop the idea further and hopefully if and as I get feedback. Create a shortcut to MANYBUTT.EXE on your desktop for convenience. You can also create one for WINSAMP.EXE if you prefer to access it this way rather than from dos prompt, but if you want to see what's going on, it would be best if you make sure (by tweaking properties if necessary) that it runs Full-Screen. Remember that if you want to run it on anything other than COM2, you'll need to edit the command line to WINSAMP.EXE Cx. You can actually look at and fiddle with the software without having any sample/playback hardware. If you already built the IR Thingy from the Furby page, you're half way there hardware-wise, and if you managed to get the Furby stuff working, there's a very good chance this will work too. What I suggest is you save this page or print it out, get the software and read the readme file, maybe have a look at the programs then decide whether or not you can be bothered to build the hardware. If you get that far, then go to dos prompt and run WINSAMP.EXE, using appropriate COM port selection and start trying to sample remotes. If that seems to function, then use MANYBUTT.EXE from the desktop - it's a lot more convenient in the long run. Note that only COM1 - COM4 are supported by WINSAMP.EXE, and that MANYBUTT.EXE does not touch the ports as such. MANYBUTT simply passes a string, which the dos app uses to determine the hardware addresses it hits directly.

### Some more detail

The main software (winsamp.exe, 21k) is a dos application which can be run in stand-alone mode or shelled from Windows (tested with 95 and 98). It is not Windows-aware, and consequently takes more processor time when idle than it really needs, but the normal way to use it under Windows is to shell it with various command line options set, which causes it to start, do what you want (eg grab a sample or play back a sample) then exit without any keyboard / mouse / screen activity, so it's normally only loaded briefly, and the program is so small that it takes a negligible amount of time to load. Note that once a sample starts to arrive or starts to be played back, interrupts are suspended until the operation has finished. In the case of sampling, the default sample window is approx 250ms, but this can be pushed up to 500ms using command line options. For playback, this time is the duration of the stored sample, which again can be controlled from

the command line but defaults to around 150ms and cannot exceed 500ms. Be aware that suspending interrupts in order to make sure there are no gaps in the data streams will inevitably result in system ticks being missed (I certainly hope so, this is one of the reasons for doing it), and the result is that your clock will gradually lose time, depending on how much you use the software. Sorry. An easy way round it would be to install one of these programs which syncs your PC to an atomic clock every time you go online if it bothers you - to some, the ability to slow down time might be seen as a bonus. A more messy way round it would be for me to attempt something clever with the real-time-clock hardware following each interrupt suspension, but it doesn't bother me that much to lose a few seconds now and again - when the battery was going flat on my PC I used to lose minutes per day and managed to cope, so this is nothing in comparison.

Because the interrupts are not suspended during sampling until the start of the IR stream from the remote is detected, there is a very small chance that a system tick or similar might happen between detection and the instruction which does the suspending. The chance of this happening is extremely small, but becomes more possible on slower machines, and the ISR would probably take longer to complete if it did happen. The effect of this would be that the recorded width of the first comb would be smaller than it actually was, which would perhaps make the sample invalid on playback for some protocols. I think I've seen this once in all the playing around during development, but of course it might be that pressing the button on the remote causes it to move into range of the receiver just at the critical time or whatever. I would suggest observing a few samples on the sillyscope screen to get a feel for what a good sample looks like, and have a quick look at the sample files to see if any are outstandingly different before burning them onto a million CD Roms or whatever. And of course test the samples.

The most usual way to operate the program in 'real' dos or dos prompt under Windows is to type 'WINSAMP' or WINSAMP Cx where x is the COM port you want to use if the default of COM2 does not suit. A 80\*50 text-mode screen is presented (which might not be compatible with \_very\_ old graphics cards, but if your machine can run Windows it should be ok), and this has lots of things squished onto it. You need to read all the bits of writing on it because the user controls are all hinted at somewhere there. For details of command line options, type WINSAMP ? and a horrible text screen will appear, which might help. A separate README.TXT file is included explaining the command line options and controls in more detail, along with examples of how to make it work from Windows programs. I have also included a simple Windows front-end demonstration written in VB5 which uses the dos app for all the low-level stuff. This is only the actual .exe file - due to bandwidth restrictions I can't put up a full distribution - the .exe is only a few k, but all the support bits take up megs of space, so unless you already have VB5 or have acquired the support files through other software installations you've done over the years, I'm afraid you're stuck, but there is a somewhat less satisfactory but quite functional method for accessing the dos program under Windows using nothing more than program groups and shortcuts which is explained in README.TXT.

It may be possible to rewrite the whole thing as a 'proper' Windows application, but using the dos program makes it relatively simple to achieve the necessary low-level hardware access and timing. Certainly, VB or similar could be used to provide sophisticated sample management (archiving, selection, grouping, exporting, importing etc), but I have only done enough to suit my own needs for the time being.

I hope you like it.

---

I have added a FAQ page. I welcome feedback on the project, but if you have specific questions or requests, please do me a favour and make sure they haven't already been covered by this web page, the docs or the FAQ. Your question and / or my response might be added in a generic and unattributed sort of way to the next revision of the FAQ.



View my other offerings

Copyright? Possibly .....

59661

# RLE - Run Length Encoding

Written by Paul Bourke  
August 1995

## Source code

Standard compression C source: rle.c  
Example code based upon the above:  
rletest.c

---

## Introduction

Run length encoding is a straightforward way of encoding data so that it takes up less space. It relies on the string being encoded containing runs of the same character. Consider storing the following short string.

```
abcdccccccbbbbbabcdef
```

There are 20 letters above, if each is stored as a single byte that is 20 bytes in all. However, the runs of "d" and "b" above can be stored as two bytes each, the first indicating how many letters in the run. For example, the following run length encoded string takes only 14 bytes.

```
abc6dc4babcdef
```

In general of course it needs to be a bit more sophisticated than the above. For example, there is no way in the above encoding to encode strings with numbers, that is, how would one know whether the number was the length of the run or part of the string content. Also, one would not want to encode runs of length 1 so how does one tell when a run starts and when a literal sequence starts.

The common approach is to use only 7 of the 8 bits to indicate the run length, this is normally interpreted as a signed byte. If the length byte is positive it indicates a replicated run (run of the following byte). If the number is negative then it indicates a literal run, that is, that number of following bytes is copied as is. To illustrate this the following sequence of bytes would encode the example string given above, it requires 17 bytes.

```
-3 a b c 6 d -1 c 4 b 6 a b c d e f
```

While 17 bytes to encode what would take 20 without RLE may not sound like much, but as the frequency and length of the repeating characters increases the compression ratio gets better.

## Worst case

Of course RLE will not always result in a compression, consider a string where the next character is different from the current character. Every 127 bytes will require an extra byte to indicate a new literal run length.

## Best case

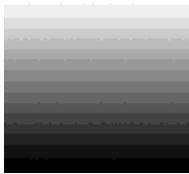
The best case is when 128 identical characters follow each other, this is compressed into 2 bytes instead

of 128 giving a compression ratio of 64.

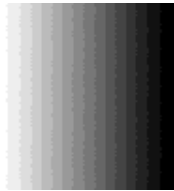
### Example

For this reason RLE is most often used to compress black and white or 8 bit indexed colour images where long runs are likely. RLE compression is therefore what was used for the original low colour images expected for the Macintosh PICT file format. RLE is not generally used for high colour images such as photographs where in general each pixel will vary from the last.

The following 3 images illustrate the different extremes, the first image contains runs along each row and will compress well. The second image is the same as the first but rotated 90 degrees so there are no runs giving worse case and a larger file. This suggests a natural extension to RLE for images, that is, one compresses vertically and horizontally and uses the best, the flag indicating which one is used is stored in the image header. The last case is the best scenario where the whole image is a constant value.



Original size: 10000 bytes  
Compressed size: 5713 bytes  
Ratio: 1.75



Original size: 10000 bytes  
Compressed size: 10100  
Ratio: 0.99



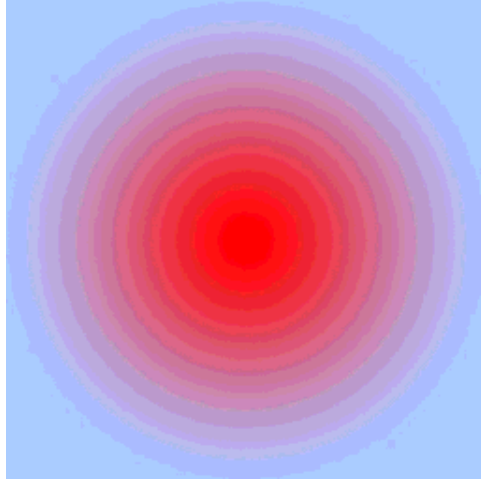
Original size: 10000 bytes  
Compressed size: 200  
Ratio: 50

### Image comparison

Run length encoding is used within a number of image formats, for example PNG, TIFF, and TGA. While RLE is normally used as a lossless compression, it can be assisted (to create small files) by quantising the rgb values thus increasing the chances of runs of the same colour. There are two ways one can run length encode the pixels, the first as used in the TGA format is to look for runs of all three components, the other is to compress each colour plane separately. The second approach normally gives smaller files. Below is a table for two different images along with the file size for the image quantised to different levels and saved in rgb order or planar order.

Image details and quantisation level	Image example	RLE on RGB (KBytes)	RLE on Planes (KBytes)
---	---------------	------------------------	---------------------------

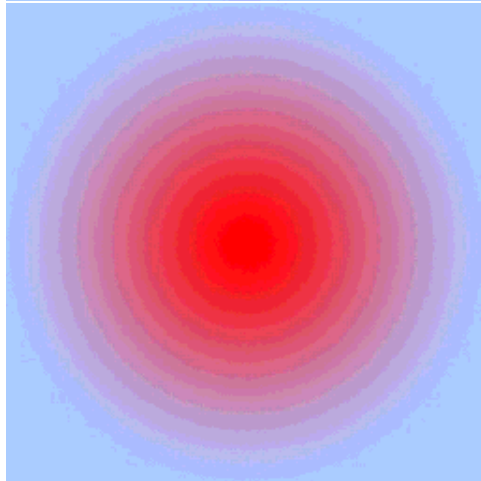
Uncompressed



197

197

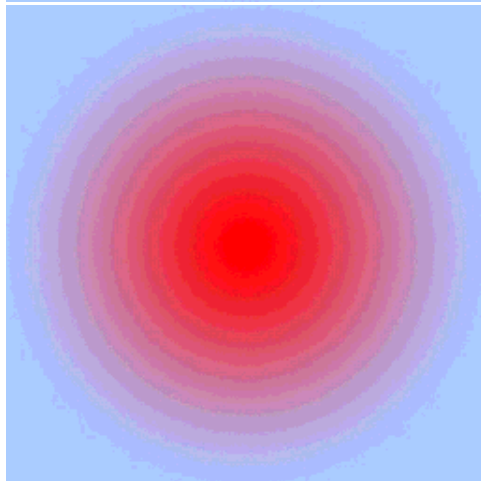
1 (none)



151

141

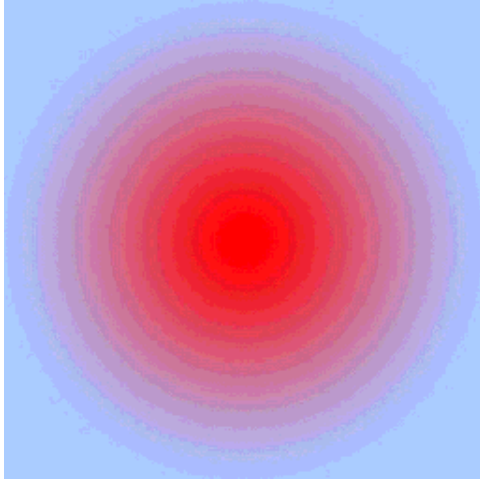
2



135

110

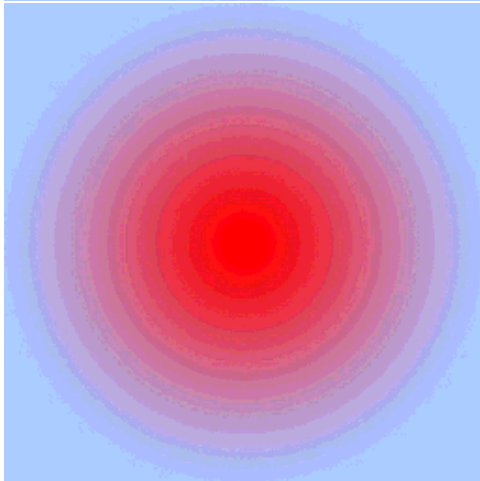
4



101

70

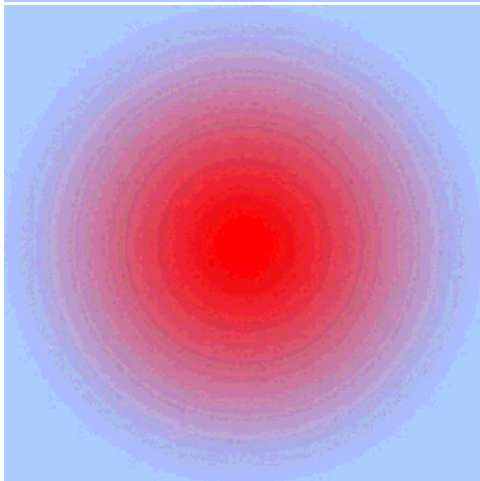
6



81

49

8

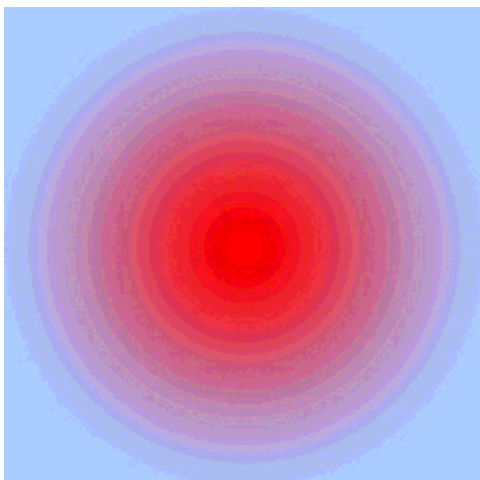


64

36



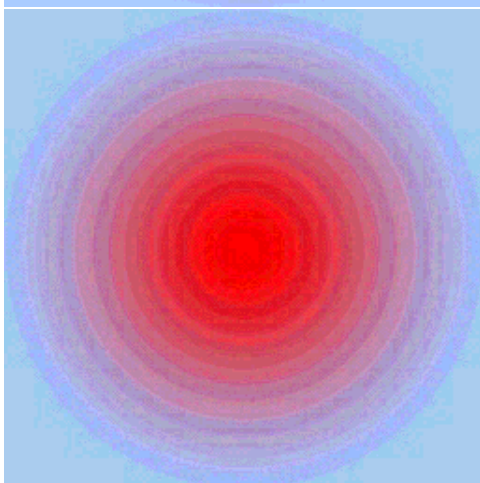
12



46

24.5

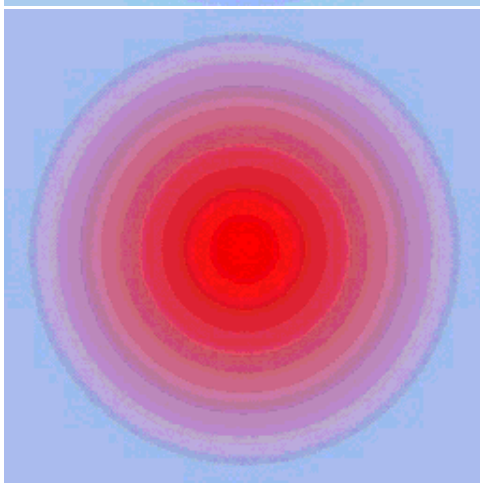
16



35

18.5

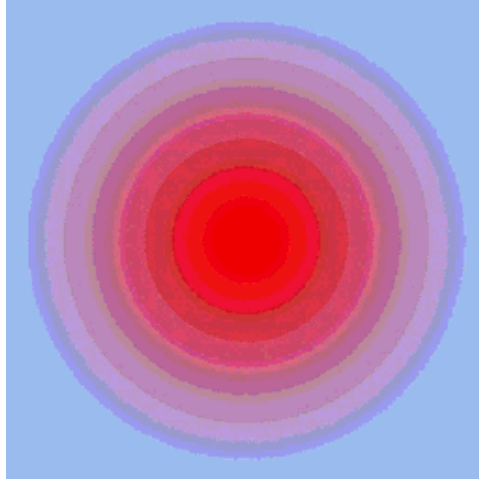
20



28

14.5

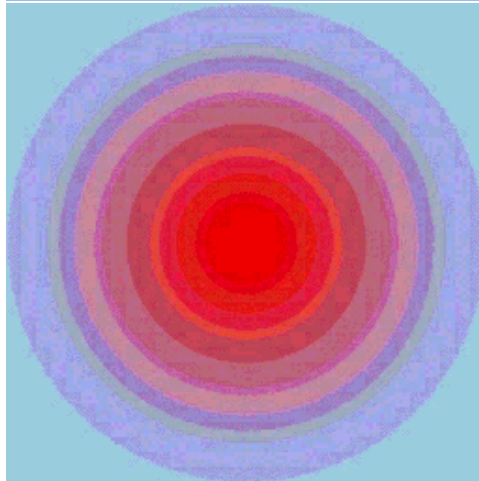
26



22

11.5

32



17.5

9.5

The above example was chosen because it doesn't have long runs of equal colour and because any banding due to quantisation should be obvious to spot. This occurs somewhere between 4 and 8 depending on how fussy one is. Note the planar compression works much better than the rgb based compression.

Image details and  
quantisation level

Image example

RLE on RGB  
(KBytes)

RLE on Planes  
(KBytes)

Uncompressed



197

197

1 (none)



195

190

2



188

173

4



163

140

6



142

119

8



127

106

12



105

88

16



86.5

74

20



92

73.5

26



74

60.5

32



60

50.5

Unlike the first example, because each of the colour layers in this images are "busy" the difference between rgb and planar RLE compression is not so marked. Note that the visual artifacts that occur to so on the wall where there is a smooth and subtle shade variation, even at the highest quantisation level the artifacts on the vase are hard to pick.

# RS232 Data Interface

a Tutorial on Data Interface and cables

RS-232 is simple, universal, well understood and supported but it has some serious shortcomings as a data interface. The standards to 256kbps or less and line lengths of 15M (50 ft) or less but today we see high speed ports on our home PC running very high speeds and with high quality cable maxim distance has increased greatly. The rule of thumb for the length a data cable depends on speed of the data, quality of the cable.

---

## a Tutorial

Electronic data communications between elements will generally fall into two broad categories: single-ended and differential. RS232 (single-ended) was introduced in 1962, and despite rumors for its early demise, has remained widely used through the industry.

Independent channels are established for two-way (full-duplex) communications. The RS232 signals are represented by voltage levels with respect to a system common (power / logic ground). The "idle" state (MARK) has the signal level negative with respect to common, and the "active" state (SPACE) has the signal level positive with respect to common. RS232 has numerous handshaking lines (primarily used with modems), and also specifies a communications protocol.

The RS-232 interface presupposes a common ground between the DTE and DCE. This is a reasonable assumption when a short cable connects the DTE to the DCE, but with longer lines and connections between devices that may be on different electrical busses with different grounds, this may not be true.

RS232 data is bi-polar.... +3 TO +12 volts indicates an "ON or 0-state (SPACE) condition" while A -3 to -12 volts indicates an "OFF" 1-state (MARK) condition.... Modern computer equipment ignores the negative level and accepts a zero voltage level as the "OFF" state. In fact, the "ON" state may be achieved with lesser positive potential. This means circuits powered by 5 VDC are capable of driving RS232 circuits directly, however, the overall range that the RS232 signal may be transmitted/received may be dramatically reduced.

The output signal level usually swings between +12V and -12V. The "dead area" between +3v and -3v is designed to absorb line noise. In the various RS-232-like definitions this dead area may vary. For instance, the definition for V.10 has a dead area from +0.3v to -0.3v. Many receivers designed for RS-232 are sensitive to differentials of 1v or less.

This can cause problems when using pin powered widgets - line drivers, converters, modems etc.

These type of units need enough voltage & current to power them self's up. Typical URART (the RS-232 I/O chip) allows up to 50ma per output pin - so if the device needs 70ma to run we would need to use at least 2 pins for power. Some devices are very efficient and only require one pin (some times the Transmit or DTR pin) to be high - in the "SPACE" state while idle.

An RS-232 port can supply only limited power to another device. The number of output lines, the type of interface driver IC, and the state of the output lines are important considerations.

The types of driver ICs used in serial ports can be divided into three general categories:

- Drivers which require plus (+) and minus (-) voltage power supplies such as the 1488 series of interface integrated circuits. (Most desktop and tower PCs use this type of driver.)
- Low power drivers which require one +5 volt power supply. This type of driver has an internal charge pump for voltage conversion. (Many industrial microprocessor controls use this type of driver.)
- Low voltage (3.3 v) and low power drivers which meet the EIA-562 Standard. (Used on notebooks and laptops.)

---

Data is transmitted and received on pins 2 and 3 respectively. Data Set Ready (DSR) is an indication from the Data Set (i.e., the modem or DSU/CSU) that it is on. Similarly, DTR indicates to the Data Set that the DTE is on. Data Carrier Detect (DCD) indicates that a good carrier is being received from the remote modem.

Pins 4 RTS (Request To Send - from the transmitting computer) and 5 CTS (Clear To Send - from the Data set) are used to control. In most Asynchronous situations, RTS and CTS are constantly on throughout the communication session. However where the DTE is connected to a multipoint line, RTS is used to turn carrier on the modem on and off. On a multipoint line, it's imperative that only one station is transmitting at a time (because they share the return phone pair). When a station wants to transmit, it raises RTS. The modem turns on carrier, typically waits a few milliseconds for carrier to stabilize, and then raises CTS. The DTE transmits when it sees CTS up. When the station has finished its transmission, it drops RTS and the modem drops CTS and carrier together.

Clock signals (pins 15, 17, & 24) are only used for synchronous communications. The modem or DSU extracts the clock from the data stream and provides a steady clock signal to the DTE. Note that the transmit and receive clock signals do not have to be the same, or even at the same baud rate.

**Note:** Transmit and receive leads (2 or 3) can be reversed depending on the use of the equipment - DCE Data Communications Equipment or a DTE Data Terminal Equipment.

---

**Glossary of Abbreviations etc.**



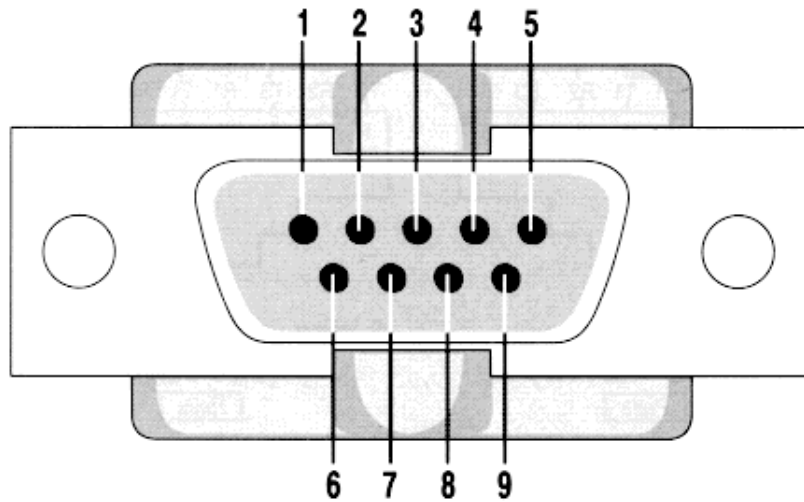
<b>CTS</b>	<b>Clear To Send [DCE --&gt; DTE]</b>
<b>DCD</b>	<b>Data Carrier Detected (Tone from a modem) [DCE --&gt; DTE]</b>
<b>DCE</b>	<b>Data Communications Equipment eg. modem</b>
<b>DSR</b>	<b>Data Set Ready [DCE --&gt; DTE]</b>
<b>DSRS</b>	<b>Data Signal Rate Selector [DCE --&gt; DTE] (Not commonly used)</b>
<b>DTE</b>	<b>Data Terminal Equipment eg. computer, printer</b>
<b>DTR</b>	<b>Data Terminal Ready [DTE --&gt; DCE]</b>
<b>FG</b>	<b>Frame Ground (screen or chassis)</b>
<b>NC</b>	<b>No Connection</b>
<b>RCK</b>	<b>Receiver (external) Clock input</b>
<b>RI</b>	<b>Ring Indicator (ringing tone detected)</b>
<b>RTS</b>	<b>Ready To Send [DTE --&gt; DCE]</b>
<b>RxD</b>	<b>Received Data [DCE --&gt; DTE]</b>
<b>SG</b>	<b>Signal Ground</b>
<b>SCTS</b>	<b>Secondary Clear To Send [DCE --&gt; DTE]</b>
<b>SDCD</b>	<b>Secondary Data Carrier Detected (Tone from a modem) [DCE --&gt; DTE]</b>
<b>SRTS</b>	<b>Secondary Ready To Send [DTE --&gt; DCE]</b>
<b>SRxD</b>	<b>Secondary Received Data [DCE --&gt; DTE]</b>
<b>STxD</b>	<b>Secondary Transmitted Data [DTE --&gt; DTE]</b>
<b>TxD</b>	<b>Transmitted Data [DTE --&gt; DTE]</b>

### Is Your Interface a DTE or a DCE?

**Find out by following these steps: The point of reference for all signals is the terminal (or PC).**

- 1 ) Measure the DC voltages between (DB25) pins 2 & 7 and between pins 3 & 7. Be sure the black lead is connected to pin 7 (Signal Ground) and the red lead to whichever pin you are measuring.**
- 2) If the voltage on pin 2 (TD) is more negative than -3 Volts, then it is a DTE, otherwise it should be near zero volts.**
- 3) If the voltage on pin 3 (RD) is more negative than -3 Volts, then it is a DCE.**
- 4) If both pins 2 & 3 have a voltage of at least 3 volts, then either you are measuring incorrectly, or your device is not a standard EIA-232 device. Call technical support.**
- 5) In general, a DTE provides a voltage on TD, RTS, & DTR, whereas a DCE provides voltage on RD, CTS, DSR, & CD.**

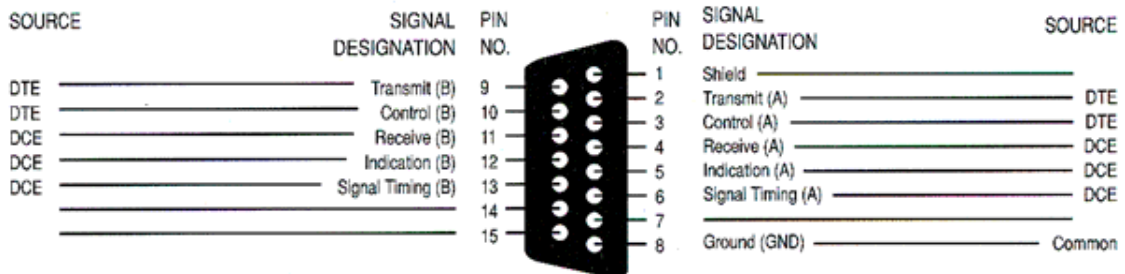
used for Asynchronous Data



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

### X.21 interface on a DB 15 connector

#### X.21 Interface



also see X.21 write up  
also see end of page for more info

## X.21

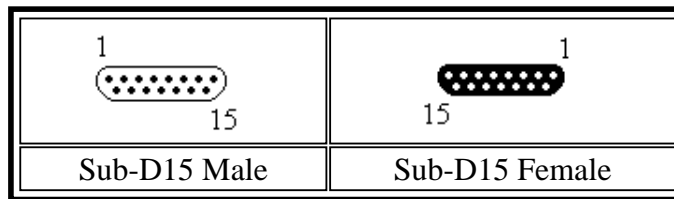
### General

Voltages:	+/- 0.3Vdc
Speeds:	Max. 100Kbps (X.26)
	Max. 10Mbps (X.27)

The X.21 interface was recommended by the CCITT in 1976. It is defined as a digital signalling interface between customers (DTE) equipment and carrier's equipment (DCE). And thus primarily used for telecom equipment.

All signals are balanced. Meaning there is always a pair (+/-) for each signal, like used in RS422. The X.21 signals are the same as RS422, so please refer to RS422 for the exact details.

Pinning according to ISO 4903



Pin	Signal	abbr.	DTE	DCE
1	Shield		-	-
2	Transmit (A)		Out	In
3	Control (A)		Out	In
4	Receive (A)		In	Out
5	Indication (A)		In	Out
6	Signal Timing (A)		In	Out
7	Unassigned			
8	Ground		-	-
9	Transmit (B)		Out	In
10	Control (B)		Out	In
11	Receive (B)		In	Out
12	Indication (B)		In	Out
13	Signal Timing (B)		In	Out
14	Unassigned			
15	Unassigned			

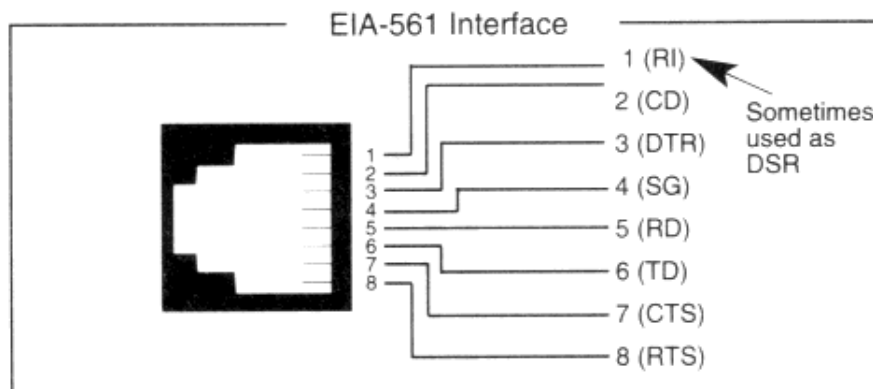
#### Functional Description

As can be seen from the pinning specifications, the Signal Element Timing (clock) is provided by the DCE. This means that your provider (local telco office) is responsible for the correct clocking and that X.21 is a synchronous interface. Hardware handshaking is done by the Control and Indication lines. The Control is used by the DTE and the Indication is the DCE one.

#### Cross-cable pinning

X.21 Cross Cable	
X.21	X.21
1	1
2	4
3	5
4	2
5	3
6	7
7	6
8	8
9	11
10	12
11	9
12	10
13	14
14	13
15	

**EIA-561** defines RS-232 on a modular connector. (For nonsynchronous applications only, since it does not provide for the synchronous clocking signals.)



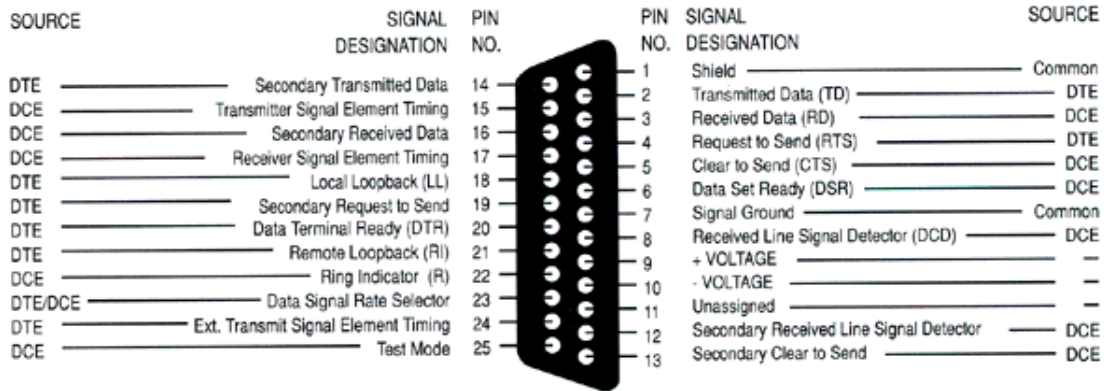
**RS232D** uses **RJ45** type connectors  
(similar to telephone connectors)

Pin No.	Signal Description	Abbr.	Direction	
			DTE	DCE
1	DCE Ready, Ring Indicator	DSR/RI	←	→
2	Received Line Signal Detector	DCD	←	→
3	DTE Ready	DTR	→	←
4	Signal Ground	SG		
5	Received Data	RxD	←	→
6	Transmitted Data	TxD	→	←
7	Clear To Send	CTS	←	→
8	Request To Send	RTS	→	←

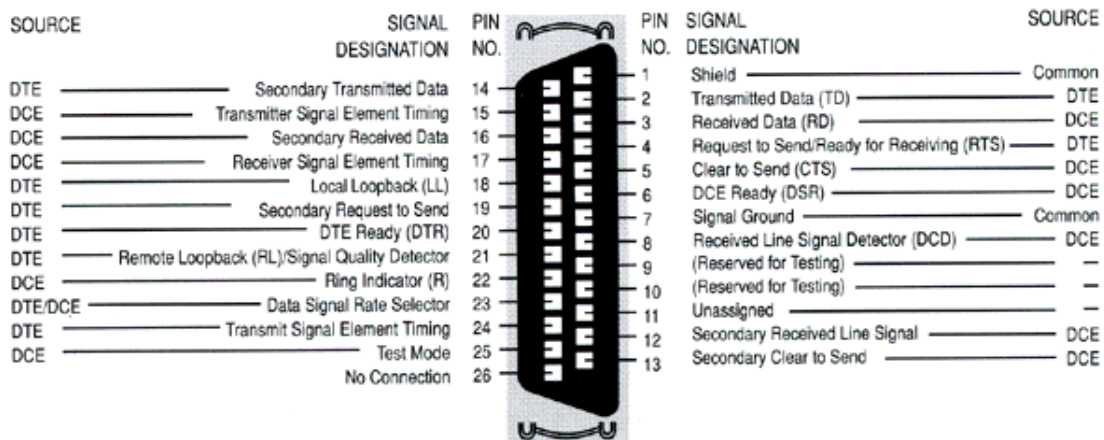
This is a standard 9 to 25 pin cable layout for async data on a PC AT serial cable

Description	Signal	9-pin DTE	25-pin DCE	Source DTE or DCE
Carrier Detect	CD	1	8	from Modem
Receive Data	RD	2	3	from Modem
Transmit Data	TD	3	2	from Terminal/Computer
Data Terminal Ready	DTR	4	20	from Terminal/Computer
Signal Ground	SG	5	7	from Modem
Data Set Ready	DSR	6	6	from Modem
Request to Send	RTS	7	4	from Terminal/Computer
Clear to Send	CTS	8	5	from Modem
Ring Indicator	RI	9	22	from Modem

## V.24/RS-232 Interface



## V.24/RS-232E ALT A Connector



## 25 pin D-shell connector RS232

commonly used for Async. data

PIN SIGNAL DESCRIPTION

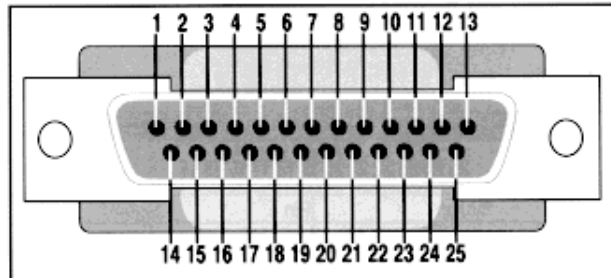
- 1 PGND Protective Ground
- 2 TXD Transmit Data
- 3 RXD Receive Data
- 4 RTS Ready To Send
- 5 CTS Clear To Send

- 6 DSR Data Set Ready
- 7 SG Signal Ground
- 8 CD Carrier Detect
- 20 DTR Data Terminal Ready
- 22 RI Ring Indicator

Some applications require more than a simple async. configurat

### RS-232 Interface

RS-232 (EIA Std.) applicable to the 25 pin interconnection of Data Terminal Equipment (DTE) and Data Communications Equipment (DCE) using serial binary data



Pin	Description	EIA CKT	From DCE	To DCE
1	Frame Ground	AA		
2	Transmitted Data	BA		D (Data)
3	Received Data	BB	D	
4	Request to Send	CA		C (Control)
5	Clear to Send	CB	C	
6	Data Set Ready	CC	C	
7	Signal Gnd/Common Return	AB		
8	Rcvd. Line Signal Detector	CF	C	
11	Undefined			
12	Secondary Rcvd. Line Sig. Detector	SCF	C	
13	Secondary Clear to Send	SCB	C	
14	Secondary Transmitted Data	SBA		D
15	Transmitter Sig. Element Timing	DB	T (Timing)	
16	Secondary Received Data	SBB	D	
17	Receiver Sig. Element Timing	DD	T	
18	Undefined			
19	Secondary Request to Send	SCA		C
20	Data Terminal Ready	CD		C
21	Sig. Quality Detector	CG		C
22	Ring Indicator	CE	C	
23	Data Sig. Rate Selector (DCE)	CI	C	
23	Data Sig. Rate Selector (DTE)	CH		C
24	Transmitter Sig. Element Timing	DA		T
25	Undefined			

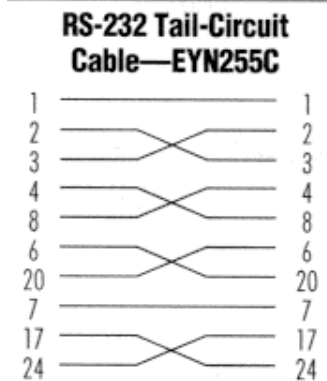


## Pins used for Synchronous data

jump to [Other Connector](#) pages

---

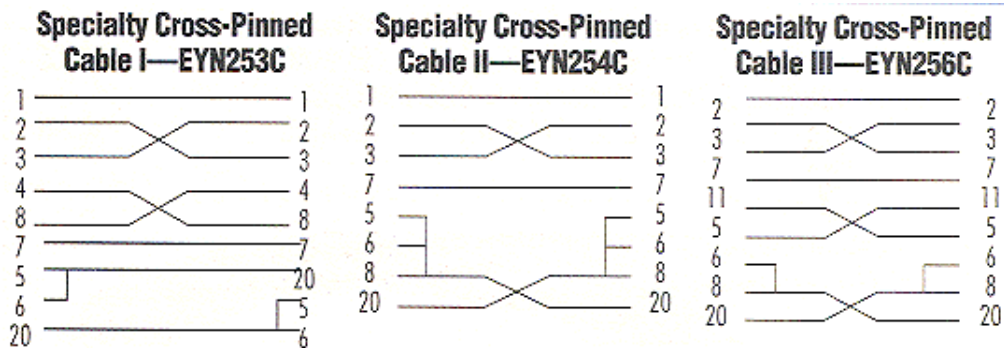
### RS232 (25 pin) Tail Circuit Cable



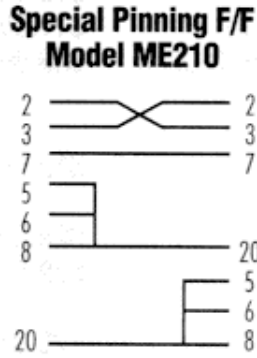
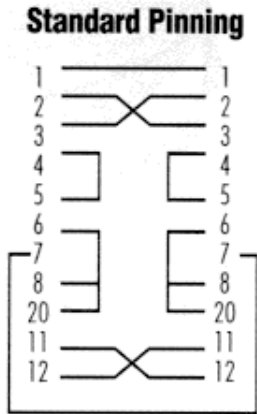
Null Modem cable diagrams

- Nullmodem (9p to 9p)
- Nullmodem (9p to 25p)
- Nullmodem (25p to 25p)

### Cross Pinned cables for Async data.



## Pin out for local Async Data transfer



### Loopback plugs:

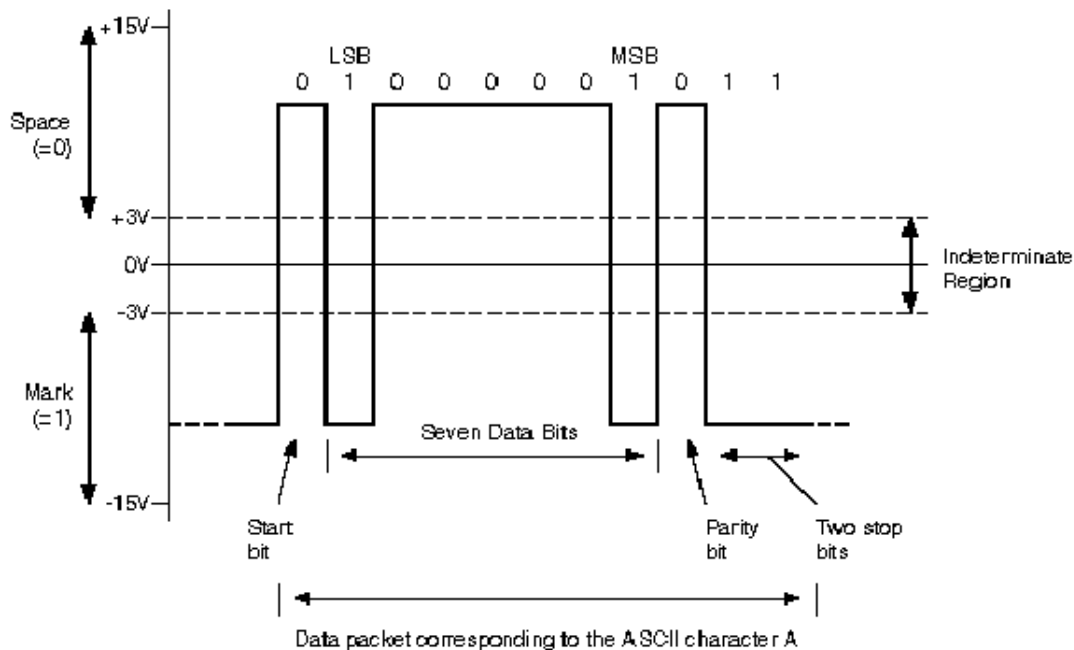
- Serial Port Loopback (9p)
- Serial Port Loopback (25p)

### RS-232 Specs .

SPECIFICATIONS		RS232	RS423
Mode of Operation		SINGLE -ENDED	SINGLE -ENDED
Total Number of Drivers and Receivers on One Line		1 DRIVER 1 RECVR	1 DRIVER 10 RECVR
Maximum Cable Length		50 FT.	4000 FT.
Maximum Data Rate		20kb/s	100kb/s
Maximum Driver Output Voltage		+/-25V	+/-6V
Driver Output Signal Level (Loaded Min.)	Loaded	+/-5V to +/-15V	+/-3.6V
Driver Output Signal Level (Unloaded Max)	Unloaded	+/-25V	+/-6V
Driver Load Impedance (Ohms)		3k to 7k	>=450
Max. Driver Current in High Z State	Power On	N/A	N/A
Max. Driver Current in High Z State	Power Off	+/-6mA @ +/-2v	+/-100uA
Slew Rate (Max.)		30V/uS	Adjustable
Receiver Input Voltage Range		+/-15V	+/-12V
Receiver Input Sensitivity		+/-3V	+/-200mV
Receiver Input Resistance (Ohms)		3k to 7k	4k min.

---

## One byte of async data



---

**Cabling considerations - you should use cabling made for RS-232 data but I have seen low speed data go over 250' on 2 pair phone cable. Level 5 cable can also be used but for best distance use a low capacitance data grade cable.**

**The standard maxim length is 50' but if data is async you can increase that distance to as much as 500' with a good grade of cable.**

**The RS-232 signal on a single cable is impossible to screen effectively for noise. By screening the entire cable we can reduce the influence of outside noise, but internally generated noise remains a problem. As the baud rate and line length increase, the effect of capacitance between the different lines introduces serious crosstalk (this especially true on synchronous data - because of the clock lines) until a point is reached where the data itself is unreadable. Signal Crosstalk can be reduced by using low capacitance cable and shielding each pair**

**Using a high grade cable (individually shield low capacitance pairs) the distance can be extended to 4000'**

At higher frequencies a new problem comes to light. The high frequency component of the data signal is lost as the cable gets longer resulting in a rounded, rather than square wave signal.

The maximum distance will depend on the speed and noise level around the cable run.

On longer runs a line driver is needed. This is a simple modem used to increase the maximum distance you can run RS-232 data.

## **Making sense of the specifications**

Selecting data cable isn't difficult, but often gets lost in the shuffle of larger system issues. Care should be taken, however, because intermittent problems caused by marginal cable can be very difficult to troubleshoot.

Beyond the obvious traits such as number of conductors and wire gauge, cable specifications include a handful of less intuitive terms.

**Characteristic Impedance (Ohms):** A value based on the inherent conductance, resistance, capacitance and inductance of a cable that represents the impedance of an infinitely long cable. When the cable is cut to any length and terminated with this Characteristic Impedance, measurements of the cable will be identical to values obtained from the infinite length cable. That is to say that the termination of the cable with this impedance gives the cable the appearance of being infinite length, allowing no reflections of the transmitted signal. If termination is required in a system, the termination impedance value should match the Characteristic Impedance of the cable.

**Shunt Capacitance (pF/ft):** The amount of equivalent capacitive load of the cable, typically listed in a per foot basis. One of the factors limiting total cable length is the capacitive load. Systems with long lengths benefit from using low capacitance cable.

**Propagation velocity (% of c):** The speed at which an electrical signal travels in the cable. The value given typically must be multiplied by the speed of light (c) to obtain units of meters per second. For example, a cable that lists a propagation velocity of 78% gives a velocity of  $0.78 \times 300 \times 10^6 = 234 \times 10^6$  meters per second.

## **Plenum cable**

Plenum rated cable is fire resistant and less toxic when burning than non-plenum rated cable. Check building and fire codes for requirements. Plenum cable is generally more expensive due to the sheathing material used.

The specification recommends 24AWG twisted pair cable with a shunt capacitance of 16 pF per foot and 100 ohm characteristic impedance.

It can be difficult to qualify whether shielding is required in a particular system or not, until problems arise. We recommend erring on the safe side and using shielded cable. Shielded cable is

only slightly more expensive than unshielded.

There are many cables available meeting the recommendations of RS-422 and RS-485, made specifically for that application. Another choice is the same cable commonly used in the Twisted pair Ethernet cabling. This cable, commonly referred to as Category 5 cable, is defined by the EIA/TIA/ANSI 568 specification. The extremely high volume of Category 5 cable used makes it widely available and very inexpensive, often less than half the price of specialty RS422/485 cabling. The cable has a maximum capacitance of 17 pF/ft (14.5 pF typical) and characteristic impedance of 100 ohms.

Category 5 cable is available as shielded twisted pair (STP) as well as unshielded twisted pair (UTP) and generally exceeds the recommendations making it an excellent choice for RS232 systems.

---

## **RS232 - V.24/V.28 - IS2110 - X.20 bis (for Async)**

-

### **X.21 bis (for Sync)**

#### General

In this document the term RS232 will be used when referred to this serial interface. The description of RS232 is an EIA/TIA norm and is identical to CCITT V.24/V.28, X.20bis/X.21bis and ISO IS2110. The only difference is that CCITT has split the interface into its electrical description (V.28) and a mechanical part (V.24) or Asynchronous (X.20 bis) and Synchronous (X.21 bis) where the EIA/TIA describes everything under RS232.

As said before RS232 is a serial interface. It can be found in many different applications where the most common ones are modems and Personal Computers. All pinning specifications are written for the DTE side.

All DTE-DCE cables are straight through meaning the pins are connected one on one. DTE-DTE and DCE-DCE cables are cross cables. To make a distinction between all different types of cables we have to use a naming convention.

DTE - DCE: Straight Cable

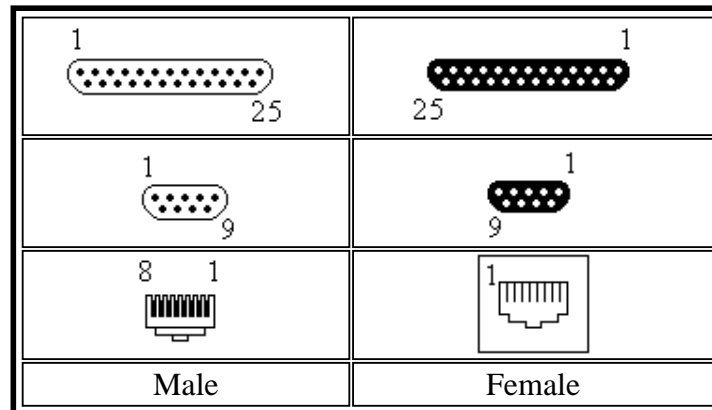
DTE - DTE: Null-Modem Cable

DCE - DCE: Tail Circuit Cable

#### Interface Mechanical

RS232 can be found on different connectors. There are special specifications for this. The CCITT only defines a Sub-D 25 pins version where the EIA/TIA has two versions RS232C and RS232D which are resp. on a Sub-D25 and a RJ45. Next to this IBM has added a Sub-D 9 version which is found almost

all Personal Computers and is described in TIA 457.



Pinning

RS232-C	Description	Circuit EIA	Circuit CCITT	RJ45	TIA 457
1	Shield Ground	AA			
7	Signal Ground	AB	102	4	5
2	Transmitted Data	BA	103	6	3
3	Received Data	BB	104	5	2
4	Request To Send	CA	105	8	7
5	Clear To Send	CB	106	7	8
6	DCE Ready	CC	107	1	6
20	DTE Ready	CD	108.2	3	4
22	Ring Indicator	CE	125	1	9
8	Received Line Signal Detector	CF	109	2	1
23	Data Signal Rate Select (DTE/DCE Source>	CH/CI	111/112		
24	Transmit Signal Element Timing (DTE Source)	DA	113		
15	Transmitter Signal Element Timing (DCE Source)	DB	114		
17	Receiver Signal Element Timing (DCE Source)	DD	115		
18	Local Loopback / Quality Detector	LL	141		
21	Remote Loopback	RL/CG	140/110		
14	Secondary Transmitted Data	SBA	118		
16	Secondary Received Data	SBB	119		
19	Secondary Request To Send	SCA	120		
13	Secondary Clear To Send	SCB	121		
12	Secondary Received Line Signal Detector/ Data signal Rate Select (DCE Source)	SCF/CI	122/112		
25	Test Mode	TM	142		
9	Reserved for Testing				
10	Reserved for Testing				
11	Unassigned				

## Interface Electrical

All signals are measured in reference to a common ground, which is called the signal ground (AB). A positive voltage between 3 and 15 Vdc represents a logical 0 and a negative voltage between 3 and 15 Vdc represents a logical 1.

This switching between positive and negative is called bipolar. The zero state is not defined in RS232

and is considered a fault condition (this happens when a device is turned off). According to the above a maximum distance of 50 ft or 15 m. can be reached at a maximum speed of 20k bps. This is according to the official specifications, the distance can be exceeded with the use of Line Drivers.

#### Functional description

Description	Circuit	Function
Shield Ground	AA	Also known as protective ground. This is the chassis ground connection between DTE and DCE.
Signal Ground	AB	The reference ground between a DTE and a DCE. Has the value 0 Vdc.
Transmitted Data	BA	Data send by the DTE.
Received Data	BB	Data received by the DTE.
Request To Send	CA	Originated by the DTE to initiate transmission by the DCE.
Clear To Send	CB	Send by the DCE as a reply on the RTS after a delay in ms, which gives the DCEs enough time to energize their circuits and synchronize on basic modulation patterns.
DCE Ready	CC	Known as DSR. Originated by the DCE indicating that it is basically operating (power on, and in functional mode).
DTE Ready	CD	Known as DTR. Originated by the DTE to instruct the DCE to setup a connection. Actually it means that the DTE is up and running and ready to communicate.
Ring Indicator	CE	A signal from the DCE to the DTE that there is an incoming call (telephone is ringing). Only used on switched circuit connections.
Received Line Signal Detector	CF	Known as DCD. A signal send from DCE to its DTE to indicate that it has received a basic carrier signal from a (remote) DCE.
Data Signal Rate Select (DTE/DCE Source)	CH/CI	A control signal that can be used to change the transmission speed.
Transmit Signal Element Timing (DTE Source)	DA	Timing signals used by the DTE for transmission, where the clock is originated by the DTE and the DCE is the slave.
Transmitter Signal Element Timing (DCE Source)	DB	Timing signals used by the DTE for transmission.
Receiver Signal Element Timing (DCE Source)	DD	Timing signals used by the DTE when receiving data.
Local Loopback / Quality Detector	LL	
Remote Loopback	RL/CG	Originated by the DCE that changes state when the analog signal received from the (remote) DCE becomes marginal.



Test Mode	TM	
Reserved for Testing		

The secondary signals are used on some DCE's. Those units have the possibility to transmit and/or receive on a secondary channel. Those secondary channels are mostly of a lower speed than the normal ones and are mainly used for administrative functions.

### Cable pinning

Here are some cable pinning that might be useful. Not all applications are covered, it is just a help:

#### Straight DB25 Cable

Pin	Pin
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
20	20
21	21
22	22
23	23
24	24
25	25

#### DB25 Null- modem cable (Async)

Pin	Pin
1	1
2	3
3	2
4	5
5	4
6, 8	20
7	7
20	6, 8

</

#### DB9 Null- modem cable

1,6	4
2	3
3	2
4	1,6
5	5
7	8
8	7

#### DB25 Tail- circuit cable (Sync)

Pin	Pin
1	1
2	3
3	2
4	8
6	20
7	7
8	4
17	24
20	6
24	17

#### DB25 to DB9 DTE - DCE cable

jump to **related fiber cable pages**

---

jump to **The Belden Cable Company's cable selection tutorial** pages

jump to **Data Communication by CAMI Research** good write up

jump to **RS-232 by CAMI Research** good write up

jump to **Interfacing the Serial / RS232 Port** good write up  
(in-depth very technical)

jump to **Data Modems for phone lines**

jump to **Data Modems for fiber optics**

jump to **Interface converters**

---

**ARC Electronics ...**

**800-926-0226**

Home Page

arc@arcelect.com

# PC serial port buffer

## Summary of circuit features

- Brief description of operation: Buffer to run RS-232 data to longer distanced as normally
- Circuit protection: No special protection circuits used
- Circuit complexity: Very simple two transistor buffer circuit
- Circuit performance: Worked nicely in one special application, doubled the line throughput
- Availability of components: Widely available components at the time when the circuit was built
- Design testing: Circuit was in constant use by my friend for over a year
- Applications: Maximizing RS-232 line throughput on long cable runs
- Power supply: +-12V DC power supply 80 mA
- Estimated component cost: Few dollars
- Safety considerations: No special safety considerations

## Circuit description

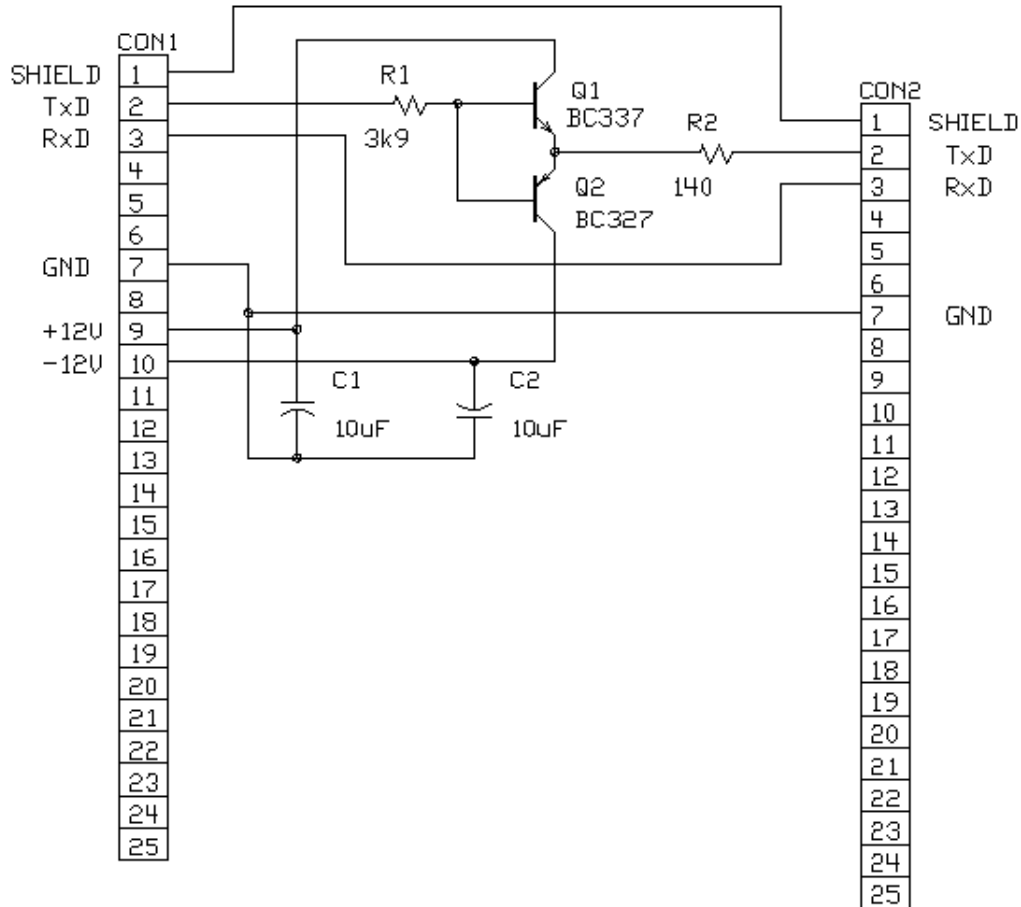
This is a simple serial port buffer circuit I designed for a friend to speed up his SLIP connection in campus computer network "TRINET" of Helsinki University of Technology. The problem in the network was that the RS232 connections from rooms to terminal server were long and made of bad quality wiring.

The circuit is a simple buffer which adds more driving capacity to PC serial port for the signal to go successfully from PC computer to terminal server (other direction had no problems). The computer is connected to connector CON1 and the buffered output is available at CON2. With this circuit the speed of RS232 connection to terminal server could be successfully raised from 9600 bps to 38400 bps.

The circuit is basically a two transistor buffer consisting of transistors Q1 and Q2 which can drive up to 1A current pulses, but the maximum output current of the circuit is limited by resistor R2. Value R2 was experimentally selected by testing resistor values in range of 22 ohm to 270 ohm and value 140 ohm gave best results (it provides quite good impedance matching to cable used). It is a good idea to use at least 1W resistor in place of R2 to make sure that it does not overheat in output short circuit situation (RS232 devices must withstand that to meet the standard).

The circuit was designed to be a compact box which is powered through D25 connector as some commercial RS232 buffer circuits. The idea is to feed the power to the buffer unit through serial port voltage test pins 9 and 10. The power was taken from an external power supply (cheap universal wall transformer) and wired to the D25 connector by modifying the cable connected between computer and the buffer circuit. The circuit in this configuration takes maximally continuous current of about 100 mA.

RS-232C buffer for high speed tranfer over long wires  
 (C) Tomi Engdahl 1994  
 Used succesfully in speeding up TRINET connections in Otaniemi




---

Tomi Engdahl <then@delta.hut.fi>

## Interfacing the Serial / RS232 Port

The Serial Port is harder to interface than the Parallel Port. In most cases, any device you connect to the serial port will need the serial transmission converted back to parallel so that it can be used. This can be done using a UART. On the software side of things, there are many more registers that you have to attend to than on a Standard Parallel Port. (SPP)

So what are the advantages of using serial data transfer rather than parallel?

1. Serial Cables can be longer than Parallel cables. The serial port transmits a '1' as -3 to -25 volts and a '0' as +3 to +25 volts where as a parallel port transmits a '0' as 0v and a '1' as 5v. Therefore the serial port can have a maximum swing of 50V compared to the parallel port which has a maximum swing of 5 Volts. Therefore cable loss is not going to be as much of a problem for serial cables than they are for parallel.
2. You don't need as many wires than parallel transmission. If your device needs to be mounted a far distance away from the computer then 3 core cable (Null Modem Configuration) is going to be a lot cheaper than running 19 or 25 core cable. However you must take into account the cost of the interfacing at each end.
3. Infra Red devices have proven quite popular recently. You may of seen many electronic diaries and palmtop computers which have infra red capabilities build in. However could you imagine transmitting 8 bits of data at the one time across the room and being able to (from the devices point of view) decipher which bits are which? Therefore serial transmission is used where one bit is sent at a time. IrDA-1 (The first infra red specifications) was capable of 115.2k baud and was interfaced into a UART. The pulse length however was cut down to 3/16th of a RS232 bit length to conserve power considering these devices are mainly used on diaries, laptops and palmtops.
4. Microcontroller's have also proven to be quite popular recently. Many of these have in built SCI (Serial Communications Interfaces) which can be used to talk to the outside world. Serial Communication reduces the pin count of these MPU's. Only two pins are commonly used, Transmit Data (TXD) and Receive Data (RXD) compared with at least 8 pins if you use a 8 bit Parallel method (You may also require a Strobe).

## **Part 1 : Hardware (PC's)**

Hardware Properties

Serial Pinouts (D25 and D9 connectors)

Pin Functions

Null Modems

Loopback Plugs

DTE/DCE Speeds

Flow Control

The UART (8250's and Compatibles)

Type of UARTS (For PC's)

## **Part 2 : Serial Ports' Registers (PC's)**

Port Addresses and IRQ's

Table of Registers

DLAB ?

Interrupt Enable Register (IER)

Interrupt Identification Register (IIR)

First In / First Out Control Register (FCR)

Line Control Register (LCR)

Modem Control Register (MCR)

Line Status Register (LSR)

Modem Status Register (MSR)

Scratch Register

## **Part 3 : Programming (PC's)**

Polling or Interrupt Driven?

Source Code - Termpoll.c (Polling Version)

Source Code - Buff1024.c (ISR Version)

Interrupt Vectors

Interrupt Service Routine

UART Configuration

Main Routine (Loop)

Determining the type of UART via Software

## **Part 4 : External Hardware - Interfacing Methods**

RS-232 Waveforms

RS-232 Level Converters

Making use of the Serial Format

8250 and compatible UART's

CDP6402, AY-5-1015 / D36402R-9 etc UARTs

Microcontrollers

---

## *Part One : Hardware (PC's)*

---

Hardware Properties

---

Devices which use serial cables for their communication are split into two categories. These are DCE (Data Communications Equipment) and DTE (Data Terminal Equipment.) Data Communications Equipment are devices such as your modem, TA adapter, plotter etc while Data Terminal Equipment is your Computer or Terminal.

The electrical specifications of the serial port is contained in the EIA (Electronics Industry Association) RS232C standard. It states many parameters such as -

1. A "Space" (logic 0) will be between +3 and +25 Volts.
2. A "Mark" (Logic 1) will be between -3 and -25 Volts.
3. The region between +3 and -3 volts is undefined.
4. An open circuit voltage should never exceed 25 volts. (In Reference to GND)
5. A short circuit current should not exceed 500mA. The driver should be able to handle this without damage. (Take note of this one!)

Above is no where near a complete list of the EIA standard. Line Capacitance, Maximum Baud Rates etc are also included. For more information please consult the EIA RS232-C standard. It is interesting to note however, that the RS232C standard specifies a maximum baud rate of 20,000 BPS!, which is rather slow by today's standards. A new standard, RS-232D has been recently released.

Serial Ports come in two "sizes", There are the D-Type 25 pin connector and the D-Type 9 pin connector both of which are male on the back of the PC, thus you will require a female connector on your device. Below is a table of pin connections for the 9 pin and 25 pin D-Type connectors.

### Serial Pinouts (D25 and D9 Connectors)

---

<b>D-Type-25 Pin No.</b>	<b>D-Type-9 Pin No.</b>	<b>Abbreviation</b>	<b>Full Name</b>
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

Table 1 : D Type 9 Pin and D Type 25 Pin Connectors

## Pin Functions

---

Abbreviation	Full Name	Function
TD	Transmit Data	Serial Data Output (TXD)
RD	Receive Data	Serial Data Input (RXD)
CTS	Clear to Send	This line indicates that the Modem is ready to exchange data.
DCD	Data Carrier Detect	When the modem detects a "Carrier" from the modem at the other end of the phone line, this Line becomes active.
DSR	Data Set Ready	This tells the UART that the modem is ready to establish a link.
DTR	Data Terminal Ready	This is the opposite to DSR. This tells the Modem that the UART is ready to link.
RTS	Request To Send	This line informs the Modem that the UART is ready to exchange data.
RI	Ring Indicator	Goes active when modem detects a ringing signal from the PSTN.

## Null Modems

---

A Null Modem is used to connect two DTE's together. This is commonly used as a cheap way to network games or to transfer files between computers using Zmodem Protocol, Xmodem Protocol etc. This can also be used with many Microprocessor Development Systems.

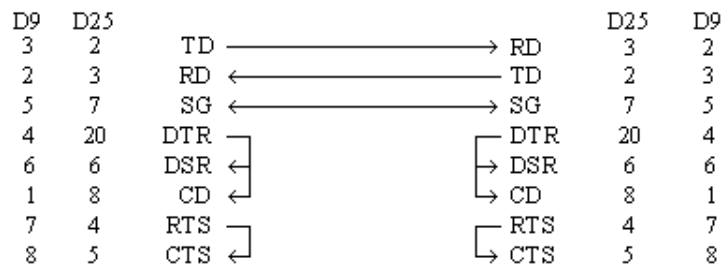


Figure 1 : Null Modem Wiring Diagram

Above is my preferred method of wiring a Null Modem. It only requires 3 wires (TD, RD & SG) to be wired straight through thus is more cost effective to use with long cable runs. The theory of operation is reasonably easy. The aim is to make to computer think it is talking to a modem rather than another computer. Any data transmitted from the first computer must be received by the second thus TD is connected to RD. The second computer must have the same set-up thus RD is connected to TD. Signal Ground (SG) must also be connected so both grounds are common to each computer.



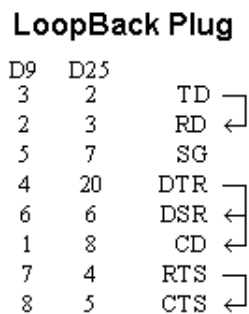
The Data Terminal Ready is looped back to Data Set Ready and Carrier Detect on both computers. When the Data Terminal Ready is asserted active, then the Data Set Ready and Carrier Detect immediately become active. At this point the computer thinks the Virtual Modem to which it is connected is ready and has detected the carrier of the other modem.

All left to worry about now is the Request to Send and Clear To Send. As both computers communicate together at the same speed, flow control is not needed thus these two lines are also linked together on each computer. When the computer wishes to send data, it asserts the Request to Send high and as it's hooked together with the Clear to Send, It immediately gets a reply that it is ok to send and does so.

Notice that the ring indicator is not connected to anything of each end. This line is only used to tell the computer that there is a ringing signal on the phone line. As we don't have a modem connected to the phone line this is left disconnected.

## LoopBack Plug

---



This loopback plug can come in extremely handy when writing Serial / RS232 Communications Programs. It has the receive and transmit lines connected together, so that anything transmitted out of the Serial Port is immediately received by the same port. If you connect this to a Serial Port and load a Terminal Program, anything you type will be immediately displayed on the screen. This can be used with the examples later in this tutorial.

*Please note that this is not intended for use with Diagnostic Programs and thus will probably not work. For these programs you require a differently wired Loop Back plug which may vary from program to program.*

Figure 2 : Loopback Plug Wiring Diagram

## DTE / DCE Speeds

---

We have already talked briefly about DTE & DCE. A typical Data Terminal Device is a computer and a typical Data Communications Device is a Modem. Often people will talk about DTE to DCE or DCE to DCE speeds. DTE to DCE is the speed between your modem and computer, sometimes referred to as your terminal speed. This should run at faster speeds than the DCE to DCE speed. DCE to DCE is the link between modems, sometimes called the line speed.

Most people today will have 28.8K or 33.6K modems. Therefore we should expect the DCE to DCE speed to be either 28.8K or 33.6K. Considering the high speed of the modem we should expect the DTE to DCE speed to be about 115,200 BPS.(Maximum Speed of the 16550a UART) This is where some people often fall into a trap. The communications program which they use have settings for DCE to DTE speeds. However they see 9.6 KBPS, 14.4 KBPS etc and think it

is your modem speed.

Today's Modems should have Data Compression build into them. This is very much like PK-ZIP but the software in your modem compresses and decompresses the data. When set up correctly you can expect compression ratios of 1:4 or even higher. 1 to 4 compression would be typical of a text file. If we were transferring that text file at 28.8K (DCE-DCE), then when the modem compresses it you are actually transferring 115.2 KBPS between computers and thus have a DCE-DTE speed of 115.2 KBPS. Thus this is why the DCE-DTE should be much higher than your modem's connection speed.

Some modem manufacturers quote a maximum compression ratio as 1:8. Lets say for example its on a new 33.6 KBPS modem then we may get a maximum 268,800 BPS transfer between modem and UART. If you only have a 16550a which can do 115,200 BPS tops, then you would be missing out on a extra bit of performance. Buying a 16C650 should fix your problem with a maximum transfer rate of 230,400 BPS.

However don't abuse your modem if you don't get these rates. These are MAXIMUM compression ratios. In some instances if you try to send a already compressed file, your modem can spend more time trying the compress it, thus you get a transmission speed less than your modem's connection speed. If this occurs try turning off your data compression. This should be fixed on newer modems. Some files compress easier than others thus any file which compresses easier is naturally going to have a higher compression ratio.

## Flow Control

---

So if our DTE to DCE speed is several times faster than our DCE to DCE speed the PC can send data to your modem at 115,200 BPS. Sooner or later data is going to get lost as buffers overflow, thus flow control is used. Flow control has two basic varieties, Hardware or Software.

Software flow control, sometimes expressed as Xon/Xoff uses two characters Xon and Xoff. Xon is normally indicated by the ASCII 17 character where as the ASCII 19 character is used for Xoff. The modem will only have a small buffer so when the computer fills it up the modem sends a Xoff character to tell the computer to stop sending data. Once the modem has room for more data it then sends a Xon character and the computer sends more data. This type of flow control has the advantage that it doesn't require any more wires as the characters are sent via the TD/RD lines. However on slow links each character requires 10 bits which can slow communications down.

Hardware flow control is also known as RTS/CTS flow control. It uses two wires in your serial cable rather than extra characters transmitted in your data lines. Thus hardware flow control will not slow down transmission times like Xon-Xoff does. When the computer wishes to send data it takes active the Request to Send line. If the modem has room for this data, then the modem will reply by taking active the Clear to Send line and the computer starts sending data. If the modem does not have the room then it will not send a Clear to Send.

## The UART (8250 and Compatibles)

---

UART stands for Universal Asynchronous Receiver / Transmitter. Its the little box of tricks found on your serial card which plays the little games with your modem or other connected devices. Most cards will have the UART's integrated into other chips which may also control your parallel port, games port, floppy or hard disk drives and are typically surface mount devices. The 8250 series, which includes the 16450, 16550, 16650, & 16750 UARTs are the most commonly found type in your PC. Later we will look at other types which can be used in your homemade devices and projects.

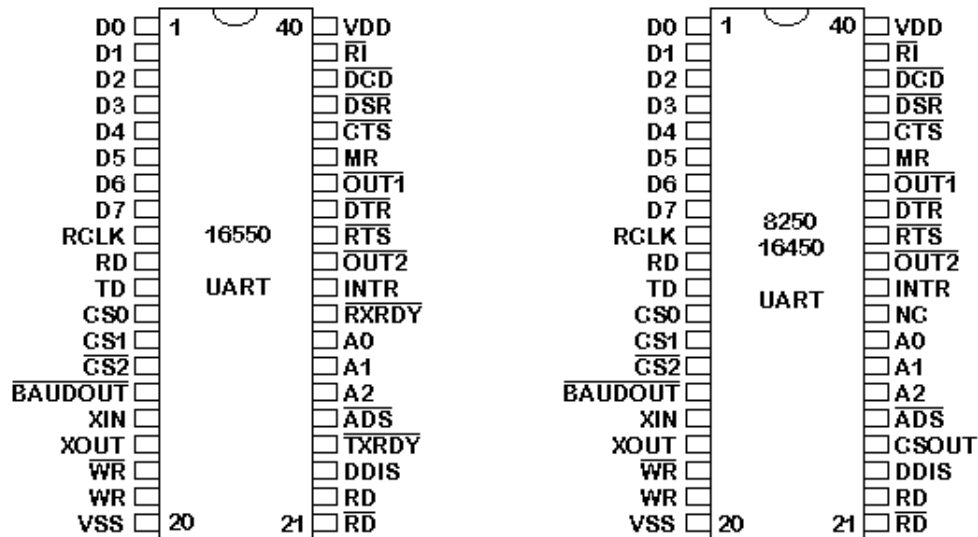


Figure 3 : Pin Diagrams for 16550, 16450 & 8250 UARTs

The 16550 is chip compatible with the 8250 & 16450. The only two differences are pins 24 & 29. On the 8250 Pin 24 was chip select out which functioned only as a indicator to if the chip was active or not. Pin 29 was not connected on the 8250/16450 UARTs. The 16550 introduced two new pins in their place. These are Transmit Ready and Receive Ready which can be implemented with DMA (Direct Memory Access). These Pins have two different modes of operation. Mode 0 supports single transfer DMA where as Mode 1 supports Multi-transfer DMA.

Mode 0 is also called the 16450 mode. This mode is selected when the FIFO buffers are disabled via Bit 0 of the FIFO Control Register or When the FIFO buffers are enabled but DMA Mode Select = 0. (Bit 3 of FCR) In this mode RXRDY is active low when at least one character (Byte) is present in the Receiver Buffer. RXRDY will go inactive high when no more characters are left in the Receiver Buffer. TXRDY will be active low when there are no characters in the Transmit Buffer. It will go inactive high after the first character / byte is loaded into the Transmit Buffer.

Mode 1 is when the FIFO buffers are active and the DMA Mode Select = 1. In Mode 1, RXRDY will go active low when the trigger level is reached or when 16550 Time Out occurs and will return to inactive state when no more characters are left in the FIFO. TXRDY will be active when no characters are present in the Transmit Buffer and will go inactive when the FIFO Transmit Buffer is completely Full.

All the UARTs pins are TTL compatible. That includes TD, RD, RI, DCD, DSR, CTS, DTR and

RTS which all interface into your serial plug, typically a D-type connector. Therefore RS232 Level Converters (which we talk about in detail later) are used. These are commonly the DS1489 Receiver and the DS1488 as the PC has +12 and -12 volt rails which can be used by these devices. The RS232 Converters will convert the TTL signal into RS232 Logic Levels.

Pin No.	Name	Notes
Pin 1:8	D0:D7	Data Bus
Pin 9	RCLK	Receiver Clock Input. The frequency of this input should equal the receivers baud rate * 16
Pin 10	RD	Receive Data
Pin 11	TD	Transmit Data
Pin 12	CS0	Chip Select 0 - Active High
Pin 13	CS1	Chip Select 1 - Active High
Pin 14	nCS2	Chip Select 2 - Active Low
Pin 15	nBAUDOUT	Baud Output - Output from Programmable Baud Rate Generator. Frequency = (Baud Rate x 16)
Pin 16	XIN	External Crystal Input - Used for Baud Rate Generator Oscillator
Pin 17	XOUT	External Crystal Output
Pin 18	nWR	Write Line - Inverted
Pin 19	WR	Write Line - Not Inverted
Pin 20	VSS	Connected to Common Ground
Pin 21	RD	Read Line - Inverted
Pin 22	nRD	Read Line - Not Inverted
Pin 23	DDIS	Driver Disable. This pin goes low when CPU is reading from UART. Can be connected to Bus Transceiver in case of high capacity data bus.
Pin 24	nTXRDY	Transmit Ready
Pin 25	nADS	Address Strobe. Used if signals are not stable during read or write cycle
Pin 26	A2	Address Bit 2
Pin 27	A1	Address Bit 1
Pin 28	A0	Address Bit 0
Pin 29	nRXRDY	Receive Ready
Pin 30	INTR	Interrupt Output
Pin 31	nOUT2	User Output 2
Pin 32	nRTS	Request to Send
Pin 33	nDTR	Data Terminal Ready

Pin 34	nOUT1	User Output 1
Pin 35	MR	Master Reset
Pin 36	nCTS	Clear To Send
Pin 37	nDSR	Data Set Ready
Pin 38	nDCD	Data Carrier Detect
Pin 39	nRI	Ring Indicator
Pin 40	VDD	+ 5 Volts

Table 2 : Pin Assignments for 16550A UART

The UART requires a Clock to run. If you look at your serial card a common crystal found is either a 1.8432 MHZ or a 18.432 MHZ Crystal. The crystal is connected to the XIN-XOUT pins of the UART using a few extra components which help the crystal to start oscillating. This clock will be used for the Programmable Baud Rate Generator which directly interfaces into the transmit timing circuits but not directly into the receiver timing circuits. For this an external connection must be made from pin 15 (BaudOut) to pin 9 (Receiver clock in.) Note that the clock signal will be at Baudrate \* 16.

If you are serious about pursuing the 16550 UART used in your PC further, then would suggest downloading a copy of the PC16550D data sheet from National Semiconductors Site. Data sheets are available in .PDF format so you will need Adobe Acrobat Reader to read these. Texas Instruments has released the 16750 UART which has 64 Byte FIFO's. Data Sheets for the TL16C750 are available from the Texas Instruments Site.

### Types of UARTS (For PC's)

---

- 8250 First UART in this series. It contains no scratch register. The 8250A was an improved version of the 8250 which operates faster on the bus side.
  - 8250A This UART is faster than the 8250 on the bus side. Looks exactly the same to software than 16450.
  - 8250B Very similar to that of the 8250 UART.
  - 16450 Used in AT's (Improved bus speed over 8250's). Operates comfortably at 38.4KBPS. Still quite common today.
  - 16550 This was the first generation of buffered UART. It has a 16 byte buffer, however it doesn't work and is replaced with the 16550A.
  - 16550A Is the most common UART use for high speed communications eg 14.4K & 28.8K Modems. They made sure the FIFO buffers worked on this UART.
  - 16650 Very recent breed of UART. Contains a 32 byte FIFO, Programmable X-On / X-Off characters and supports power management.
  - 16750 Produced by Texas Instruments. Contains a 64 byte FIFO.
- 

## *Part Two : Serial Port's Registers (PC's)*

---

## Port Addresses & IRQ's

---

Name	Address	IRQ
COM 1	3F8	4
COM 2	2F8	3
COM 3	3E8	4
COM 4	2E8	3

Table 3 : Standard Port Addresses

Above is the standard port addresses. These should work for most P.C's. If you just happen to be lucky enough to own a IBM P/S2 which has a micro-channel bus, then expect a different set of addresses and IRQ's. Just like the LPT ports, the base addresses for the COM ports can be read from the BIOS Data Area.

Start Address	Function
0000:0400	COM1's Base Address
0000:0402	COM2's Base Address
0000:0404	COM3's Base Address
0000:0406	COM4's Base Address

Table 4 - COM Port Addresses in the BIOS Data Area;

The above table shows the address at which we can find the Communications (COM) ports addresses in the BIOS Data Area. Each address will take up 2 bytes. The following sample program in C, shows how you can read these locations to obtain the addresses of your communications ports.

```
#include
#include

void main(void)
{
    unsigned int far *ptraddr; /* Pointer to location of Port Addresses */
    unsigned int address;      /* Address of Port */
    int a;

    ptraddr=(unsigned int far *)0x00000400;

    for (a = 0; a < 4; a++)
    {
        address = *ptraddr;
        if (address == 0)
            printf("No port found for COM%d \n",a+1);
        else
            printf("Address assigned to COM%d is %Xh\n",a+1,address);
        *ptraddr++;
    }
}
```

}  
}

## Table of Registers

---

Base Address	DLAB	Read/Write	Abr.	Register Name
+ 0	=0	Write	-	Transmitter Holding Buffer
	=0	Read	-	Receiver Buffer
	=1	Read/Write	-	Divisor Latch Low Byte
+ 1	=0	Read/Write	IER	Interrupt Enable Register
	=1	Read/Write	-	Divisor Latch High Byte
+ 2	-	Read	IIR	Interrupt Identification Register
	-	Write	FCR	FIFO Control Register
+ 3	-	Read/Write	LCR	Line Control Register
+ 4	-	Read/Write	MCR	Modem Control Register
+ 5	-	Read	LSR	Line Status Register
+ 6	-	Read	MSR	Modem Status Register
+ 7	-	Read/Write	-	Scratch Register

Table 5 : Table of Registers

## DLAB ?

---

You will have noticed in the table of registers that there is a DLAB column. When DLAB is set to '0' or '1' some of the registers change. This is how the UART is able to have 12 registers (including the scratch register) through only 8 port addresses. DLAB stands for Divisor Latch Access Bit. When DLAB is set to '1' via the line control register, two registers become available from which you can set your speed of communications measured in bits per second.

The UART will have a crystal which should oscillate around 1.8432 MHZ. The UART incorporates a divide by 16 counter which simply divides the incoming clock signal by 16. Assuming we had the 1.8432 MHZ clock signal, that would leave us with a maximum, 115,200 hertz signal making the UART capable of transmitting and receiving at 115,200 Bits Per Second (BPS). That would be fine for some of the faster modems and devices which can handle that speed, but others just wouldn't communicate at all. Therefore the UART is fitted with a Programmable Baud Rate Generator which is controlled by two registers.

Lets say for example we only wanted to communicate at 2400 BPS. We worked out that we would have to divide 115,200 by 48 to get a workable 2400 Hertz Clock. The "Divisor", in this case 48, is stored in the two registers controlled by the "Divisor Latch Access Bit". This divisor can be any number which can be stored in 16 bits (ie 0 to 65535). The UART only has a 8 bit data bus, thus this is where the two registers are used. The first register (Base + 0) when DLAB = 1 stores the "Divisor latch low byte" where as the second register (base + 1 when DLAB = 1)

stores the "Divisor latch high byte."

Below is a table of some more common speeds and their divisor latch high bytes & low bytes. Note that all the divisors are shown in Hexadecimal.

Speed (BPS)	Divisor (Dec)	Divisor Latch High Byte	Divisor Latch Low Byte
50	2304	09h	00h
300	384	01h	80h
600	192	00h	C0h
2400	48	00h	30h
4800	24	00h	18h
9600	12	00h	0Ch
19200	6	00h	06h
38400	3	00h	03h
57600	2	00h	02h
115200	1	00h	01h

Table 6 : Table of Commonly Used Baudrate Divisors

## Interrupt Enable Register (IER)

---

Bit	Notes
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Enables Low Power Mode (16750)
Bit 4	Enables Sleep Mode (16750)
Bit 3	Enable Modem Status Interrupt
Bit 2	Enable Receiver Line Status Interrupt
Bit 1	Enable Transmitter Holding Register Empty Interrupt
Bit 0	Enable Received Data Available Interrupt

Table 7 : Interrupt Enable Register

The Interrupt Enable Register could possibly be one of the easiest registers on a UART to understand. Setting Bit 0 high enables the Received Data Available Interrupt which generates an interrupt when the receiving register/FIFO contains data to be read by the CPU.

Bit 1 enables Transmit Holding Register Empty Interrupt. This interrupts the CPU when the transmitter buffer is empty. Bit 2 enables the receiver line status interrupt. The UART will interrupt when the receiver line status changes. Likewise for bit 3 which enables the modem status interrupt. Bits 4 to 7 are the easy ones. They are simply reserved. (If only everything was that easy!)



## Interrupt Identification Register (IIR)

---

Bit	Notes		
Bits 6 and 7	Bit 6	Bit 7	
	0	0	No FIFO
	0	1	FIFO Enabled but Unusable
	1	1	FIFO Enabled
Bit 5	64 Byte Fifo Enabled (16750 only)		
Bit 4	Reserved		
Bit 3	0	Reserved on 8250, 16450	
	1	16550 Time-out Interrupt Pending	
Bits 1 and 2	Bit 2	Bit 1	
	0	0	Modem Status Interrupt
	0	1	Transmitter Holding Register Empty Interrupt
	1	0	Received Data Available Interrupt
	1	1	Receiver Line Status Interrupt
Bit 0	0	Interrupt Pending	
	1	No Interrupt Pending	

Table 8 : Interrupt Identification Register

The interrupt identification register is a read only register. Bits 6 and 7 give status on the FIFO Buffer. When both bits are '0' no FIFO buffers are active. This should be the only result you will get from a 8250 or 16450. If bit 7 is active but bit 6 is not active then the UART has it's buffers enabled but are unusable. This occurs on the 16550 UART where a bug in the FIFO buffer made the FIFO's unusable. If both bits are '1' then the FIFO buffers are enabled and fully operational.

Bits 4 and 5 are reserved. Bit 3 shows the status of the time-out interrupt on a 16550 or higher.

Lets jump to Bit 0 which shows whether an interrupt has occurred. If an interrupt has occurred it's status will shown by bits 1 and 2. These interrupts work on a priority status. The Line Status Interrupt has the highest Priority, followed by the Data Available Interrupt, then the Transmit Register Empty Interrupt and then the Modem Status Interrupt which has the lowest priority.

## First In / First Out Control Register (FCR)

---

Bit	Notes		
Bits 6 and 7	Bit 7	Bit 6	Interrupt Trigger Level
	0	0	1 Byte
	0	1	4 Bytes
	1	0	8 Bytes
	1	1	14 Bytes
Bit 5	Enable 64 Byte FIFO (16750 only)		
Bit 4	Reserved		
Bit 3	DMA Mode Select. Change status of RXRDY & TXRDY pins from mode 1 to mode 2.		
Bit 2	Clear Transmit FIFO		
Bit 1	Clear Receive FIFO		
Bit 0	Enable FIFO's		

Table 9 : FIFO Control Register

The FIFO register is a write only register. This register is used to control the FIFO (First In / First Out) buffers which are found on 16550's and higher.

Bit 0 enables the operation of the receive and transmit FIFO's. Writing a '0' to this bit will disable the operation of transmit and receive FIFO's, thus you will loose all data stored in these FIFO buffers.

Bit's 1 and 2 control the clearing of the transmit or receive FIFO's. Bit 1 is responsible for the receive buffer while bit 2 is responsible for the transmit buffer. Setting these bits to 1 will only clear the contents of the FIFO and will not affect the shift registers. These two bits are self resetting, thus you don't need to set the bits to '0' when finished.

Bit 3 enables the DMA mode select which is found on 16550 UARTs and higher. More on this later. Bits 4 and 5 are those easy type again, Reserved.

Bits 6 and 7 are used to set the triggering level on the Receive FIFO. For example if bit 7 was set to '1' and bit 6 was set to '0' then the trigger level is set to 8 bytes. When there is 8 bytes of data in the receive FIFO then the Received Data Available interrupt is set. See (IIR)

## Line Control Register (LCR)

---

Bit 7	1			
	Divisor Latch Access Bit			
	0			
	Access to Receiver buffer, Transmitter buffer & Interrupt Enable Register			
Bit 6	Set Break Enable			
Bits 3, 4 And 5	Bit 5	Bit 4	Bit 3	Parity Select
	X	X	0	No Parity
	0	0	1	Odd Parity
	0	1	1	Even Parity
	1	0	1	High Parity (Sticky)
	1	1	1	Low Parity (Sticky)
Bit 2	Length of Stop Bit			
	0	One Stop Bit		
	1	2 Stop bits for words of length 6,7 or 8 bits or 1.5 Stop Bits for Word lengths of 5 bits.		
Bits 0 And 1	Bit 1	Bit 0	Word Length	
	0	0	5 Bits	
	0	1	6 Bits	
	1	0	7 Bits	
	1	1	8 Bits	

Table 10 : Line Control Register

The Line Control register sets the basic parameters for communication. Bit 7 is the Divisor Latch Access Bit or DLAB for short. We have already talked about what it does. (See DLAB?) Bit 6 Sets break enable. When active, the TD line goes into "Spacing" state which causes a break in the receiving UART. Setting this bit to '0' Disables the Break.

Bits 3,4 and 5 select parity. If you study the 3 bits, you will find that bit 3 controls parity. That is, if it is set to '0' then no parity is used, but if it is set to '1' then parity is used. Jumping to bit 5, we can see that it controls sticky parity. Sticky parity is simply when the parity bit is always transmitted and checked as a '1' or '0'. This has very little success in checking for errors as if the first 4 bits contain errors but the sticky parity bit contains the appropriately set bit, then a parity error will not result. Sticky high parity is the use of a '1' for the parity bit, while the opposite, sticky low parity is the use of a '0' for the parity bit.

If bit 5 controls sticky parity, then turning this bit off must produce normal parity provided bit 3 is still set to '1'. Odd parity is when the parity bit is transmitted as a '1' or '0' so that there is an odd number of 1's. Even parity must then be the parity bit produces an even number of 1's. This provides better error checking but still is not perfect, thus CRC-32 is often used for software error correction. If one bit happens to be inverted with even or odd parity set, then a parity error will occur, however if two bits are flipped in such a way that it produces the correct parity bit then a parity error will not occur.

Bit 2 sets the length of the stop bits. Setting this bit to '0' will produce one stop bit, however setting it to '1' will produce either 1.5 or 2 stop bits depending upon the word length. Note that the receiver only checks the first stop bit.

Bits 0 and 1 set the word length. This should be pretty straight forward. A word length of 8 bits is most commonly used today.

## Modem Control Register (MCR)

---

Bit	Notes
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Autoflow Control Enabled (16750 only)
Bit 4	LoopBack Mode
Bit 3	Aux Output 2
Bit 2	Aux Output 1
Bit 1	Force Request to Send
Bit 0	Force Data Terminal Ready

Table 11 : Modem Control Register

The Modem Control Register is a Read/Write Register. Bits 5,6 and 7 are reserved. Bit 4 activates the loopback mode. In Loopback mode the transmitter serial output is placed into marking state. The receiver serial input is disconnected. The transmitter out is looped back to the receiver in. DSR, CTS, RI & DCD are disconnected. DTR, RTS, OUT1 & OUT2 are connected to the modem control inputs. The modem control output pins are then placed in an inactive state. In this mode any data which is placed in the transmitter registers for output is received by the receiver circuitry on the same chip and is available at the receiver buffer. This can be used to test the UART's operation.

Aux Output 2 maybe connected to external circuitry which controls the UART-CPU interrupt process. Aux Output 1 is normally disconnected, but on some cards is used to switch between a 1.8432MHZ crystal to a 4MHZ crystal which is used for MIDI. Bits 0 and 1 simply control their relevant data lines. For example setting bit 1 to '1' makes the request to send line active.

## Line Status Register (LSR)

---

Bit	Notes
Bit 7	Error in Received FIFO
Bit 6	Empty Data Holding Registers
Bit 5	Empty Transmitter Holding Register
Bit 4	Break Interrupt
Bit 3	Framing Error
Bit 2	Parity Error
Bit 1	Overrun Error
Bit 0	Data Ready

Table 12 : Line Status Register

The line status register is a read only register. Bit 7 is the error in received FIFO bit. This bit is high when at least one break, parity or framing error has occurred on a byte which is contained in the FIFO.

When bit 6 is set, both the transmitter holding register and the shift register are empty. The UART's holding register holds the next byte of data to be sent in parallel fashion. The shift register is used to convert the byte to serial, so that it can be transmitted over one line. When bit 5 is set, only the transmitter holding register is empty. So what's the difference between the two? When bit 6, the transmitter holding and shift registers are empty, no serial conversions are taking place so there should be no activity on the transmit data line. When bit 5 is set, the transmitter holding register is empty, thus another byte can be sent to the data port, but a serial conversion using the shift register may be taking place.

The break interrupt (Bit 4) occurs when the received data line is held in a logic state '0' (Space) for more than the time it takes to send a full word. That includes the time for the start bit, data bits, parity bits and stop bits.

A framing error (Bit 3) occurs when the last bit is not a stop bit. This may occur due to a timing error. You will most commonly encounter a framing error when using a null modem linking two computers or a protocol analyzer when the speed at which the data is being sent is different to that of what you have the UART set to receive it at.

An overrun error normally occurs when your program can't read from the port fast enough. If you don't get an incoming byte out of the register fast enough, and another byte just happens to be received, then the last byte will be lost and an overrun error will result.

Bit 0 shows data ready, which means that a byte has been received by the UART and is at the receiver buffer ready to be read.

## Modem Status Register (MSR)

---

Bit	Notes
Bit 7	Carrier Detect
Bit 6	Ring Indicator
Bit 5	Data Set Ready
Bit 4	Clear To Send
Bit 3	Delta Data Carrier Detect
Bit 2	Trailing Edge Ring Indicator
Bit 1	Delta Data Set Ready
Bit 0	Delta Clear to Send

Table 13 : Modem Status Register

Bit 0 of the modem status register shows delta clear to send, delta meaning a change in, thus delta clear to send means that there was a change in the clear to send line, since the last read of this register. This is the same for bits 1 and 3. Bit 1 shows a change in the Data Set Ready line where as Bit 3 shows a change in the Data Carrier Detect line. Bit 2 is the Trailing Edge Ring Indicator which indicates that there was a transformation from low to high state on the Ring Indicator line.

Bits 4 to 7 show the current state of the data lines when read. Bit 7 shows Carrier Detect, Bit 6 shows Ring Indicator, Bit 5 shows Data Set Ready & Bit 4 shows the status of the Clear To Send line.

## Scratch Register

---

The scratch register is not used for communications but rather used as a place to leave a byte of data. The only real use it has is to determine whether the UART is a 8250/8250B or a 8250A/16450 and even that is not very practical today as the 8250/8250B was never designed for AT's and can't hack the bus speed.

### **Part 3 : Programming (PC's)**

Polling or Interrupt Driven?

Source Code - Termpoll.c (Polling Version)

Source Code - Buff1024.c (ISR Version)

Interrupt Vectors

Interrupt Service Routine

UART Configuration

Main Routine (Loop)

Determining the type of UART via Software

### **Part 4 : External Hardware - Interfacing Methods**

RS-232 Waveforms

RS-232 Level Converters

Making use of the Serial Format

8250 and compatible UART's  
CDP6402, AY-5-1015 / D36402R-9 etc UARTs  
Microcontrollers

Copyright 1999-2001 Craig Peacock 19th August 2001.

# TIFF Image Creation

Written by Paul Bourke  
August 1998

---

The following demonstrates how to create 24 bit colour RGB TIFF (Tagged Image File Format) files. That is, how to create images from your own software that can be then opened and manipulated with image handling software, for example: GIMP, PhotoShop, etc. Given this aim, this document illustrates the "minimal" requirements necessary to create a TIFF file, it does not provide enough information for writing a TIFF file reader. For more information on the full TIFF specification the following postscript and pdf files describe the TIFF version 6.

tiff.ps.gz    tiff.pdf.gz

The basic structure of a TIFF file is as follows:

The first 8 bytes forms the **header**. The first two bytes of which is either "II" for little endian byte ordering or "MM" for big endian byte ordering. In what follows we'll be assuming little endian ordering. Note: any true TIFF reading software is supposed to be handle both types. The next two bytes of the header should be 0 and 42<sub>dec</sub> (2a<sub>hex</sub>). The remaining 4 bytes of the header is the offset from the start of the file to the first "Image File Directory" (IFD), this normally follows the image data it applies to. In the example below there is only one image and one IFD.

An **IFD** consists of two bytes indicating the number of entries followed by the entries themselves. The IFD is terminated with 4 byte offset to the next IFD or 0 if there are none. A TIFF file must contain at least one IFD!

Each **IFD entry** consists of 12 bytes. The first two bytes identifies the tag type (as in Tagged Image File Format). The next two bytes are the field type (byte, ASCII, short int, long int, ...). The next four bytes indicate the number of values. The last four bytes is either the value itself or an offset to the values. Considering the first IFD entry from the example given below:

```
    0100 0003 0000 0001 0064 0000
      |           |           |           |
tag  --+         |           |           |
short int -----+         |           |
one value -----+         |           |
value of 100  -----+         |           |
```

## Example

The following is an example using the TIFF file shown on the right, namely a black image with a single white pixel at the top left and the bottom right position. The image is 100 pixels wide by 200 pixels high.

A hex dump is given below along with matching pointers and locations marked in matching colours,

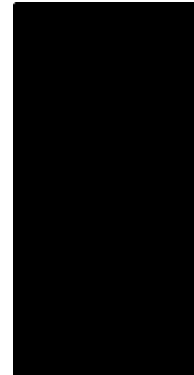


these colours further match the appropriate parts of the source code given later. The tags are underlined.

```
4d4d 002a 0000 ea68 ffff ff00 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000

.... ....  black 0's deleted  .... ....

0000 0000 00ff ffff 000e 0100 0003 0000
0001 0064 0000 0101 0003 0000 0001 00c8
0000 0102 0003 0000 0003 0000 eb16 0103
0003 0000 0001 0001 0000 0106 0003 0000
0001 0002 0000 0111 0004 0000 0001 0000
0008 0112 0003 0000 0001 0001 0000 0115
0003 0000 0001 0003 0000 0116 0003 0000
0001 00c8 0000 0117 0004 0000 0001 0000
ea60 0118 0003 0000 0003 0000 eb1c 0119
0003 0000 0003 0000 eb22 011c 0003 0000
0001 0001 0000 0153 0003 0000 0003 0000
eb28 0000 0000 0008 0008 0008 0000 0000
0000 00ff 00ff 00ff 0001 0001 0001
```



The above example uses 14<sub>dec</sub> (000e<sub>hex</sub>) directory entries.

- 0100 - Image width
- 0101 - Image height
- 0102 - Bits per sample (8)
- 0103 - Compression method (1 = uncompressed)
- 0106 - Photometric Interpretation (2 = RGB)
- 0111 - Strip Offsets
- 0112 - Orientation (1 = 0 top, 0 left hand side)
- 0115 - Samples per pixel (1)
- 0116 - Rows per strip (200 = image height)
- 0117 - Strip Byte Counts (60000 = 100 x 200 x 3)
- 0118 - Minimum sample value (0,0,0)
- 0119 - Maximum sample value (255,255,255)
- 011c - Planar configuration (1 = single image plane)
- 0153 - Sample format

---

## Source code example

The following is the guts of a C program to create a TIFF file of width nx, height ny. Each pixel is made up of 3 bytes, one byte for each of Red, Green, Blue. Each colour component ranges from 0 (black) to 255 (white).

```
/* Write the header */
WriteHexString(fp, "4d4d002a"); /* Little endian & TIFF identifier */
offset = nx * ny * 3 + 8;
putc((offset & 0xff000000) / 16777216, fp);
putc((offset & 0x00ff0000) / 65536, fp);
putc((offset & 0x0000ff00) / 256, fp);
putc((offset & 0x000000ff), fp);
```

```

/* Write the binary data */
for (j=0;j<ny;j++) {
    for (i=0;i<nx;i++) {
... calculate the RGB value between 0 and 255 ...
        fputc(red,fptr);
        fputc(green,fptr);
        fputc(blue,fptr);
    }
}

/* Write the footer */
WriteHexString(fptr,"000e"); /* The number of directory entries (14) */

/* Width tag, short int */
WriteHexString(fptr,"0100000300000001");
fputc((nx & 0xff00) / 256,fptr); /* Image width */
fputc((nx & 0x00ff),fptr);
WriteHexString(fptr,"0000");

/* Height tag, short int */
WriteHexString(fptr,"0101000300000001");
fputc((ny & 0xff00) / 256,fptr); /* Image height */
fputc((ny & 0x00ff),fptr);
WriteHexString(fptr,"0000");

/* Bits per sample tag, short int */
WriteHexString(fptr,"0102000300000003");
offset = nx * ny * 3 + 182;
putc((offset & 0xff000000) / 16777216,fptr);
putc((offset & 0x00ff0000) / 65536,fptr);
putc((offset & 0x0000ff00) / 256,fptr);
putc((offset & 0x000000ff),fptr);

/* Compression flag, short int */
WriteHexString(fptr,"0103000300000000100010000");

/* Photometric interpolation tag, short int */
WriteHexString(fptr,"0106000300000000100020000");

/* Strip offset tag, long int */
WriteHexString(fptr,"01110004000000001000000008");

/* Orientation flag, short int */
WriteHexString(fptr,"0112000300000000100010000");

/* Sample per pixel tag, short int */
WriteHexString(fptr,"0115000300000000100030000");

/* Rows per strip tag, short int */
WriteHexString(fptr,"01160003000000001");
fputc((ny & 0xff00) / 256,fptr);
fputc((ny & 0x00ff),fptr);
WriteHexString(fptr,"0000");

/* Strip byte count flag, long int */
WriteHexString(fptr,"01170004000000001");
offset = nx * ny * 3;
putc((offset & 0xff000000) / 16777216,fptr);
putc((offset & 0x00ff0000) / 65536,fptr);
putc((offset & 0x0000ff00) / 256,fptr);
putc((offset & 0x000000ff),fptr);

```

```

/* Minimum sample value flag, short int */
WriteHexString(fp_ptr,"0118000300000003");
offset = nx * ny * 3 + 188;
putc((offset & 0xff000000) / 16777216,fp_ptr);
putc((offset & 0x00ff0000) / 65536,fp_ptr);
putc((offset & 0x0000ff00) / 256,fp_ptr);
putc((offset & 0x000000ff),fp_ptr);

/* Maximum sample value tag, short int */
WriteHexString(fp_ptr,"0119000300000003");
offset = nx * ny * 3 + 194;
putc((offset & 0xff000000) / 16777216,fp_ptr);
putc((offset & 0x00ff0000) / 65536,fp_ptr);
putc((offset & 0x0000ff00) / 256,fp_ptr);
putc((offset & 0x000000ff),fp_ptr);

/* Planar configuration tag, short int */
WriteHexString(fp_ptr,"011c00030000000100010000");

/* Sample format tag, short int */
WriteHexString(fp_ptr,"0153000300000003");
offset = nx * ny * 3 + 200;
putc((offset & 0xff000000) / 16777216,fp_ptr);
putc((offset & 0x00ff0000) / 65536,fp_ptr);
putc((offset & 0x0000ff00) / 256,fp_ptr);
putc((offset & 0x000000ff),fp_ptr);

/* End of the directory entry */
WriteHexString(fp_ptr,"00000000");

/* Bits for each colour channel */
WriteHexString(fp_ptr,"000800080008");

/* Minimum value for each component */
WriteHexString(fp_ptr,"000000000000");

/* Maximum value per channel */
WriteHexString(fp_ptr,"00ff00ff00ff");

/* Samples per pixel for each channel */
WriteHexString(fp_ptr,"000100010001");

```