RS232 Cables, Wiring and Pinouts

RS232 Wiring Stuff

RS232 standards are defined by EIA/TIA (Electronic Industries Alliance /Telecommunications Industry Association). RS232 defines both the physical and electrical characteristics of the interface. RS232 is practically identical to ITU V.24 (signal description and names) and V.28 (electrical). RS232 is an Active LOW voltage driven interface and operates at +12V to -12V where:

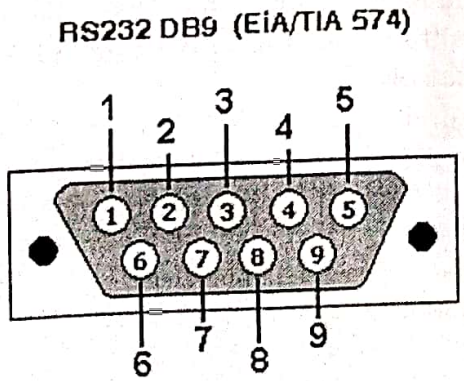
Signal = 0 (LOW) > +3.0V

Signal = 1 (HIGH) < -3.0V

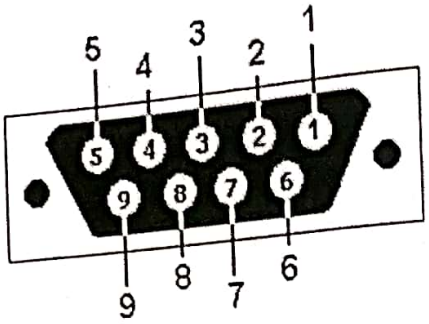
RS232 on DB9 (EIA/TIA 574)

Pin No.	Name	Notes/Description
1	DCD	Data Carrier Detect
2	RD	Receive Data (a.k.a RxD, Rx)
3	TD	Transmit Data (a.k.a TxD, Tx)
4	DTR	Data Terminal Ready
5	SGND	Ground
6	DSR	Data Set Ready
7	RTS	Request To Send
8	CTS	Clear To Send
9	RI	Ring Indicator

DB9 Male and Female



View looking into male connector



View looking into female connector

RS-232 Signal Descriptions

The interface transfers data between the computer and the modem via the TD and RD lines. The other signals are essentially used for FLOW CONTROL, in that they either grant or deny requests for the transfer of information between a DTE and a DCE. Data cannot be transferred unless the appropriate flow control line are first asserted.

The interface can send data either way(DTE to DCE, or, DCE to DTE) independently at the same time. This is called FULL DUPLEX operation. Some systems may utilize software codes so that information may only be transmitted in one direction at a time (HALF DUPLEX), and requires software codes to switch from one direction to another (i.e., from a transmit to receive state).

The following is a list of common RS232 signals.

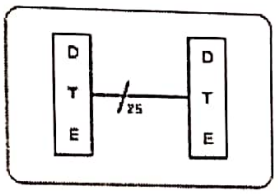
- **Request To Send (RTS)** This signal line is asserted by the computer (DTE) to inform the modem (DCE) that it wants to transmit data. If the modem decides this is okay, it will assert the CTS line. Typically, once the computer asserts RTS, it will wait for the modem to assert CTS. When CTS is asserted by the modem, the computer will begin to transmit data.
- **Clear To Send (CTS)** Asserted by the modem after receiving a RTS signal, indicating that the computer can now transmit.
- **Data Terminal Ready (DTR)** This signal line is asserted by the computer, and informs the modem that the computer is ready to receive data.
- **Data Set Ready (DSR)** This signal line is asserted by the modem in response to a DTR signal from the computer. The computer will monitor the state of this line after asserting DTR to detect if the modem is turned on.
- **Receive Signal Line Detect (RSLD)** This control line is asserted by the modem, informing the computer that it has established a physical connection to another modem. It is sometimes known as *Carrier Detect (CD)*. It would be pointless a computer transmitting information to a modem if this signal line was not asserted. If the physical connection is broken, this signal line will change state.
- **Transmit Data (TD)** is the line where the data is transmitted, a bit at a time.

- **Receive Data (RD)** is the line where data is received, a bit at a time.

CONNECTING TWO DTE DEVICES TOGETHER

Often, two DTE devices need to be connected together using a serial link. This is for file transfer or printer access. The problem is that DTE devices expect to talk directly to DCE devices, not another device of the same type. DTE's cannot generate signals like DSR and CTS, so connecting two DTE's together will result in neither getting permission to send, and thinking that the modem is off-line (by not receiving DSR).

To allow the interconnection of two DTE devices without using DCE's, a special type of cable must be used. This is called a *Null Modem Cable*, which fools the DTE into thinking that it is connected to a DCE device. In this case, modems are not used, so the connection looks like.



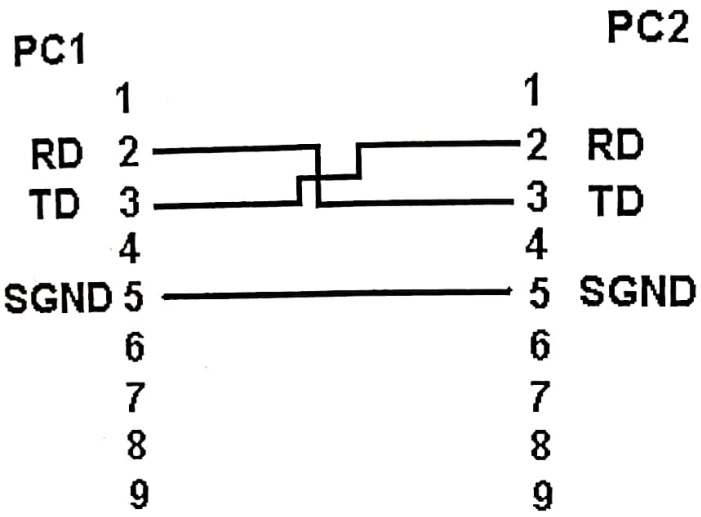
RS232 DB9 NULL Modem Pinout

Use when connecting two systems (e.g. PCs) via their DB9 interfaces without a modem (i.e. back-to-back).

DB9	Signal	DB9	Signal
2	RD	3	TD
3	TD	2	RD
4	DTR	6,1	DSR, DCD
6,1	DSR, DCD	4	DTR
7	RTS	8	CTS
8	CTS	7	RTS
5	SGND	5	SGND
9	RI	9	RI

The Bare Minimum of signals

Yes you can build a cable with only three pins connected (TD, RD and SGND) see below and it may work (typically if 'Flow control' is set to 'none' in the Control Panel at both ends).



Serial I/O

RS-232 communications (serial I/O)

Declaration:

int bioscom(int cmd, char abyte, int port);
unsigned _bios_serialcom(int cmd, int port, char abyte);

Remarks

Both bioscom and _bios_serialcom use BIOS interrupt 0x14 to perform various RS-232 communications over the I/O port given in port.

Arguments: what it is/does?

abyte- OR combination of bits that specifies COM port settings
(ignored if cmd = 2 or 3)
cmd- Specifies the I/O operation to perform
port- Identifies the I/O port; 0 = COM1, 1 = COM2, etc.

Return Value:

For all values of cmd, both functions return a 16-bit integer.

The upper 8 bits of the return value are status bits.

- If one or more status bits is set to 1, an error has occured.
- If no status bits are set to 1, the byte was received without error.

The lower 8 bits of the return value depend on the value of cmd specified:

Value of cmd	Lower 8 bits of return value
0 (_COM_INIT) or 3 (_COM_STATUS)	The lower bits are defined as shown in the preceding diagram.
1 (_COM_SEND)	...
2 (_COM_RECEIVE)	The byte read is in the lower bits of the return value--if there is no error (no upper bits are set to 1).

Program

```
#include<iostream.h>
#include<bios.h>
#include<conio.h>
#define COM1 0
#define DATA_READY 0X100
#define SETTINGS (0x80|0x02|0x00|0x00)
int main(void)
{
int in, out, status;
clrscr();
bioscom(0,SETTINGS,COM1);
cprintf("...Bioscom[Esc] to exit...");
cprintf("\nData sent to you:");;
while(1)
{
status=bioscom(3,0,COM1);
cout<<status<<endl;
if(status & DATA_READY )
if((out=bioscom(2,0,COM1) & 0x7F)!=0)
putch(out);
//if(kbhit())
if(1)
{
cprintf("Enter Data to sent:");
if((in=getch())==27)
break;
//cout<<"\n";
putch(in);
bioscom(1,in,COM1);
cout<<"\ndata sent\n";
}
}
return 0;
}
```

```
#include<stdio.h>
#include<iostream.h>
#include<bios.h>
#include<graphics.h>
#include<stdlib.h>
#include<conio.h>
#define COM1 0
#define DATA_READY 0X100
#define SETTINGS (0x80|0x02|0x00|0x00)
void clr_scr()
{
clrscr();
 /* request auto detection */
int gdriver = DETECT, gmode, errorcode;

/* initialize graphics mode */
initgraph(&gdriver, &gmode, "C:\\TC\BGI");

/* read result of initialization */
errorcode = graphresult();

if (errorcode != grOk)  /* an error occurred */
{
  printf("Graphics error: %s\n", grapherrormsg(errorcode));
  printf("Press any key to halt:");
  getch();
  exit(1);             /* return with error code */
}
}
void chat_window()
{
setcolor(2);
settextstyle(10,0,2);
outtextxy(95,12,"...Bioscom[Esc] to exit...");
setcolor(1);
rectangle(95,95,535,230);
rectangle(98,125,532,227);
setcolor(6);
settextstyle(7,0,2);
```

```
outtextxy(215,95,"Transmitted Data");
gotoxy(15,9);printf(">");
gotoxy(15,10);printf(">");
gotoxy(15,11);printf(">");
gotoxy(15,12);printf(">");
gotoxy(15,13);printf(">");
gotoxy(14,14);printf("--->");
setcolor(1);
rectangle(95,287,535,422);
rectangle(98,317,532,419);
setcolor(6);
settextstyle(7,0,2);
outtextxy(240,287,"Received Data");
gotoxy(15,21);printf(">");
gotoxy(15,22);printf(">");
gotoxy(15,23);printf(">");
gotoxy(15,24);printf(">");
gotoxy(15,25);printf(">");
gotoxy(14,26);printf("--->");
}
int main()
{
clr_scr();
char transmit[5][51]={'\0'};
char receive[5][51]={'\0'};
char blank[51]={"                    "};
int in, out, status,i,j,t,r;
i=j=t=r=0;
chat_window();
bioscom(0,SETTINGS,COM1);
while(in!=27)
{
/*******Data Receiving Block******/
status=bioscom(3,0,COM1);
if(status & DATA_READY )
if((out=bioscom(2,0,COM1) & 0x7F)!=0)
if(r<50 && out!='\r')
{
if(out==8 && r>0)
{
```

```
r--;
gotoxy(16+r,26);
putchar(' ');
}
else
{
gotoxy(16+r,26);
putchar(out);
receive[4-j][r++]=char(out);
}
}
else
{
receive[4-j][r]='\0';
r=0;
gotoxy(16,21);printf("%s",blank);
gotoxy(16,21);
printf("%s",receive[(8-j)%5]);
gotoxy(16,22);printf("%s",blank);
gotoxy(16,22);
printf("%s",receive[(7-j)%5]);
gotoxy(16,23);printf("%s",blank);
gotoxy(16,23);
printf("%s",receive[(6-j)%5]);
gotoxy(16,24);printf("%s",blank);
gotoxy(16,24);
printf("%s",receive[(5-j)%5]);
gotoxy(16,25);printf("%s",blank);
gotoxy(16,25);
printf("%s",receive[(4-j)%5]);
gotoxy(16,26);printf("%s",blank);
j=(j+1)%5;
}
if(kbhit())
{
in=getch();
bioscom(1,in,COM1);
if(t<50 && in!='\r' && in!=27)
{
if(in==8 && t>0)
```

```
{
t--;
gotoxy(16+t,14);
putchar(' ');
}
else
{
gotoxy(16+t,14);
putchar(in);
transmit[4-i][t++]=char(in);
}
}
else if (in!=27)
{
transmit[4-i][t]='\0';
t=0;
gotoxy(16,9);printf("%s",blank);
gotoxy(16,9);
printf("%s",transmit[(8-i)%5]);
gotoxy(16,10);printf("%s",blank);
gotoxy(16,10);
printf("%s",transmit[(7-i)%5]);
gotoxy(16,11);printf("%s",blank);
gotoxy(16,11);
printf("%s",transmit[(6-i)%5]);
gotoxy(16,12);printf("%s",blank);
gotoxy(16,12);
printf("%s",transmit[(5-i)%5]);
gotoxy(16,13);printf("%s",blank);
gotoxy(16,13);
printf("%s",transmit[(4-i)%5]);
gotoxy(16,14);printf("%s",blank);
i=(i+1)%5;
}
}
}
/*clean up*/
closegraph();
return 0;
}
```