Versalent

USBM232 Mouse-Serial Converter Manual

Version 2.04

Revised Jan 24, 2025

General Description:

The Versalent USBM232 is a small adapter which converts a standard USB *wired* mouse to an RS232 mouse for use with non-USB systems. RS232 messages are generated periodically so the mouse position and button states can be tracked by computer systems which do not provide a USB interface. There is no single RS232 mouse standard for representing mouse movements. Microsoft, Logitech, Sun, and MouseSystems all had slightly different standards when serial mice were prevalent. They supported different numbers of buttons, and typically no scroll wheels. USBM232 offers selectable output data formats including the original Microsoft Serial Mouse Format (no scroll wheel), a Custom Format which extends the Microsoft format with scroll wheel data, and a unique Versalent ASCII Format which consists entirely of ASCII viewable characters.

Normal mouse usage requires some type of graphic interface which allows the operator to see a mouse cursor so that appropriate actions can be taken when the mouse is in particular positions, and the USBM232 provides the incremental position data needed to navigate the cursor on a non-USB system screen. The USBM232 operates with ***wired mice only***. If a mouse will operate with a standard PC without loading any special drivers, it will operate with the USBM232. Some mice offer more functions than the standard two-button mouse, however USBM232 cannot provide any of those advanced features since it cannot load unique drivers. It provides information on X-Y movement, left and right button clicks, and scroll wheel rotation. Future versions may add operation with composite devices (mouse/keyboard combos) which will allow for wireless mice.

When the USBM232 is powered on the GREEN LED flashes very quickly until it finds a USB mouse – then flashes noticeably slower. It then begins sending position messages as it detects mouse movements.

Microsoft still includes its original serial mouse driver in it's latest operating systems (up to WIN10) which means a standard Windows computer can still use a serial mouse -- or a USB mouse with the USBM232. This driver requires that when the serial Request-To-Send signal is toggled, the mouse responds with a 1200 Baud/No Parity 'M' character, so the USBM implements this RTS feature in MS Mouse mode. **If you use MS Mouse mode, the RTS signal MUST be driven by your host's serial port, or if unused must tied to an inactive low level (0v to -25v)**. The USBM232 does tie this signal weakly to ground, however if a long cable/wire is connected to the USBM232's DB9 pin and is left unconnected, cable crosstalk can randomly inject noise that acts like an RTS signal disrupting mouse communications.

Alternatively the USBM232 can be ordered with the 'RR5' option which disconnects the RTS signal from the DB9 connector. MS Mouse mode will still operate however the auto-find feature of the Windows driver will not operate so the mouse will not be automatically detected by a Windows/DOS computer.

If your WIN10 or greater computer fails to communicate with a serial mouse, it may be necessary to issue the following command 'sc config sermouse start=demand' in a DOS command-line window with administrator privileges.

Models Available:

To accommodate various power options and RS232 configurations, the USBM232 is offered in the following models:

Model#	RS232 Type	Power Input	Defining Features
USBM232-014	DCE	+5VDC	Power input on DB9 Pin 4
USBM232-016	DCE	+5VDC	Power input on DB9 Pin 6
USBM232-024	DCE	6-12VDC	Power input on DB9 Pin 4, or power jack/ext wall brick
USBM232-026	DCE	6-12VDC	Power input on DB9 Pin 6, or power jack/ext wall brick
USBM232-034	DTE	+5VDC	Power input on DB9 Pin 4
USBM232-036	DTE	+5VDC	Power input on DB9 Pin 6
USBM232-044	DTE	6-12VDC	Power input on DB9 Pin 4, or power jack/ext wall brick
USBM232-046	DTE	6-12VDC	Power input on DB9 Pin 6, or power jack/ext wall brick
USBM232-124	DCE	4-30VDC	Power input on DB9 Pin 4, or power jack/ext wall brick
USBM232-126	DCE	4-30VDC	Power input on DB9 Pin 6, or power jack/ext wall brick
USBM232-144	DTE	4-30VDC	Power input on DB9 Pin 4, or power jack/ext wall brick
USBM232-146	DTE	4-30VDC	Power input on DB9 Pin 6, or power jack/ext wall brick

The two following optional suffixes add these additional features:

- RR5 -- this suffix disables the RTS signal and is intended for use with devices NOT operating the Microsoft Serial Protocol. MS protocol uses RTS to auto-identify the COM port to which a serial mouse is connected .. and is generally required for that protocol. If the user configures the USBM232 to operate a different protocol, it is possible that an unused/undriven RTS signal can inject noise and interfere with USBM232 operation.
- E -- this suffix adds both an E-baud feature, and a 'command-protection' feature. E-baud provides a command which allows the user to change the unit baud rate without opening the case .. instead of changing the internal baud jumpers. Command protection adds a key-string that must precede each single-character command. This feature prevents inadvertent configuration changes which can occur during host startup for instance, when the host may generate a random serial character which could alter the desired operating mode or sensitivity.

These suffixes/features can be added individually or in combination.

<u>USBM232 Configuration Application:</u>

A Windows USBM232 Configuration utility is available for download at www.versalent.biz/dl.htm which allows you to configure and test a USBM232 device. It allows you to execute the command set below for unit setup and testing.

<u>USBM232 Output Message Formats:</u>

There are three selectable formats for the serial output of mouse data. Refer to the command set below to set the desired format. Once set, this value is retained in non-volatile storage so the device powers up and/or resets to this mode.

1) **<u>Microsoft Serial Mouse Format</u>** -- this is a binary format (meaning that some characters are not printable/viewable). It is the most efficient of the three available formats in that each message contains only 3 characters (bytes) of data. Because of the small message size it is most suitable for low baud rate applications, although when it was devised, mice did not have scroll wheels so it contains no field to transfer that information. This format transfers data for X-Y movement and L/R mouse buttons only. The details of the data format are contained below in the Command Set description that follows.

2) **<u>Custom Binary Mode</u>** -- this binary mode is similar to the Microsoft Serial Mouse Format, except that it adds another byte for the scroll wheel and the message length is therefore 4 characters (1 more than the Microsoft format above). This is a unique format created by Versalent to maintain the transfer efficiency of the Microsoft format while adding scroll wheel data. Characters are not printable/viewable but it allows for efficient transfer of X-Y movement, L/R mouse buttons, and scroll wheel movement. The details of this data format are contained in the Command Set description that follows.

3) **<u>ASCII Mode</u>** -- this unique Versalent ASCII mode format is the least efficient of the 3 user modes, however provides for human-readable messages. It uses 10 characters per movement message and displays neatly formatted messages on a terminal-emulator screen. It can be used for diagnostics and validation, or for applications that can provide higher baud rates (typically > 9600). Data analysis by the receiving application is easier since X-Y data and button data is not mixed across bytes as it is in the two binary formats. The receiver's serial buffer may have to be larger than for the binary modes since there are significantly more characters transferred. The details of this data format are contained in the Command Set description that follows.

XON/XOFF Data Management:

The USB232 will only accept an XOFF if there is currently a USB mouse connected, and enumerated. This means that the LED flash rate has slowed to its lower rate of about 1 flash per second with equal on and off times. Once the XOFF is received, the flash ***rate*** remains the same, however the led on time becomes very short so the light pulse changes noticeably. This generally will not be seen because the duration of an XOFF state is typically very short .. the host normally sends an XON following an XOFF within a fraction of a second to resume communications. However if the host leaves the USBM232 in the XOFF state for an extended period, this change in flash pattern will be evident. The XON character is 0x11 also referred to as DC1, and XOFF is 0x13 also referred to as DC3.

USBM232 provides a fail-safe communications re-start timeout of 3 seconds .. if the USBM232 has been left in the XOFF state for longer than this, it will automatically negate the XOFF command and resume sending position messages. This prevents the USBM232 from unintentionally being disabled for more than a short period.

Power and LED Operation:

When the USBM232 is powered on either by providing power through the DB9 connector, or with the optional wall-brick, the GREEN LED flashes very quickly with no USB mouse attached. The flash rate slows to 1 flash per second once it recognizes an attached mouse. This visual indication provides a confirmation that the mouse was detected and initialized (or 'enumerated' in USB terms). It issues no power-up messages to the RS232 host and begins sending mouse position messages in the selected output format. This allows the fastest and most seamless return to normal operation from a power interruption, or other USBM232 reset. The host can at any time confirm its connection to a USBM232 by sending the 'V' command – USBM232 returns a firmware version (ASCII) string.

Command set:

USBM232 executes the following command set. These commands and responses are all ASCII characters and can be executed at any time during operation. **Non-E version** device commands are shown and are all single character commands. However, it is not uncommon for a host to generate a few random RS232 characters at power-up or on reset and these could potentially corrupt a USBM232 setting. E-version devices are available to address this providing more robust operation, as well as offering electrically programmable baud parameters (eliminating the need to open the case).

E-version devices add an 8-character 'key' to precede all commands and a 0x0d (carriage return) character as a command terminator. It is very unlikely that the key will be generated inadvertently so settings are protected. The key is the fixed 8-charactger string "/USBM232" .

Examples: Non-E Version command **V** (to retrieve firmware version)

E Version command **/USBM232V{CR}**

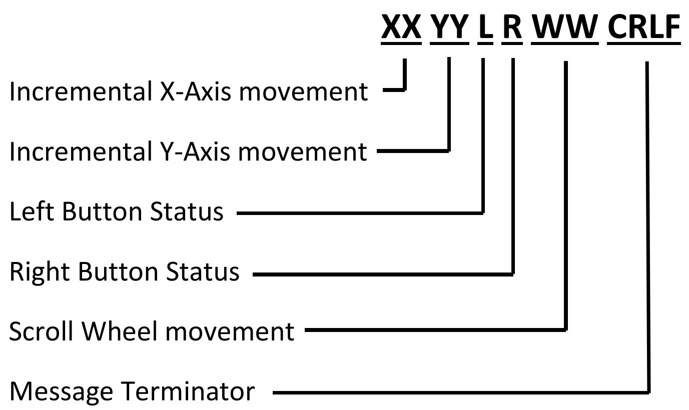
The list below describes non-E version commands. E-version commands are identical except that as above, they must be immediately preceded with the 'key' string, and are terminated with a {CR] character – except as noted in each command description. (key string = "/USBM232")

1) **V** -- Return the firmware version number. The USBM232 flushes any mouse position data in its buffer and returns "USBM232vX.XX" (or "USBM232vX.XXE" for E version devices) where X.XX is the firmware version number i.e. 1.08 .

2) **A** -- Set the output mode to Versalent ASCII mode. This provides a more human readable output format which can be viewed directly with a terminal emulation program. Instead of 3 bytes of binary data this format uses 10 bytes of hexadecimal formatted digits. It is less efficient and requires higher baud rates to prevent significant transmission latency in the mouse system. When this command is received the USBM232 flushes its output buffer, saves the setting in non-volatile memory and responds with ASCII [ACK] if successful or [NAK} if unsuccessful. The [ACK] will be delayed by the ~20ms required to write to non-volatile memory. Mouse data generation and transmission then resumes.

ASCII Format Description:

For each incremental mouse movement or click, a short RS232 (ASCII) message is output which has the following format:



XX represents two characters which describe the x-axis incremental movement since the last message. These two ASCII hexadecimal digits form a ***signed*** (two's complement) number ranging from "00"   to "7F" for forward x-axis motion, and "FF" down to "80" for reverse x-axis motion. So a single increment of motion in the positive direction is represented by "01" and a single increment of motion in the reverse direction is represented by "FF".

YY represents two characters which describe the y-axis incremental movement since the last message. These two ASCII hexadecimal digits form a ***signed*** (two's complement) number ranging from "00"   to "7F" for forward y-axis motion, and "FF" down to "80" for reverse y-axis motion. . Again a single increment of motion in the positive direction is represented by "01" and a single increment of motion in the reverse direction is represented by "FF".

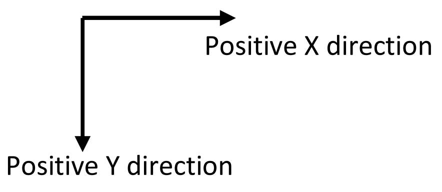
L represents a single character indicating the current state of the Left Mouse Button . When it is pressed L= "1" and when released this value is "0". A press or release of the button (with or without any x/y movement) will cause a new RS232 message to be generated.

R represents a single character indicating the current state of the Right Mouse Button . When it is pressed R = "1" and when released this value is "0". A press or release of the button (with or without any x/y movement) will cause a new RS232 message to be generated.

WW represents two characters which describe the incremental scroll wheel movement since the last message. These two ASCII hexadecimal digits form a ***signed*** (two's complement) number ranging from "00"   to "7F" for forward y-axis motion, and "FF" down to "80" for reverse y-axis motion.

CRLF represents the standard RS232 Carriage Return and Line Feed Characters which mark the end of the message.

The sign of the mouse motion is given by the mouse standard below:



Download sample C# message parsing code at
https://www.versalent.biz/downloads/usbmsamplecode.txt

ASCII Format Messaging Rate:

This message format provides human readability at the expense of message size over the binary formats. Quick mouse movements can transfer a lot of characters – fast movements can generate approximately:

(10 chars/message) X (250 messages/0.25 second) = 2500 characters per ¼ second.

Since each movement message consists of 10 characters, it is recommended that the USBM232 be operated at a minimum of 9600 baud corresponding to a message transmission time of approximately 10ms -- higher rates are recommended to minimize transmission latencies and keep the mouse responsive. The USBM232 supports up to 115k baud and is shipped with a default 9600k baud setting.

The above numbers are typical for a fast mouse movement of approximately one foot, so the host should have a receive buffer of 2k characters or more. Even this may not be enough if the host does not process these received messages quickly, or it pauses processing while performing other tasks. To help prevent the host from being overrun, the USBM232 implements the XON/XOFF protocol which allows the host to stop the USBM232 from transmitting new messages if it is in danger of being overrun. When the host sends the RS232 XOFF character, the USBM232 immediately stops sending messages until it receives an XON character. If the mouse continues to move and generate data, that data is discarded.

3) **B** -- Set the output mode to Microsoft Serial Mouse Format. When received the USBM232 flushes its output buffer, saves the setting in non-volatile memory and responds with ASCII [ACK] if successful or [NAK} if unsuccessful. . The response may be delayed by ~ 20ms as required to erase/re-write flash memory. In this binary format mode only 3 bytes are sent per mouse movement as shown below. Notice that no scroll wheel data is available in this format. This format is very efficient can be very useful for low baud rate systems. **

	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

```
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.
```

Note that these are 8-bit characters with the upper bit always 1. And as these messages can be packed, the host can synchronize itself and identify the first byte/character of a message by D6 = '1' in the first byte of a message.

Special Notes for Win98/DOS and other old Microsoft-Driver Users:

This mode is compatible with the Microsoft drivers, however to use those drivers you MUST configure USBM232 to operate at 1200 baud/No Parity even if your COM port shows that it is set to some other speed. When the mouse is detected and the driver 'takes over' the COM port it internally changes the baud rate to 1200 but in some cases does NOT update the System Device Manager to show this change. Online documents describe the MS Serial protocol as using 7-bit characters (bits 0-6) which infers that bit 7 should be inactive (set to 0).. however as shown above the protocol actually expects that Bit 7 of each character = 1.. This mode can be used at faster baud rates with a user-provided driver however the standard MS driver only operates at 1200 baud regardless of the setting of the COM port.

4) **C** -- Set the output mode to Custom Binary Mode. When received the USBM232 flushes its output buffer, saves the setting in non-volatile memory and responds with ASCII [ACK] if successful or [NAK} if unsuccessful. This binary mode is similar to the Microsoft Serial Mouse format, except that it adds another byte for the scroll wheel. The message length is therefore 4 bytes. The response may be delayed by ~ 20ms as required to erase/re-write flash memory. Mouse data generation and transmission then resumes. . This format can be very useful for low baud rate systems. **

	D7	D6	D5	D4	D3	D2	D1	D0
1st byte	1	0	LB	RB	Y7	Y6	X7	X6
2nd byte	0	S7	X5	X4	X3	X2	X1	X0
3rd byte	0	S6	Y5	Y4	Y3	Y2	Y1	Y0
4th byte	0	0	S5	S4	S3	S2	S1	S0

```
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.
```
S0-7 is the movement of the scroll wheel. Positive is a roll forward, negative is a roll back.

The user can decide whether to add this movement to the x-axis, or y-axis position. Note that the first byte of each message has D7=1 and can be used by the receiver to synchronize itself to the message stream.

5) **[key]Dxy{CR}** – Set the baud rate electronically. This command is ***ONLY available in E-version*** devices so as shown the command character 'D' is preceded by the key and followed by two ASCII parameters and ends with a {CR}:

Param	Baud Rate	Param	Parity
x = ‘0’	Use internal jumper settings	y=’0,1 or 2’	ignored
x = ‘1’	600		
x = ‘2’	1200		
x = ‘3’	4800	y=’0’	ODD
x = ‘4’	9600	y=’1’	NONE
x = ‘5’	19.2k	y=’2’	EVEN
x = ‘6’	38.4k		
x = ‘7’	57.6k		
x = ‘8’	115.2k		

The USBM232 stores this setting in non-volatile memory and responds with an {ACK} issued at the initial baud rate on success or a [NAK} on failure. It then changes the baud/parity, flushes its serial buffers and resumes operation with no reset or power-cycle required. It will then power up with these serial settings until changed.

6) **M** -- Return the current (ASCII/Binary/Custom) mode of the USBM232. . When received, the USBM232 stops transmitting, flushes any mouse data in its output buffer, and responds with an ‘A’ , ‘B’ , or ‘C’ character indicating whether it is in ASCII mode, Microsoft Binary mode, or Custom Binary mode.

7) **S** -- Stop generating mouse position data (temporarily). When received, the USBM232 flushes any mouse data queued for transmission and does not generate any more mouse data. There is no command response. The USBM232 will respond to other commands like (V)ersion, and (M)ode requests etc which return responses but mouse data is temporarily suspended. For safety, the STOP state is exited automatically after 3 seconds, or immediately after receipt of the Resume command. This command can be used to stop mouse-position data during command interactions with USBM232.

8) **R** – Resume generating mouse data after an (S)top command was issued. If no Stop was issued, or a previous Stop has timed out, this command has no effect. The USBM232 issues an [ACK] if successful or [NAK] on failure.

9) **Yx** – This command allows the user to change or retrieve the mouse sensitivity The USBM232 responds with an ASCII [ACK]. Serial mice were used when screen resolutions were low compared to newer hi-res screens. So it is not surprising that newer USB mice have a different motion sensitivity. The x character in the command can range from ‘0’ to ‘9’. ‘1’ sets the USBM232 to the lowest sensitivity (slowest mouse speed) and ‘9’ sets it to the highest sensitivity. X= ‘0’ causes the USBM232 to return the current sensitivity (one character ‘1’-‘9’) . x > 0 sets a new sensitivity level

and returns an [ACK]. This value is stored in internal non-volatile memory so the unit powers up at this sensitivity until changed.

The following commands <u>never</u> use a key, or {CR} terminator and there are no ACK/NAK responses issued

10) **[XON]** -- This is the single ASCII character (0x11) which re-enables communications after a previous [XOFF] command had paused communications to prevent host overrun. USBM232 resumes outputting serial characters. An XON command received when XOFF is not in effect is ignored.

11) **[XOFF]** – This is the single ASCII character (0x13) which temporarily disables USBM232 serial output to prevent host overrun USBM232 remains in this disabled state awaiting an XON command to resume communications. If no XON is received within 5 seconds, USBM232 automatically resumes communications.

12) **RTS (Request To Send)** signal. This is not an ASCII character command, but an R232-signal-line command provided for compatibility with older operating system drivers. When RTS is activated, USBM232 flushes its data, switches to 1200 BAUD and responds with a single 'M' character. This scheme was used by the original serial mice as a means of finding which port the mouse was connected. This can occur at any time during operation and not just at power-up – so if the host computer restarts the USBM232 will respond to this port-identification (without being power-cycled). ***Special Note: The RTS signal is terminated lightly onboard, so if it is not used for auto-identification it can be ignored. However if your signal cable connects to RTS, but your system does not drive it or ground it, the floating signal line can pickup and inject noise that activates RTS auto-identification randomly and corrupts mouse data. You can either ground RTS at the end of the cable, or order the RR5 option (i.e. USBM232-024RR5) which disables RTS onboard.***

<u>Mouse Responsiveness:</u>

Transferring mouse data through a low-baud serial channel requires balancing message count and mouse responsiveness. Serial messages require processing so they should be minimized, but they need to be transferred frequently enough that the on-screen cursor can track mouse movements closely. Long ago Microsoft determined that a 25ms reporting period could track a mouse well enough to keep up with human perception so mice were designed to aggregate mouse movements and report on this interval. USBM232 mimics this aggregation period which is especially important for USB mice which are faster. USBM232 uses a 10ms aggregation period for baud rates faster than 1200 to provide an even more responsive mouse yet still limit the number of messages required for good tracking.

The regularly occurring 25/10ms periods are shortened by either a mouse click (which requires reporting immediately) or by very fast mouse movements which threaten to overflow the signed 8-bit X-Y movement values. (See the messaging formats above). So the reporting rate is variable, even down to zero -- if there is no mouse activity there are no messages, and the host wastes no time processing a 'quiet' mouse.

Mouse sensitivity also affects perceived responsiveness – the number of mouse 'counts' delivered for a given motion. USBM232 provides a sensitivity setting to allow the user to adjust the speed of the perceived motion similar to the setting available in the Windows Control Panel. This setting extends the Control Panel setting, or If a Windows driver is not operating the USBM232, this setting provides the needed speed/sensitivity control. Sensitivities can be set from 1 to 9 (lowest to highest) with the factory default value set to 4. Note that changing the sensitivity does not generally affect mouse resolution because small motions of a few counts are not altered by the sensitivity algorithms.

Reliability Features:

USBM232 implements an internal watchdog timer, and an internal brown-out detector. Should its microcontroller get disrupted through static discharge or other temporary interference, the watchdog will automatically reset the unit so that normal operation resumes with no user intervention. Or should power droop below an operational threshold the brown-out detector will suspend operation until power is restored to a normal level. At that point it will resume operation from a reset state again with no user intervention. Resumption of operation following any kind of reset occurs in approximately 1 second.

Baud Rate/Parity Control:

Baud and parity are set using the 5 internal shorting jumpers as shown below. (E-version devices can be set electronically instead). To open the snap-together plastic case, there are 2 'screw-driver slots' on each side of the case at the seam. Gently pry the case open using a small flat-blade screwdriver at any one of these slots with a little inward pressure while twisting the blade. After the first internal latch releases, the others release even easier and the two halves of the case separate.

USBM232 can be set to any of the baud rates/parity settings shown in TABLE 1. To configure it the case is opened, and 5 internal shunt blocks are moved to the positions shown. Note that the 3 shunt blocks on the left control the baud rate and the right two select parity. The table shows shaded blocks where shunts are to be installed to select the specified settings. The unit will them power-up with these settings. All baud settings implement 1 stop bit.

An E-Baud setting with x='0' causes the USBM232 to use the jumper settings for its serial settings ('compatibility' mode). Any other x value sets the baud rate as detailed in the preceding table.

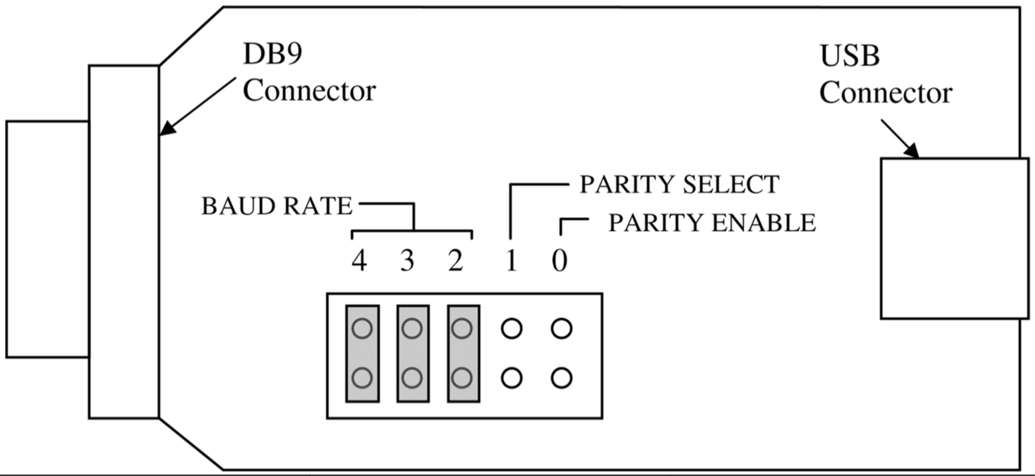
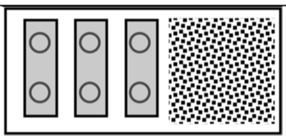
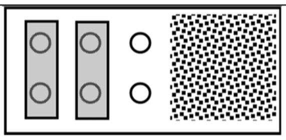
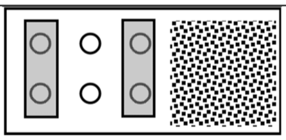
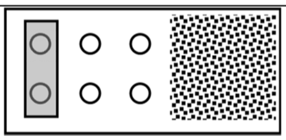
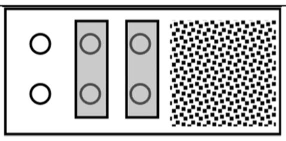
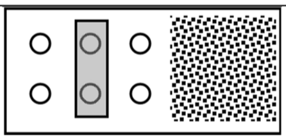
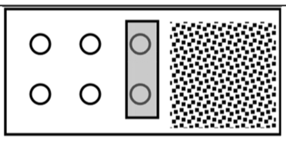
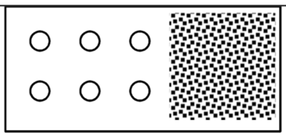


DIAGRAM	BAUD RATE	PARITY
	600 bps	See below
	1200 bps	See below
	4800 bps	See below
	9600 bps	See below
	19.2k bps	See below
	38.4k bps	See below
	57.6k bps	See below
	115k bps	See below

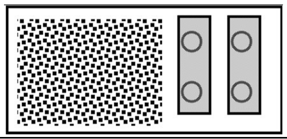
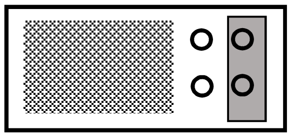
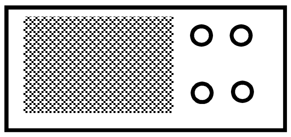
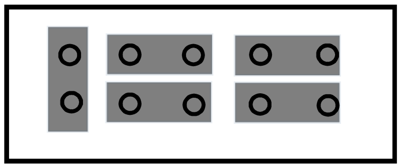
	See above	Parity Enabled, ODD Parity
	See above	Parity Enabled, EVEN Parity
	See above	Parity Disabled

TABLE 1

.

Note that any shunt blocks in a horizontal orientation have no effect on serial settings and are typically used for storing the shunts. USBM232 devices are shipped in the following default configuration .. 9600 baud, no parity. (All the horizontal blocks have no effect; the only effective one is the vertical one in the middle – 9600 setting shown.)



Electrical Parameters:

Absolute Maximum Input Voltage 5V models: -014,-016,-034,-036 6-12V models: -024,-026,-044,-046 4-30V models: -124,-126,-144,-146	 +5.25VDC (35mA no mouse attached) +15VDC (40mA no mouse attached) +31VDC (90mA @4V, 15mA @ 30V typical)
Nominal Voltage Input:	+5VDC , 6-12VDC, 4-30VDC
Baud Rate:	600 to 115k selectable via shorting blocks
Parity:	Selectable ODD/EVEN/NONE
DB9 Connector Configuration:	DTE or DCE (per model number)
Handshake Supported	XON/XOFF

Environmental:

Max Operating Temperature:	+70^{o}C
Min Operating Temperature:	0^{o}C
Max Storage Temperature:	+80^{o}C
Min Storage Temperature:	-20^{o}C
Humidity:	Non-condensing at all temperatures

Physical:

Size:	2.9” X 1.7” X 0.8”
Weight:	1.4 oz
Mouse Connector:	Standard USB Type A
Host Connector:	Standard DB9 (female)
Case Color:	Black (Ivory available special order)
Power Connector:	5.5 mm X 2.1 mm (male, Ctr Positive)

Document Revision Record:

Revision #	Revision Date	Description
V1.00	May 14, 2020	Modified from the Preliminary manual
V1.01	May 17, 2020	Add binary output modes
V1.02	May 20, 2020	Add hyperlink to sample code
V1.03	Dec 6, 2020	Add auto-detection (RTS) scheme, change default baud to 9600
V1.04	Dec 15, 2020	Change Binary Mode: characters to set bit 7=1 for all 3 characters. Add mouse-movement aggregation so the host messaging is at 40Hz (25ms) with larger movements in each message for fewer messages. Cut timer interval from 10ms to 1ms to better measure the 25ms messaging period,
V1.05	Jan 9, 2021	Add User-Sensitivity adjustment
V1.06	Feb 10, 2021	Add special note regarding floating RTS signal, and RR5 option available to disable the feature.

V1.07	Nov 20, 2021	Add note box describing RTS feature and need to connect this signal on the cable.
V1.08	Feb 20, 2021	Correct overall length from 2.6” to 2.9” under ‘PHYSICAL’
V1.09	Apr 12, 2022	Change default baud as shipped to 9600. Note that Serial MS Mouse protocol uses NO PARITY.
V1.10	May 23, 2022	Change some text regarding mouse sensitivity.
V2.00	Jul 22, 2023	Change to show description for E-version devices with e-baud
V2.01	Sep 20, 2023	Add a bit more description to command key string
V2.02	Dec 16, 2023	Change supply connector dimension from 5mm to 5.5mm
V2.03	Mar 24, 2024	Add note on issuing sc config sermouse command for Win10
V2.04	Jan 24, 2025	Update to add 4-30V models