17741 MITCHELL NORTH, IRVINE CALIFORNIA 92614 USA
PHONE: (949) 833-3355 FAX: (949) 833-0322
INTERNET: http://www.genovation.com
Email: support@genovation.com

Micropad 623 Serial Keypad

Developer Manual

For Windows 95/98, NT4, 2000, ME, XP

Manual Last Updated May 16, 2002

NOTICE

Genovation, Inc. reserves the right to revise this documentation and to make changes in content from time to time without obligation on the part of Genovation, Inc. to provide notification of such revision or changes.

GENOVATION, INC. SHALL NOT BE LIABLE FOR TECHNICAL OR EDITORIAL ERRORS OR OMISSIONS HEREIN; NOR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL OR ACCOMPANYING SOFTWARE.

For the latest version of this document, visit
http://www.genovation.com

The sample code is provided on an "as is" basis, and Genovation, Inc. expressly disclaims any or all warranties expressed or implied, including without limitation warranties of fitness for a particular purpose. In no event shall Genovation, Inc. be liable for any direct, indirect, incidental consequential damages of any kind whatsoever with respect to the use of the code.

Genovation, Inc. reserves the right, in its sole discretion and without any obligation, to make improvements to, or correct any error in any portion of the code.

Introduction

The Micropad 623 is a versatile numeric RS-232 serial keypad. Using our provided redefinition program for Windows or your own software, the keypad can be configured to operate any keyboard function that the end-user may desire.

The purpose of this manual is to explain how a software developer can communicate with the serial port(s) on a PC for the purpose of reading key-presses. All example code is provided in C, but the theories can be applied to any programming language that allows access to the Windows API.

The keypad can also be employed in non-Windows systems that have a free RS-232 serial port such as embedded applications, DOS systems, Apple Computer systems or any system that interfaces with a data terminal. Contact Genovation Inc. for more information regarding these type of applications.

Keypad Specifications:

- *Connects to any RS-232 serial port*
- *Operates at 1200 baud, 8 data bits, no parity*
- *Keys are fully debounced by firmware in the keypad*
- *Operates from voltage available on the COM port (3.5 to 15 volts)*
- *Complies with F.C.C. Class B standards. Complies with T.U.V. and CE standards for European applications*
- *Includes installation and redefinition program (Windows)*
- *Uses full travel switches manufactured by Cherry Electronics*
- *Operating force is 0.686N{79gf}*
- *Momentary contact with tactile sound*
- *Temperature range is 0 to 70 degrees C (32 to 158 degrees F)*
- *Keypad size is 15.5cm X 11.7cm X 3.8cm (6.1" X 4.6" X 1.5")*
- *Weight of keypad alone is 0.29kg (0.64 lb) Keypad with bulk pack sleeve is .37kg (0.82lb). Keypad with single ship pack is 0.42kg (0.94lb)*
- *Relative humidity 0% to 98% non-condensing*
- *Attached serial cable is 1m (60 inches) long*
- *Life 5 X 10^7 cycles*

DB-9 Pin Assignments

The device uses standard DB-9 pin assignments, ready to be connected you your host computer. In order to provide power for the keypad, it is recommended that the host software drive both DTR and RTS high (+4 to +15v).

Pin Name	DB-9	Comment
Receive Data	2	Data from PC to keypad [Note 1]
Transmit Data	3	Data from keypad to PC
DTR	4	Set high to supply power
Ground	5	Ground
RTS	7	Set high to supply power

Note 1: Although there is no data transmission from the host computer to the keypad, the keypad still requires a connection to the host TX pin (pin 2). This is to provide a negative voltage bias so that the keypad can provide a true RS-232 voltage swing on the keypad data out (pin 3).

Key Codes

The Micropad 623 outputs the following keycodes when pressed. The codes are fully debounced and will repeat if the key is held down:

Auto-Repeat Delay ……. 500 ms
Auto-Repeat Rate …….. 95 ms (~10 chars/sec)

Key	***Numlock OFF*** ***(Hexadecimal Notation)***	**Numlock ON** ***(Hexadecimal Notation)***
Esc	70	75
Tab	71	76
****	72	77
Backspace	73	78
Num Lock	79	41
/	6A	5A
*****	6B	5B
-	6C	5C
+	6D	5D
Enter	74	74
Period	6E	5E
0	6F	5F
1	60	50
2	61	51
3	62	52
4	63	53
5	64	54
6	65	55
7	66	56
8	67	57
9	68	58

Hardware Test

Before writing any software, you should test the keypad to make sure the hardware requirements are met and the pin assignments are correct. Genovation has a program called **SerialTest.exe** that can be used, but any terminal emulation software on any platform should work. Most Windows PC's come with **HyperTerm** installed. A better alternative is **Tera Term Pro** which can be downloaded from many web sites.

Connect the keypad to the host computer's COM port. Run the test application (terminal emulator program) and set it's configuration for 1200 baud, 8 data bits, no parity and 1 stop bit.

If the keypad is receiving power, pressing the NumLock key should light or extinguish the green LED. The keypad gets its power from RTS and DTR. Most terminal emulator programs will drive DTR and RTS automatically (but if you have problems you may need to check these settings and/or voltages).

Type on the keypad and confirm that ASCII characters appear in the window of your test application. The ASCII values are found in the table on page 4.

Software Overview

To read keystrokes in your application software you can access the serial port directly (using the Windows API) or you can use a DLL supplied by Genovation. In both cases the source code is available and it is expected that you will modify the supplied code to suit your requirements.

Windows API Overview

The following steps are typical for an application that wishes to read data from the keypad using the Windows API:

- `CreateFile()` – Opens the specified communications port.
- `ReadFile()` – Retrieves characters from the communications port.
- `CloseFile()` – Shuts down the communications port.

Kybd623.DLL Overview

The following steps are typical for an application that wishes to read data from the keypad using the Kybd623.DLL:

- `Kybd623_Open()` – Opens the keypad on the specified com port.
- `Kybd623_KbHit()` – Checks to see if there has been a key pressed.
- `Kybd623_Getch()` – Retrieves a key press.
- `Kybd623_Close()` – Closes the keypad on the specified com port.

The DLL itself makes use of the Windows API functions calls, and since source code is provided, you can see how this is done by examining the DLL source.

WinAPI Examples

The following code snippets demonstrate using the Windows API:

Opening the Communications Port

The first step is to open the desired com port.

```
HANDLE hComPort = NULL;                // Handle of the com port.

hComPort = CreateFile("COM1", GENERIC_READ|GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0,
NULL);

if(hComPort == INVALID_HANDLE_VALUE)
   {
   // Port open failure.
   }

// Port open success
```

The second step is to configure the port for the correct communications parameters

```
DCB           dcb;
COMMTIMEOUTS  CommTimeOuts;

SetupComm(hComPort, 128, 128);
PurgeComm(hComPort, PURGE_RXCLEAR|PURGE_TXCLEAR);

dcb.DCBlength = sizeof(DCB);
GetCommState(hComPort, &dcb);
dcb.BaudRate        = 1200;
dcb.fBinary         = TRUE;
dcb.fParity         = TRUE;
dcb.fOutxCtsFlow    = FALSE;        // No CTS output flow control
dcb.fOutxDsrFlow    = FALSE;        // No DSR output flow control
dcb.fDtrControl     = DTR_CONTROL_ENABLE;
dcb.fDsrSensitivity = FALSE;
dcb.fTXContinueOnXoff=TRUE;         // XOFF continues Tx
dcb.fOutX           = FALSE;        // No XON/XOFF out flow control
dcb.fInX            = FALSE;        // No XON/XOFF in flow control
dcb.fErrorChar      = FALSE;        // Disable error replacement
dcb.fNull           = FALSE;        // Disable null stripping
dcb.fRtsControl     = RTS_CONTROL_ENABLE;
dcb.fAbortOnError   = FALSE;
dcb.ByteSize        = 8;
dcb.Parity          = NOPARITY;
dcb.StopBits        = ONESTOPBIT;

SetCommState(hComPort, &dcb);

CommTimeOuts.ReadIntervalTimeout         = 10;    // Maximum time between read chars.
CommTimeOuts.ReadTotalTimeoutMultiplier  = 1;     // Multiplier of characters.
CommTimeOuts.ReadTotalTimeoutConstant    = 1;     // Constant in milliseconds.
CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
CommTimeOuts.WriteTotalTimeoutConstant   = 0;
SetCommTimeouts(hComPort, &CommTimeOuts);
```

Reading Key Data

Reading the key data is very simple. Later in this document there are some discussions about processing the data.

```
DWORD dwRead = 0;
unsigned char pBuf[8];

// Check for a char on the serial port.
ReadFile(hComPort, pBuf, 1, &dwRead, NULL);

if(dwRead == 1)
   {
   // Character available in pBuf[0]
   }
else
   {
   // No data available
   }
```

Closing the Communications Port

Close the com port when the application terminates.

```
// Close the communications channel.
CloseHandle(hComPort);
hComPort = NULL;
```

Kybd623.DLL Examples

The following files are provided:

Kybd623.DLL – The dynamic link library itself.
Kybd623.LIB – The library file for the linker.
Kybd623.H – The include file for your application.

Consult your compiler documentation for instructions on how to build an application with a third-party DLL.

The following functions prototypes are from Kybd623.H:

```
// Exported DLL functions
extern "C" __declspec(dllexport) int  Kybd623_Open(char *szPortName);
extern "C" __declspec(dllexport) void Kybd623_Close(void);
extern "C" __declspec(dllexport) int  Kybd623_KbHit(void);
extern "C" __declspec(dllexport) char Kybd623_Getch(void);
```

Here are the function descriptions:

```
int Kybd623_Open(char *szPortName);

Opens the specified COM port where the keypad is attached and spawns the communications
sub-thread.

Requires: szComPort - address of string identifying the port (eg. "COM1")
Returns: 0 on success, -1 on failure.
```

```
void Kybd623_Close(void);

Closes the port and kills off the communications sub-thread.

Requires - Nothing
Returns - Nothing
```

```
int Kybd623_KbHit(void);

Returns the number of bytes in the keyboard queue that have been received from the
keypad, but not extracted by the caller.

Requires - nothing
Returns  - the number of characters available in the keyboard queue
```

```
char Kybd623_Getch(void);

Removes the oldest character from the keyboard queue.  The caller should first call
Kybd623_KbHit() to make sure there is data available and then call this function once for
each character to extract.  There is no provision or notification of buffer overflow.  It
is up to the caller to make sure that the program generally extracts the keys faster than
they are received.

Requires: Nothing
Returns:  The oldest character in the keyboard queue.
```

Opening the Keypad

Simply specify the port that the keypad is connected to.

```
// Open the keyboard at the specified com port.
if(Kybd623_Open("COM1") == 0)
   {
   // Success
   }
else
   {
   // Failure
   }
```

Reading Key Data

Reading the key data is very simple. First check to see if there is a key available and then read the key (if there is one). Later in this document there are some discussions about processing the data.

```
char c;

if( Kybd623_KbHit() )            // Peek to see if there is a key.
   {
   c = Kybd623_Getch();          // There is a key, go get it.
   }
```

Closing the Keypad

Close the keypad when the application terminates.

```
Kybd623_Close();                    // Done with the keypad, clean up,
```

Processing the Key Data

The following describes how to take the received key characters and use them in an end application. The comments that follow assume that you wish to place some characters in the PC's keyboard buffer, but using a similar technique it is possible to enqueue WM_USER messages to any target application (window handle). Other inter-process communication techniques may apply depending on your host environment.

The primary function that allows for a key to be sent to the application which currently has the focus is `keybd_event()`, however some keys require the function `PostKeybdMessage().`

The keys output different codes depending on the setting of NumLock, but if you are not concerned about which key set is active, simply process both sets of codes in a similar manner.

```
char c;

if( Kybd623_KbHit() )            // Peek to see if there is a key.
   {
   c = Kybd623_Getch();          // There is a key, go get it.

   switch(c)
      {
      case 0x50: // Process '1' with NumLock ON and
      case 0x60: // process '1' with NumLock OFF.
                 break;

      case 0x51: // Process '2' with NumLock ON only.
                 break;

      case 0x61: // Process '2' with NumLock OFF only.
                 break;

      //  etc...
      }
   }
```

To enqueue a key to the keyboard queue:

```
keybd_event('a', 0, 0, 0);  // Enqueue 'a' key down.
keybd_event('a', 0, 2, 0);  // Enqueue 'a' key up.
```

For more generic message handling that does not involve the system keyboard queue:

```
// Place the keypad char in one of the params or use a unique WM_USER+xyz value for each
// character.  Target window handle must first be obtained (see FindWindow() function).
PostMessage(hMyWinHandle, WM_USER, My_wParam, My_lParam);
```

Other techniques may be applied to launch applications or control any other aspect of the host computer. Consult the Windows API or relevant documentation for your host operating system.