Blues Virtual
Host Manual

THIS DOCUMENT HAS BEEN PREPARED TO ASSIST CUSTOMERS IN USING SOFTWARE AND HARDWARE. NEWHART SYSTEMS INCORPORATED WILL NOT BE LIABLE FOR DAMAGES TO CUSTOMERS DUE TO ANY ERRORS CONTAINED IN THIS DOCUMENTATION, OR FOR DAMAGES TO CUSTOMERS RESULTING FROM THE USE OF THIS DOCUMENTATION AND ACCOMPANYING SOFTWARE AND HARDWARE.

BEFORE USING THIS PRODUCT WITH ANY LICENSED SOFTWARE, THE CUSTOMER SHOULD CAREFULLY READ ITS LICENSE AGREEMENT TO INSURE THAT IT WILL NOT VIOLATE ANY OF THE LICENSE TERMS OR CONDITIONS.

For More Information Write:

Newhart Systems Inc.

P.O. Box 348

Barneveld, N.Y. 13304 USA

Tel. 315-896-4131

Fax. 315- 896 - 4548

Web. http:\\www.newhartsystems.com

CONTENTS

Introduction

VirtualHost is a managed .net assembly that may be used to access IBM Mainframe and IBM Midrange AS400 computers by emulating an IBM terminal over a TCPIP network connection. VirtualHost is a .net toolkit consisting of a single .net dll, sample Microsoft Visual Studio projects, and associated documentation and help system.

The VirtualHost class has no end user interface and is designed to be used by third party .NET applications that requires access to IBM Mainframe and IBM Midrange computers. The VirtualHost class provides the necessary Properties, Methods, and Events to connect to a host computer, simulate operator entry, and scrape the virtual screen image.

Since VirtualHost has no end user interface, it is not tied to any particular .NET architecture. The class can be used in Windows Forms application, ASP.NET applications, and .NET services.

The VirtualHost class is written in 100% managed C# and is not mearly a .NET wrapper. The VirtualHost class may be used from any .NET language such as C#, VB.NET and J#.

Installation

Installation involves running the setup.exe program to copy the necessary files to a hard drive located on a stand-alone pc, network file share, or Microsoft Internet Information Server. The installation will copy the VirtualHost dll to the hard drive, and add the VirtualHost dll to the Global Assembly Cache (GAC). Once the dll is in the GAC, it may be used by any other .NET assembly and does not need to be copied to the target applications bin folder. Redistribution may either be done this way, or the VirtualHost dll may be copied directly to the target applications bin folder.

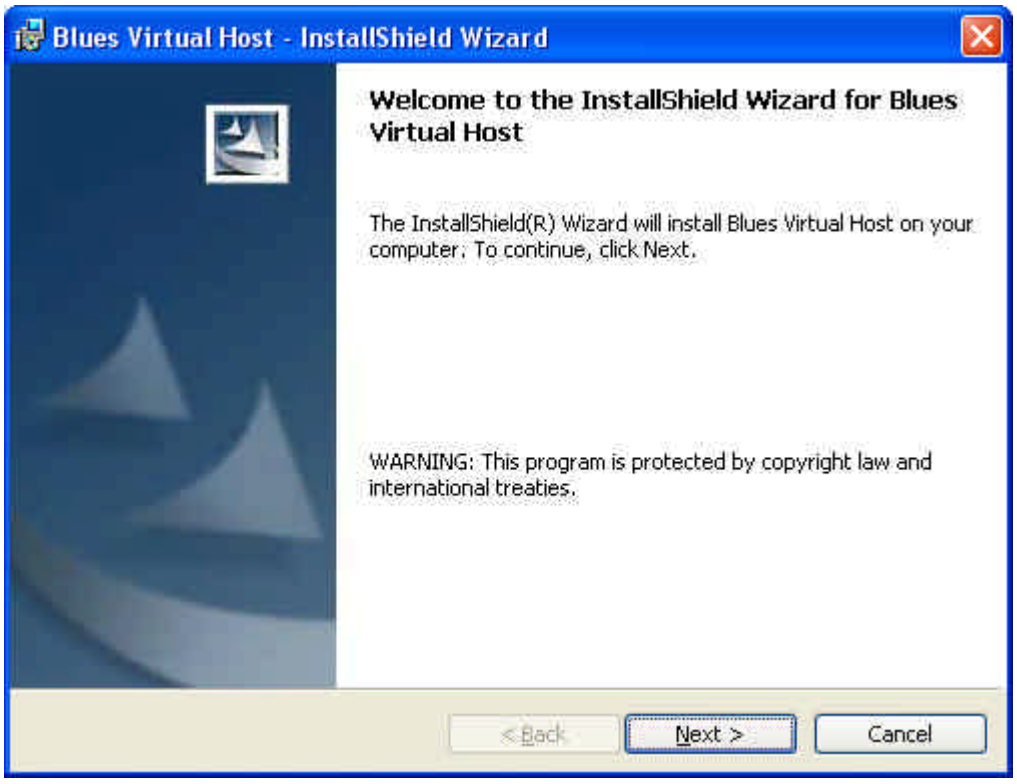
Software Requirements

To be able to install and run VirtualHost, the following software components must be previously installed:

- Microsoft Windows 98 or newer operating system.
- Microsoft .NET Framework
- Microsoft Visual Studio 2003 or higher (optional)
- Adobe reader to view the User Manual (optional)

The installation is done by performing a Start/Run from start menu of the program setup.exe . The first window will prompt for the preferred language to use for the installation.

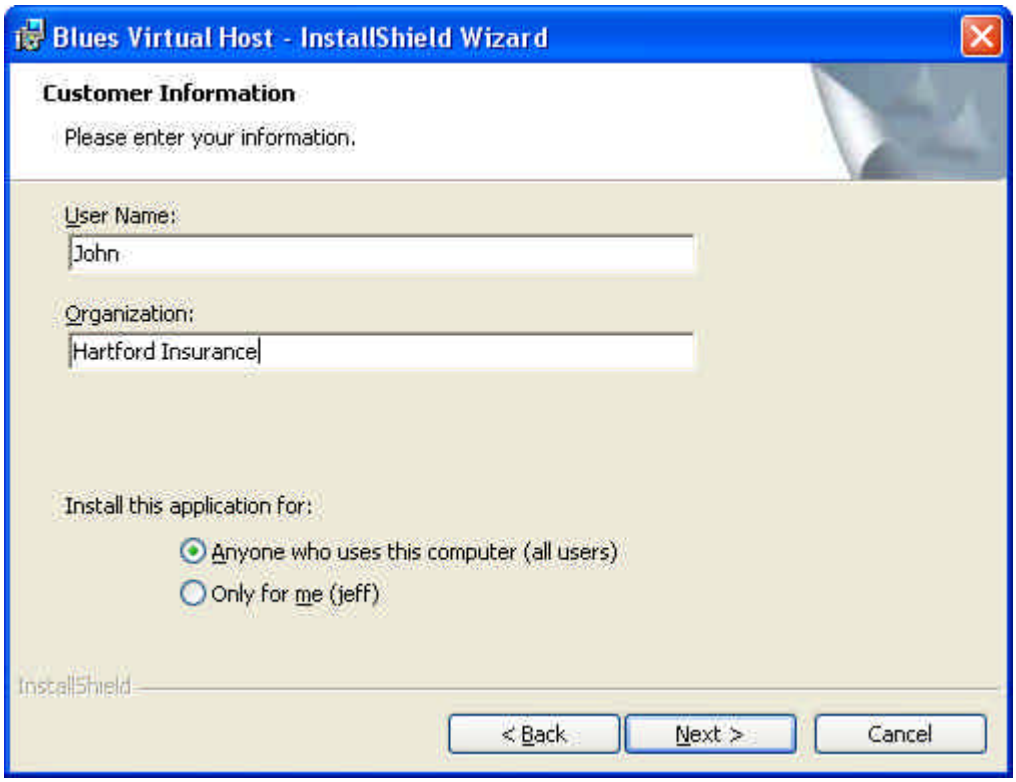
Once the installation has been started, the following window will be displayed:



License Agreement

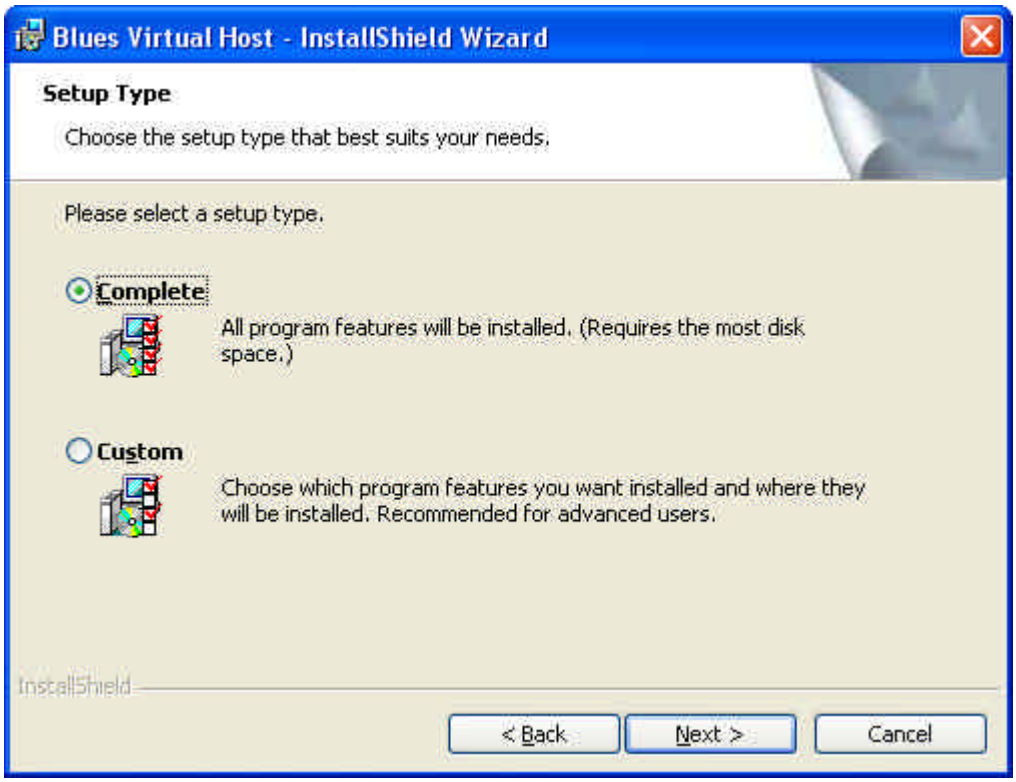
The next window displays the license agreement. Please read the license agreement before continuing. After you have accepted the license agreement, press the next button.

Selecting User Information



The software may be installed for all users of the PC, or only for yourself.

Installation Type



There are two types of installation; Complete, and Custom. Complete will install all possible components. The Custom installation will allow you to select the folder where the software will be installed and select individual features.

The software will now be copied from the installation media to your computer. After installation, the User Manual will automatically be launched.

Uninstalling

The Virtual Host software may be removed from the Control Panel “Add/Remove Programs” icon. To properly remove the software you should have administrative privileges and have all other windows closed. The uninstall wizard will guide you through removing Virtual Host from your PC.

Licensing Issues

Any third party application that uses the VirtualHost software must be properly licensed. Each PC running any part of the VirtualHost software must be licensed. For information on the software license, please read the license agreement that must be agreed to during the initial software installation process. Licensing is enforced through the use of a license file named VirtualHost.lic. The license may be either a trial license or a purchased license. The trial license is valid for 30 days from the installation date.

Packaging VirtualHost with a Third Party Application

Licensing issues still apply if the VirtualHost software is going to be packaged with any third party application. Each computer running the third party application that is using the software must be properly licensed.

Sample Projects

Opening the Sample Projects in Visual Studio

To use the sample projects, Visual Studio .NET or the .NET SDK must already be installed on the PC used for development. It is suggested that a copy of all files in the appropriate sample folder be copied to a new folder to preserve the sample for reference purposes. To build a project, first start Visual Studio .NET 2003 and use the Open Project function and browse to the project folder, and select the project to open.

The sample projects are installed under the “Program Files\Newhart Systems\Blues Virtual Host\Sample Visual Studio Projects”. The samples come in both a C# and VB version. Sample projects exist for Windows Forms applications and ASP.NET applications.

Once you are able to build and run the project with no errors, then your ready to add new functionality or add the class to your existing application.

Adding VirtualHost to an existing Visual Studio project

Important

Before adding the control to an existing application it is important to understand how the .NET runtime locates assemblies (.NET dll's) that are referenced by another application. Typically all .NET dlls that an application requires are located in either the same folder as the application, or in the .NET Global Cache (GAC). The software installation automatically places the VirtualHost dll in the GAC making it available for use by application located anyplace.

To add VirtualHost to your application from visual studio, follow these steps:

1. Right click on the "References" tree in the Solution window and select "Add Reference". Select the reference "NewhartSystems.VirtualHost" from the list and press ok.
2. C# programmers need to add a using statement to any source file using the class as follows: "`using NewhartSystems;`"
3. Construct a new VirtualHost class, set the TCPIP settings in the control, call the BeginConnect method and start using the class.

VirtualHost Class

Overview

The VirtualHost class provides the interface to a terminal emulation session. The class must be properly initialized before any host interaction can take place.

Each instance of the VirtualHost class represents an instance of a terminal emulator.

Using the class consists of creating a new instance, initializing the tcpip and device options, instructing the class to begin the connection process, waiting for screen change events, scraping the emulator screen contents, and simulating keystroke entry. The class is totally asynchronous in design. The calling class instructs the VirtualHost to perform some type of operation and an event calls the calling class back at key instances during the life of the emulator.

The simplest way to use the class is to copy and paste the few lines of code that make up the sample projects and then make the necessary changes to connect to your host computer and screen scrape your screens.

The class definition is as follows:

C#

```
private NewhartSystems.VirtualHost  Host =  new
NewhartSystems.VirtualHost("C:\\Program Files\\Newhart Systems\\Blues Virtual
Host\\Bin");
```

VB

```
Friend WithEvents Host As NewhartSystems.VirtualHost
    Host = New NewhartSystems.VirtualHost("C:\\Program Files\\Newhart
Systems\\Blues Virtual Host\\Bin")
```

Creating a new instance of the VirtualHost class requires a single argument that contains the location of the license file named VirtualHost.lic. A sample 30 day license is installed as part of the trial version of the software.

Methods

Methods are functions that the application calls to have some type of function performed. Any method that can take time to complete is asynchronous and return immediately, when the method has been performed, the associated delegate is called to notify the application of the outcome.

The following methods are listed in alphabetal order.

BeginConnect

This method starts the connection process. Before calling this method, set the event evtConnected to be notified of the result. The connection process is asynchronous. See the event evtConnected for details on determining the outcome of the call.

C#

```
Host.BeginConnect();     //Start communications and wait for
                         //screen change events
```

VB

```
Host.BeginConnect()      REM start connection process and wait for
                         REM connect complete event
```

Close

This method closes the control and frees up any resources it may be using.

C#

```
Host.Close();            //end communications and free up
                         //resources
```

VB

```
Host.Close()             REM end communications and free up
                         REM resources
```

DisconnectFromHost

This method will cause the terminal session to disconnect the tcp/ip connection.

C#

```
Host.DisconnectFromHost();    //end communications
```

VB

```
Host.DisconnectFromHost()      REM end communications
```

FindDisplayText

This method is used to search for a string on the host screen. The arguments include the string, the starting row and the starting column. The starting row and column are one based numbers. The search is exhausted at the end of the screen. If the string is found, true is returned, other false is returned. If the string is found, the found TextOnRow and founfTextOnColumn properties will contain the location of the string.

C#

```
If(Host.FindDisplayText(“signon”,1,1))    //search screen for text
{
      // inspect foundTextOnRow and foundTextOnColumn to see
      // where it was found
}
```

VB

```
If (Host.findDisplayText("signon", 1, 1)) Then
    REM Inspect foundTextOnRow and foundTextOnColumn
    REM properties
End If
```

getDisplayText

This method obtains a copy of the host screen (scrapes the screen). Any portion of the screen contents may be read. Arguments include the starting row, column, and length. All argument values are one based.

C#

```
//
// scrape screen and take appropriate action
//
string Screen = Host.getDisplayText(1,1,Host.ScreenRows *
Host.ScreenCols);
```

VB

```
REM
REM scrape screen and take appropriate action
REM
Dim Screen As String
Screen = Host.getDisplayText(1, 1, Host.ScreenRows * Host.ScreenCols)
```

keyText

This method is used to type a string of characters into the host screen. The characters are entered starting at the current cursor address.

The return code is true for success and false for failure. A False return code indicates that the either the host screen was busy receiving data from the host, or that the cursor was located in a protected location of the screen.

C#

```
Host.keyText("hello World");  //enter some text
```

VB

```
Host.keyText("hello world")  REM enter some text
```

keyTerminalKey

This method is used to send a function key to the terminal session. Function keys are special keys that cause a specific function to be performed by the terminal session.

Two prototypes of this function exist, one that takes a key code, and the other that takes a text representation of the key. The enumerator called KeyDefines contains the integer values for the key codes. The following table lists the possible values for the arguments:

Argument	**Function**
KeyDefines.DUPLICATE or “DUP” or “DUPLICATE”	3270 or 5250 Duplicate key
KeyDefines.PF1 or “PF1” or “F1”	3270 or 5250 Function key 1
KeyDefines.PF2 or “PF2” or “F2”	3270 or 5250 Function key 2
KeyDefines.PF3 or “PF3” or “F3”	3270 or 5250 Function key 3
KeyDefines.PF4 or “PF4” or “F4”	3270 or 5250 Function key 4
KeyDefines.PF5 or “PF5” or “F5”	3270 or 5250 Function key 5
KeyDefines.PF6 or “PF6” or “F6”	3270 or 5250 Function key 6
KeyDefiles.PF7 or “PF7” or “F7”	3270 or 5250 Function key 7
KeyDefines.PF8 or “PF8” or “F8”	3270 or 5250 Function key 8
KeyDefines.PF9 or “PF9” or “F9”	3270 or 5250 Function key 9
KeyDefines.PF10 or “PF10” or “F10”	3270 or 5250 Function key 10
KeyDefines.PF11 or “PF11” or “F11”	3270 or 5250 Function key 11
KeyDefines.PF12 or “PF12” or “F12”	3270 or 5250 Function key 12
KeyDefines.PF13 or “PF13” or “F13”	3270 or 5250 Function key 13
KeyDefines.PF14 or “PF14” or “F14”	3270 or 5250 Function key 14
KeyDefines.PF15 or “PF15” or “F15”	3270 or 5250 Function key 15

KeyDefines.PF16 or “PF16” or “F16”	3270 or 5250 Function key 16
KeyDefines.PF17 or “PF17” or “F17”	3270 or 5250 Function key 17
KeyDefines.PF18 or “PF18” or “F18”	3270 or 5250 Function key 18
KeyDefines.PF19 or “PF19” or “F19”	3270 or 5250 Function key 19
KeyDefines.PF20 or “PF20” or “F20”	3270 or 5250 Function key 20
KeyDefines.PF21 or “PF21” or “F21”	3270 or 5250 Function key 21
KeyDefines.PF22 or “PF22” or “F22”	3270 or 5250 Function key 22
KeyDefines.PF23 or “PF23” or “F23”	3270 or 5250 Function key 24
KeyDefines.PF24 or “PF24” or “F24”	3270 or 5250 Function key 24
KeyDefines.PA1 or “PA1”	3270 or 5250 PA 1 Function key
KeyDefines.PA2 or “PA2”	3270 or 5250 PA 2 Function key
KeyDefines.PA3 or “PA3”	3270 or 5250 PA 3 Function key
KeyDefines.ENTER or “ENTER”	3270 or 5250 ENTER Function key
KeyDefiles.SYSREQ or “SYS_REQ” or “SYSTEM REQUEST”	3270 or 5250 System Request Function key
KeyDefines.TABFWD or “TAB”	3270 or 5250 Tab forward Function key
KeyDefines.BACKTAB or “BACKTAB”	3270 or 5250 Back TabFunction key
KeyDefines.NEWLINE or “NEWLINE” or “NEW LINE”	3270 or 5250 New LineFunction key
KeyDefines.CLEAR or “CLEAR”	3270 or 5250 Clear Screen Function key
KeyDefines.ATTENTION or “ATTENTION”	3270 or 5250 Attendtion Function key
KeyDefines.CURSOR_SELECT or “CUR_SEL” or “CURSOR SELECT”	3270 or 5250 Cursor Select Function key
KeyDefines.ERASE_FIELD or “ERASE_EOF” or “ERASE EOF”	3270 or 5250 Erase to end of field Function key
KeyDefines.ERASE_INPUT or	3270 or 5250 Erase Input Function key

“ERASE_INP” or “ERASE INPUT”	
KeyDefines.FIELD_MARKER or “FLD_MARK” or “FIELD MARK”	3270 or 5250 Field Mark Function key
KeyDefines.CURHOME or “HOME”	3270 or 5250 Home Function key
KeyDefines.CURSOR_LEFT or “CUR_LEFT” or “CURSOR_LEFT”	3270 or 5250 Cursor Left Function key
KeyDefines.CURSOR_RIGHT or “CUR_RIGHT” or “CURSOR RIGHT”	3270 or 5250 Cursor Right Function key
KeyDefines.RESETor “RESET”	3270 or 5250 Reset Function key
KeyDefines.CURSOR_UP or “KEYUP” or “CURSOR UP”	3270 or 5250 Cursor Up Function key
KeyDefines.CURDOWN or “KEYDOWN” or “CURSOR DOWN”	3270 or 5250 Cursor Down Function key
KeyDefines.BACKSPACE or “BACKSPACE”	3270 or 5250 Backspace Function key
KeyDefines.INSERT or “INSERT”	3270 or 5250 Insert Function key
KeyDefines.DELCHAR or “DELETE”	3270 or 5250 Delete Function key
KeyDefines.FIELD_EXIT or “FIELD_EXIT” or “FIELD EXIT”	5250 Field Exit Function key
KeyDefines.FIELD_PLUS or “FIELD_PLUS” or “FIELD PLUS”	5250 Field Plus Function key
KeyDefines.FIELD_MINUS or “FIELD_MINUS” or “FIELD MINUS”	5250 Field Minus Function key
KeyDefines.ROLL_UP or “ROLL_UP” or “ROLL UP”	5250 Roll Up Function key
KeyDefines.ROLL_DOWN or “ROLL_DOWN” or “ROLL DOWN”	5250 Roll Down Function key
KeyDefines.PRINT_5250 or “HOST PRINT”	5250 Host Print Function key

C#

```
Host.keyTerminalKey(“ENTER”); //send enter key
```

or

```
Host.keyTerminalKey(NewhartSystems.KEYDefines.ENTER); //send enter key
```

VB

```
Host.keyTerminalKey(“ENTER”)  REM send enter key
```

or

```
Host.keyTerminalKey(NewhartSystems.KEYDefines.ENTER)  REM send enter key
```

---

setCursor

This method may be used to set the input cursor to a specific row and column for subsequent operator input or keyText method calls. The arguments are one based values.

C#

```
Host.setCursor(2,5);    //set cursor to row 2 column 5
```

VB

```
Host.setCursor(2, 5) REM set cursor to row 2 column 5
```

Properties

The following properties are listed in alphabetal order.

EmulationType

This property determines the protocol type used for the TCPIP connection. The TN3270 protocol is selected for TN3270 and TN3270E connections to

mainframe computers and TN5250 is selected for IBM AS400 connections. See the enumerator Newhart Systems.EMULATIONTYPES for valid values.

C#

```
Host.EmulationType = (int)EMULATIONTYPES.TN5250; //TN3270 or TN5250
```

VB

```
Host.EmulationType = NewhartSystems.EMULATIONTYPES.TN5250 REM select
emulation type based on your type of host computer
```

foundTextOnRow

This property contains the row that a string was found on. This property is only valid after a successful findDisplayText or waitForDisplayString method has been done. The property is 1 based and may range from 1 to the number of rows in the host session screen.

C#

```
If(Host.FindDisplayText(“signon”,1,1))    //search screen for text
{
      // inspect foundTextOnRow and foundTextOnColumn to see
      // where it was found
      if(Host.foundTextOnRow == 8 && Host.foundTextOnColumn == 10)
      {
      }
}
```

VB

```
If (Host.findDisplayText("signon", 1, 1)) Then
    REM Inspect foundTextOnRow and foundTextOnColumn
    REM properties
```

```
        If (Host.foundTextOnRow = 2 And Host.foundTextOnColumn = 10) Then

        End If

End If
```

---

foundTextOnColumn

This property contains the column that a string was found on. This property is only valid after a successful findDisplayText or waitForDisplayString method has been done. The property is 1 based and may range from 1 to the number of columns in the host session screen.

C#

```
If(Host.FindDisplayText("signon",1,1))    //search screen for text
{
      // inspect foundTextOnRow and foundTextOnColumn to see
      // where it was found
      if(Host.foundTextOnRow == 8 && Host.foundTextOnColumn == 10)
      {
      }
}
```

VB

```
If (Host.findDisplayText("signon", 1, 1)) Then
    REM Inspect foundTextOnRow and foundTextOnColumn
    REM properties

      If (Host.foundTextOnRow = 2 And Host.foundTextOnColumn = 10) Then

      End If

End If
```

HostCodePage

This property determines what code page the host computer is using to transmit and receive screens of text in. This property will attempt to default to a compatable host code page of the executing computers language settings.

This property may be set to any valid CODEPAGE value. See NewhartSystems.CODEPAGE enumeration.

C#

```
Host.HostCodePage = NewhartSystems.CODEPAGE.GERMAN;
```

VB

```
Host.HostCodePage = NewhartSystems.CODEPAGE.GERMAN
```

IPAddress

This property is used to get or set the IP address for the session to connect to. The argument is a well-formed TCP/IP address in either the form a.b.c.d, or a string that is defined in a domain name server (like “localhost”). The terminal session should not be connected at the time this method is called. An application can dynamically connect to a host if the session is not configured for the “Automatically Connect To Host” option and have the application set the ip address, ip port, and call the BeginConnect method.

C#

```
Host.IPAddress = "217.110.154.85"; //IP address of the host
```

VB

```
Host.IPAddress = "217.110.154.85" REM IP address of the host
```

IPPort

This property may be used to set the tcp/ip port number used to connect to that host. The default value is for the telnet port is 23. The terminal session should

not be connected at the time this method is called. An application can dynamically connect to a host if the session is not configured for the "Automatically Connect To Host" option and have the application set the ip address, ip port, and call the BeginConnect or ConnectToDestination method.

C#

```
Host.IPPort = 23;       //default telnet port number is 23
```

VB

```
Host.IPPort = 23        REM default telnet port number is 23
```

ModelNumber

This property determines the type of IBM terminal that is emulated. Different model types have different numbers of rows and columns. Each model type has a default screen size and an alternate screen size. Either the default or alternate screen size is active as any given instant and is controlled by the host application. This means that the screen size can change at any time. The ScreenRows and ScreenCols properties contain the current number of rows and columns being used. See the enumerator NewhartSystems.MODELTYPE for the valid values of this property.

C#

```
Host.ModelNumber = (int)MODELS.MODEL_2;   //default 24 lines by 80
characters wide
```

VB

```
Host.ModelNumber = NewhartSystems.MODELS.MODEL_2  REM select model
number, it determines the number of rows and columns
```

Message

This property contains an error message when an error occurs.

C#

```
string LastError = Host.Message;
```

VB

```
Dim LastError As String
LastError = Host.Message
```

ScreenCols

This property contains the number of columns on the host screen area.

C#

```
//
// scrape screen and take appropriate action
//
string Screen = Host.getDisplayText(1,1,Host.ScreenRows *
Host.ScreenCols);
```

VB

```
REM
REM scrape screen and take appropriate action
REM
Dim Screen As String
Screen = Host.getDisplayText(1, 1, Host.ScreenRows * Host.ScreenCols)
```

ScreenRows

This property contains the number of rows on the host screen area.

C#

```
//
// scrape screen and take appropriate action
//
string Screen = Host.getDisplayText(1,1,Host.ScreenRows *
Host.ScreenCols);
```

VB

```
REM
REM scrape screen and take appropriate action
REM
Dim Screen As String
Screen = Host.getDisplayText(1, 1, Host.ScreenRows * Host.ScreenCols)
```

Tag

This is an application defines standard control tag that may be used by the application to store information.

C#

Host.Tag = “My Special Tag”:

VB

Host.Tag = “My Special Tag”

TraceActive

This property turns the trace option on and off. When this option is enabled, a trace file with the name in the property TraceFileName will be created.

C#

```
Host.TraceActive = false;     //set to true to get a trace
```

VB

```
Host.TraceActive = False  REM set to true to cause a trace file to be
generated
```

TraceFileName

The name of the file to place tracing information into. This file is only created if the TraceActive property is set to True.

C#

```
Host.TraceFileName = "C:\\TRACEFILE.TXT";
```

VB

```
Host.TraceFileName = "C:\\TRACEFILE.TXT"
```

Delegates, Events, and Handlers

Delegates are called by the VirtualHost class to inform the application of some type of event. Most delegates return the class as the first argument to identify the class that it came from and facilitate multiple concurrent VirtualHost instances. This may be used to identify the control in the case that the application has more than one control in the application.

Threading

Because the VirtualHost class uses the .NET Socket TCPIP class, it uses 2 threads for execution, the owners thread that constructed the VirtualHost class and a TCPIP thread to asynchronously send and receive information to the host computer. This means that when the VirtualHost fires an event back to the application, it may be on either thread. Depending on what the application wants to do, it may or may not have to switch to a safe thread. For example, A Windows Forms application wanting to display something inside an evtScreenChange event will have to switch to the main GUI thread before calling the .NET framework to display the information. This is because the .NET framework only allows painting the window in the main GUI thread. For more information on threading in .NET application, see the Microsoft web site for a number of technical documents.

It is important to call the VirtualHost.Close method when the application is done using the class so that the TCPIP conection can be closed and the trhead freed up. Failure to do this can cause exceptions to occur.

evtConnected

This delegate is fired when the terminal control connect attempt is complete. The ConnectedOk argument indicates the outcome of the connect attempt. If the connection attempt fails, the Message property contains the error string.

C#

```
Host.evtConnectComplete +=new
NewhartSystems.VirtualHost.evtConnected(Host_evtConnectComplete);


.

.

.
/// <summary>
/// Connection attempt has completed
/// </summary>
/// <param name="Control"></param>
/// <param name="ConnectedOk"></param>
private void Host_evtConnectComplete(NewhartSystems.VirtualHost Control,
bool ConnectedOk)
{


}
```

VB

```
AddHandler Host.evtConnectComplete, AddressOf Host_evtConnectComplete

.
.
.

Protected Sub Host_evtConnectComplete(ByVal Control As
NewhartSystems.VirtualHost, ByVal Connected As Boolean)



End Sub
```

evtDisconnected

This delegate is fired when the terminal session connection is lost or the operator selected to disconnect from the host.

C#

```
Host.evtDisconnected +=new
NewhartSystems.VirtualHost.evtLostConnection(Host_evtDisconnected);
.

.

.

.
/// <summary>
/// Disconnect from host has occured
/// </summary>
/// <param name="Control"></param>
private void Host_evtDisconnected(NewhartSystems.VirtualHost Control)
{

}
```

VB

```
AddHandler Host.evtDisconnected, AddressOf Host_evtDisconnected
.

.

.
Protected Sub Host_evtDisconnected(ByVal Control As
NewhartSystems.VirtualHost)


End Sub
```

evtKeyboardUnlocked

This delegate is fired when the host application unlocks the keyboard. When a screen is sent to the host, the keyboard becomes locked until the host application unlocks it.

C#

```
Host.evtKeyboardUnlocked +=new
NewhartSystems.VirtualHost.evtevtKeyboardUnlocked(Host_evtKeyboardUnlock
ed);
.

.

.

.
/// <summary>
/// Keyboard has been unlocked
/// </summary>
/// <param name="Control"></param>
private void Host_evtKeyboardUnlocked(NewhartSystems.VirtualHost
Control)
{

}
```

VB

```
AddHandler Host.evtKeyboardUnlocked, AddressOf Host_evtKeyboardUnlocked
```

```
.

.

.
Protected Sub Host_evtKeyboardUnlocked(ByVal Control As
NewhartSystems.VirtualHost)


End Sub
```

evtScreenChange

This delegate is fired when the host application has written to the session screen area. It informs the application that the screen has changed.

C#

```
Host.evtScreenChanged += new
NewhartSystems.VirtualHost.evtScreenChange(Host_evtScreenChanged);
.

.

.
/// <summary>
/// Here the session screen has changed, retreive the screen contents
/// and parse the screen
/// </summary>
/// <param name="Control"></param>
private void Host_evtScreenChanged(NewhartSystems.VirtualHost Control)
{
      //
      // scrape screen and take appropriate action
      //
      string Screen = Host.getDisplayText(1,1,Host.ScreenRows *
Host.ScreenCols);

      //
      // ADD YOUR CODE HERE
      //
      // Parse screen, simulate keystrokes and wait for next screen
      //

}
```

VB

```
AddHandler Host.evtScreenChanged, AddressOf Host_evtScreenChanged
.

.

.
Protected Sub Host_evtScreenChanged(ByVal Control As
NewhartSystems.VirtualHost)

REM
REM scrape screen and take appropriate action
REM
Dim Screen As String
Screen = Host.getDisplayText(1, 1, Host.ScreenRows * Host.ScreenCols)

REM
REM add your code here to automate keyboard entry and wait for next
screen
REM

End Sub
```