WISP for WinSSHD

9/7/2011

1.1 WISP for Windows SSH Support with WinSSHD

The WISP for Windows SSH support allows WISP migrated applications to be run remotely on a Windows Server within a SSH session using the WinSSHD SSH server by Bitvise Limited (www.bitvise.com) . This functionality may be used to improve performance of applications for remote users.

Windows applications are normally designed to run on a desktop client machine that is connected to a server using a high-speed LAN connection. The server is normally used as a "file server" where the data files and sometime the executable files are stored. When you run your applications on your desktop client all the data from the server is moved across the network as it is needed.

COBOL applications that use Indexed files can cause a lot of network traffic because every IO operation on an indexed file typically requires 5-8 (or more) physical IO operations. Each of these physical IO operations causes data records to move back and forth across the network. On a high-speed LAN this is not a problem but on a low-speed remote connections the performance may not be acceptable.

The WISP for Windows SSH support allows the remote user to connect to the Windows Server via a SSH session and run their WISP migrated applications on the Windows Server instead of on the client. By running the applications on the server instead of the client the only data traveling across the network is the screen images and the keyboard keystrokes. This is the same configuration used on UNIX where the applications always run on the server.

Windows Server does not natively support SSH services. To allow your Windows Server to support SSH connections you must use third party software that provides SSH services. The only service that is currently supported is WinSSHD by Bitvise Limited.

The WISP for Windows SSH support was designed so that one server can be configured to run migrated applications in all networking modes. A single server can be configured to run applications locally at the server using remote desktop, and from a Windows client on the LAN using Windows networking, and from a remote SSH client. The same configuration will work for all modes of connection.

1.1.1 Windows SSH Requirements

The following software is required to use the WISP for Windows SSH support.

- **WinSSHD** 5.23 (or later) by Bitvise Limited. (www.bitvise.com)
- Micro Focus (Acucobol) Extend Cobol Console mode runtime (crun32.exe).
- A Microsoft Windows Server 2003 (or later).
- A SSH client program.
  - **Tunnelier** an SSH client by Bitvise Limited is recommended. In testing it provided the best user experience and truest representation of a console application.
  - **PuTTY** a free open source SSH client was also tested and found to give satisfactory results.

1.1.2 Windows SSH Restrictions

The following restrictions apply for running WISP migrated applications in a SSH session.

- Only console application may be run within a SSH session.

  Windows support two types of applications, they are "console" applications and "windows" applications. A console application performs all its screen IO within a windows console (MS-DOS style command box). A windows application expects to use the windows GUI to create a user interface. Windows applications cannot run within a SSH session. As a example the Windows Notepad.exe program is a windows application and cannot be run from within a SSH session.

- You must use a Console mode version of the Acucobol Extend runtime system.

  The standard Acucobol runtime system is a windows application. You must build a Console version (crun32.exe) of the runtime. Acucobol refers to this as the "Console" runtime. See configuration step 2 below for instructions.

- Support for Native Acucobol Screen IO is undetermined.

  No attempt has been made to be compatible with native Acucobol screen IO. It is unknown whether native Acucobol screen IO works correctly in a SSH session. Applications migrated by WISP default to using the WISP screen handler and do not use the Acucobol screen handler.

- The DISPLAYUTIL option is disabled from within a SSH session.

  The DISPLAYUTIL option is usually used to set Notepad.exe as the display utility. Since Notepad.exe is a windows application and it cannot be used within a SSH session the DISPLAYUTIL option is disabled in a SSH session. The standard WISP DISPLAY utility will always be used when running within a SSH session.

- The UTILSWINDOWS option is disabled from within a SSH session.

  The UTILSWINDOWS option allows utilities like DISPLAY to run in a separate window. A SSH session has no concept of multiple windows, it only maintains a single session connection. The UTILSWINDOWS option will be ignored for a SSH session.

- There is no native mouse support with SSH.
- Color support is variable depending on the SSH client.

  The **Tunnelier** SSH client was found to give excellent color support that exactly matched the VCOLORS selections.

  The **PuTTY** SSH client was found to have limitations in color support. It appears not to support high intensity colors (hex values 8-F) for background colors. As an example the default bright white background (hex F) was rendered as light grey (hex 7). Through the use of PuTTY "Colours" configuration options and custom VCOLORS settings a satisfactory color scheme is possible.

- Bell support is variable depending on the SSH client.

  The **Tunnelier** SSH client provides single bell support but not multiple bells.

  The **PuTTY** SSH client was found to not provide bell support.

- Graphic and non-ASCII character support is variable depending on the SSH client.

  Graphic characters are used for line drawing and for pseudo-blanks, non-ASCII characters are used with CHARMAP for foreign language support.

  The **Tunnelier** SSH client provides excellent support for graphic characters and non-ASCII characters.

  The **PuTTY** SSH client was found to not support graphic and non-ASCII characters. PuTTY offers a number of "Translation" configuration settings, however none were found to give satisfactory results.

  Non-ASCII characters with CHARMAP can be inspected by running WSHELL and pressing PF28 (^F28) to view the "Screen Character Ser Mapping" screen. Line drawing graphics characters can be seen by running the DISPLAY utility and pressing HELP (^E).

- The Workstation numbers returned by EXTRACT "W#" are not unique.

  On the Windows the workstation number is looked up in the WSYSCONF.CFG file based on the computer name. This means that every program running on the server in a SSH session will have the same workstation number. If your applications require a unique session number you should use the process group id "G#" instead.

- The WRUN utility cannot be run in a SSH session.

  The WRUN utility is a windows application so it cannot be run in a SSH session. A console version named WRUNT.EXE has been provided for use with the console version of the runtime system. When running in a SSH session you must use the WRUNT.EXE console version. You can rename WRUNT.EXE to WRUN.EXE if desired.

- The WCONFIG utility cannot be run in a SSH session.

  The WCONFIG utility is a windows application so it cannot be run in a SSH session. The WCONFIG utility must be run from the server's desktop directly.

  It is recommended that you use WISP environment variables to configure the runtime when running within a SSH session. See WCONFIG for a list of environment variables needed.

- The WISPTRAN utility cannot be run in a SSH session.

  The WISPTRAN utility is a windows application so it cannot be run in a SSH session. Further, the WISP for Windows SSH support is for running migrated applications, it was not intended to be used for translating and compiling development tasks.

1.1.3 Windows SSH Configuration

Follow these steps to configure your Windows Server for running WISP migrated applications from within a SSH session.

Step 1) Install WISP.

Perform a normal installation of WISP. Follow all the normal installation and configuration steps.

Step 2) Build a Console version of the ACUCOBOL runtime system.

To build the console version of the ACUCOBOL runtime system see the instructions in the distribution based on which release of Acucobol you are using.

For Acucobol 9.0 see ${WISPDIR}\acu\acu90\ build_wisp_acu90_con_rts.txt

You may need to change your ${WISPCONFIG}\WRUN.CFG file to use the new runtime.

You will also want to uncomment the CONSOLEACU option in the WISPCONFIG\OPTIONS file. The CONSOLEACU option is used on Windows when using Windows SSH support to instruct the runtime that the ACUCOBOL RTS is a "Console" application instead of the default "Windows" application. This will prevent window flashing on the CALL "LINK" when using the console runtime outside of SSH.

Step 3) Install WISP migrated applications.

Perform a normal installation and configuration of your WISP migrated applications onto your Windows Server. Test your applications to ensure that they run correctly from both a Windows client on the LAN and from the server workstation. This ensures that the applications are configured correctly for network access and that all file paths are being properly constructed for network access.

Be certain your applications are working correctly in both configurations before proceeding.

Step 4) Install WinSSHD services.

Install and configure the WinSSHD service onto the Windows Server.

You can download WinSSHD from the Bitvise website (www.bitvise.com).

Step 5) Configure WISP for running in SSH.

There are a number of environment variables that you may need to set while in a SSH session so that WISP migrated applications behave properly.

If you are configuring the WISP runtime using environment variables (recommended) instead of WCONFIG then at a minimum you will need the following:

```
WISPDIR
WISPCONFIG
WISPTMPDIR
```

Other environment variables you may need are:

```
WISPTERM
WISPSHAREDIR
WISPLINKPATH
VCOLORS
```

These variables are all fully described in the WISP manual.

The WISP runtime can normally automatically determine that it is running under the WinSSHD server by checking the environment variable WINSSHDGROUP that is automatically set by the WinSSHD server. This behavior can be explicitly controlled by setting the variable WISPWINSSHD=1 when running under WinSSHD (or WISPWINSSHD when not under WinSSHD).

```
WISPWINSSHD=1
```

In a SSH session the videocap terminal type is retrieved from the environment variables WISPTERM or TERM in that order. The registry is not used to determine the videocap

type when running in a SSH session. If not set the default "wincon" videocap file will be used.

```
WISPTERM=wincon
```

In a SSH session you start programs from the command line so you should set the PATH variable to include the WISP bin directory. This can be done for all users and all processes through the Control Panel --> System --> Environment Tab --> System Variables. Set PATH to include C:\WISPxxxx\BIN. As an alternative you can set the PATH environment variable using a bat file.

```
set PATH=%WISPDIR%\bin;%PATH%
```

An example **startup.bat** file may look like this:

```
@echo off
set WISPDIR=C:\WISP
set WISPCONFIG=C:\WISP\CONFIG
set WISPTMPDIR=C:\WISP\TEMP
set WISPSHAREDIR=C:\WISP\SHARE
set WISPLINKPATH=C:\WISP\BIN;C:\WISP\ACU;C:\APPS
set VCOLORS=0A0CA0ACEAECBABCA0C00ACAAECEABCB
set PATH=%WISPDIR%\bin;%PATH%
wrunt MYAPP
rem (or) wproc STARTAPP
```

1.1.4 Additional Notes

Screen I/O under WinSSHD is perfromed using Console mode I/O as described in the WISP manual. This is different then the WISP support for Telnet on Windows which uses "stream mode I/O".

1.1.5 Wprint64 for 64-bit Servers

At the time of this writing there is a bug on all Windows 64-bit servers that prevents multiple users from printing with 32-bit programs. The WISP runtime is a 32-bit program.

Microsoft Knowledge Base issue KB972616 describes the problem (but not a fix). See http://support.microsoft.com/kb/972616

Specifically in the section “More Information” you will see this paragraph:

> *On a computer that is running a 64-bit version of Microsoft Windows, only one user account may print from a 32-bit program in a single session. In a single session, the user account that prints first is the only user account in which a 32-bit process can print, until a time-out occurs or the session ends. If another user account in the same session tries to print before the session ends, the user account receives an "Invalid Handle" error message. Additionally, the print request is unsuccessful.*

Unfortunately the recommended “Hot Fix” doesn’t actually address the big problem, it only allows you to shorten the time you have to wait after the user that printed exits before the next user can print.

If you are running on a 64-bit server then you will need to use a 64-bit print utility to avoid this problem.

Shell Stream Software has developed the utility wprint64 to address this issue. This utility is available for no charge to licensed WISP users upon request.