Maintenance Documentation

i-scream WinHost

WinHost is a host application for use with the i-scream Distributed Central Monitoring System. This document provides an overview of how *WinHost* works and how you may alter it to suit your requirements.

Revision History

24/03/01	Initial creation				
		Committed by:	pjm2	Verified by:	
				Date:	
		Committed by:		Verified by:	
				Date:	
		Committed by:		Verified by:	
				Date:	
		Committed by:		Verified by:	
				Date:	
		Committed by:		Verified by:	
				Date:	

Introduction

WinHost is a host application for use with the i-scream Distributed Central Monitoring System. It is designed to be easy to alter so that those with even a limited knowledge of Visual Basic (or, indeed, any similar language) should be able to change it to suit any specific requirements. Visual Basic was used in order to provide a simple to alter host, and also due to the short amount of time we had to develop a Windows host.

Overview of what the WinHost does

The WinHost, like any i-scream host, is an application that harvests data from the machine it is running on. It then sends this data to a central server for processing.

Supported Platforms

The WinHost is designed to perform on Microsoft Windows NT or Windows 2000 and is primarily intended to be used for server monitoring.

Using WinHost

WinHost is very simple to install and use. Please check the separate usage documentation for more details.

Understanding what WinHost does

WinHost is written in Visual Basic 6, and those who know this language should be able to understand how and what it does simply by looking at the source. However, for those who are not so sure, here is an overview of the basic architecture of the program.

Required development files

CnetWksta.cls	A Visual Basic class by Karl E. Peterson that is used to obtain the number of users logged on to the machine.
CupTime.cls	A Visual Basic class used to interface with *pdh.dll*. This is used to safely obtain uptime on a machine that has been up for more than 47 days.
Imports.bas	Declares our API calls to kernel32. These are used to obtain the majority of system information for the WinHost.
Nettest.frm	Visual Basic 6 Form file. This determines the layout of the WinHost, the imported ActiveX controls and the operation of the WihHost.
Nettest.frx	Visual Basic 6 Binary FRX.
Nettest.vbp	Visual Basic 6 Project file.
Nettest.vbw	Visual Basic 6 Window file.

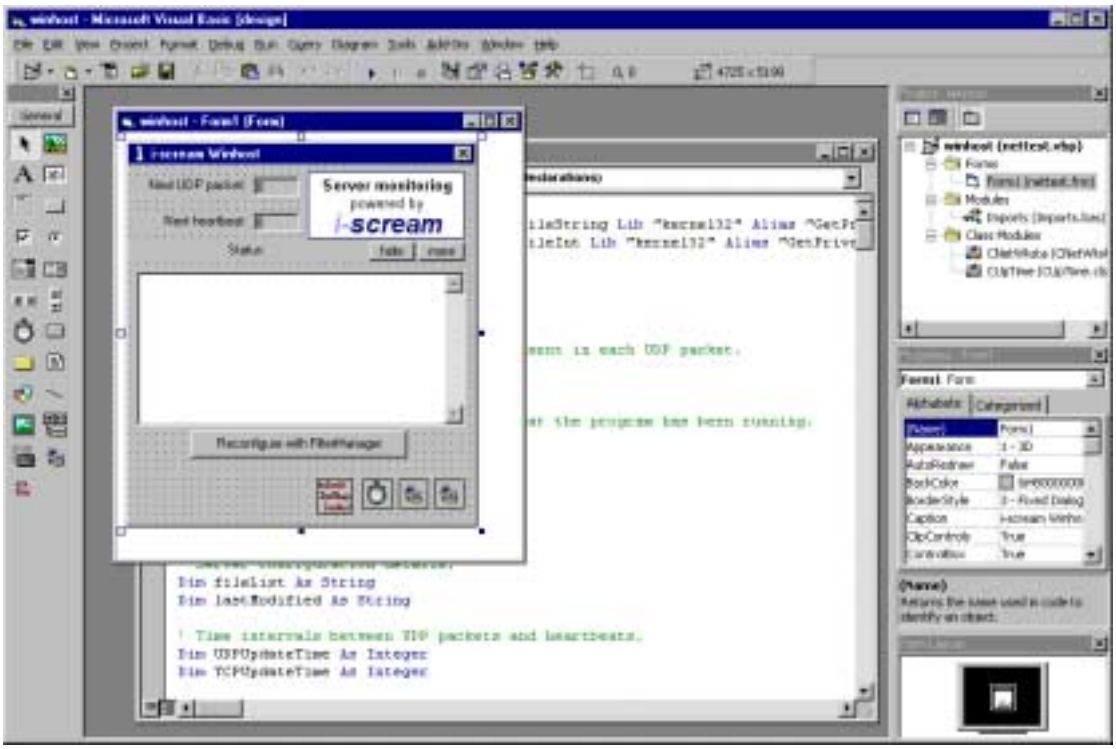
Required runtime files

winhost.exe	The WinHost executable.
mswinsck.ocx	Microsoft WinSock ActiveX control.
systray.ocx	Microsoft System Tray ActiveX control.
pdh.dll	Dynamic link library used to obtain true uptime and other CupTime class features.
winhost.ini	The configuration file that tells the WinHost which filter manager to obtain further configuration from. If this file is not present, the WinHost will exit with a warning.

The Visual Basic 6 runtime engine will also need to be present on the server. This is standard on Windows 2000.

Design Environment

WinHost in the Visual Basic IDE



The WinHost form layout has been designed within the Visual Basic IDE. Note that the form includes 4 controls that are hidden from the end user: -

1. ActiveX system tray OCX control.
2. A Visual Basic Timer control.
3. A Winsock control for handling all TCP communications from the WinHost.
4. A Winsock control for sending UDP packets from the WinHost.

Overview of operation

Startup checks

When a WinHost is started, it first checks to see if there is already another WinHost running. If there is, the user is notified of this and the WinHost exits.

The CUpTime class is initialised on startup which enables it to be used to obtain the true uptime of the system later on.

Many of the API calls are based on the NT architecture; thus, the WinHost will not be able to perform on Windows 9x machines. If a Windows 9x machine is detected, then the WinHos will exit with an error message.

The machine name and port number of the filter manager are read from *winhost.ini*. This file may be located in either the same folder as the winhost.exe executable, or in the windows\system32 folder.

The WinHost then proceeds to call the *Reconfigure_Click* function to connect to the filter manager.

Finally, the program icon is installed to the Windows system tray.

Normal running

Configuration on demand

The WinHost uses the same Winsock control to perform both configuration with the filter manager, and subsequent communications with filters. The user may instruct the WinHost to reconfigure with the filter manager at any moment, so it is important to ensure that both communications cannot happen at the same time. This is handled by the *Reconfigure_Click* function.

Communication with the filter manager or filter is started by calling the *TCPSock_Connect* function. This function is then starts off the configuration process if we are connecting to a filter manager, or starts a heartbeat with a filter.

Periodic Events

The connection with the filter manager is used to obtain the configuration for the WinHost. Two of these configuration values are UDPUpdateTime and TCPUpdateTime. These specify the number of seconds between each UDP packet being sent to a filter and the number of seconds between each TCP heartbeat with a filter, respectively.

The timing of these two independent events is triggered by the use of the Timer control on the form.

The Timer resolution is 1000 milliseconds and is responsible for decreasing the values visible to the user for next UDP packet and next heartbeat times.

If the Timer fires when it is time to perform a heartbeat, it does this by ensuring the TCP Winsock control is closed, and then connects to the filter using: -

```
TCPSock.Connect filterHostname, filterTCPPort
```

If the Timer fires when it is time to send a UDP packet to the filter, it proceeds to generate the contents of that packet before sending it.

TCP connections

Configuration

The TCP connection used to configure the WinHost with a filter manager uses the protocol defined at: -

http://www.i-scream.org.uk/cgi-bin/docs.cgi?doc=specification/protocols.txt

All bytes returned to the WinHost are handled by the *TCPSock_DataArrival* function. In the event of a configuration connection, this function will proceed to obtain the WinHost's configuration from the filter manager.

In the event of an error occurring, the WinHost will close the connection and report the error in the text area.

Heartbeats

The TCP connection used during heartbeats with filters follows the protocol defined at: -

http://www.i-scream.org.uk/cgi-bin/docs.cgi?doc=specification/protocols.txt

Note that if the "OK" response is not received when the WinHost sends its last modified time, the WinHost will reconfigure with the filter manager.

UDP Sending

Building the contents of packets

When a UDP packet is to be sent, the WinHost first begins by building the contents of the packet in a string named *xml*.

The following values are sent within the XML structure: -

seqNo

Each packet sent **must** contain a sequence number. This is incremented each time the WinHost sends a UDP packet. This number starts from 1. It is important to note that the server will reject packets that do not contain a sequence number.

machineName

This is the fully qualified domain name of the machine on which the WinHost is running. The value of this is obtained from the filter manager during configuration in order to ensure that all hosts send their machine name in the same style. All packets **must** contain their machine name.

packetDate

All packets **must** contain a timestamp. This is formatted as the number of seconds since the epoch. The *Date2Num()* function in *nettest.frm* is used to provide this value.

LocalIP

All packets **must** contain the I.P. address of the machine on which WinHost is running. This is obtained from the *TCPSock* control that is used to configure the WinHost, so it is likely to provide the correct I.P. address on machines that have more than one network interface card installed.

netbiosName

Windows machines may be more easily identified by their NetBIOS names. The TCPSock is used to provide the NetBIOS name, however, it is not essential to include this in packets.

osName

The value of osName is found by calling the GetVersion function (defined as part of the kernel32 API). This returns a string representing the name of the operation system being used.

osVersionMajor *and* osVersionMinor

These are obtained from the user type *OSVERSIONINFO* defined in *Imports.bas*. These are concatenated in the packet to form a complete version number, for example first versions of Windows 2000 will return 5 and 0 respectively, which will be sent as “5.0” in the packet.

osBuild

This is the internal build version of the operating system. This is also obtained from the user type *OSVERSIONINFO*.

processorType

This is the type of processor that the platform is using. This is obtained by examining the number returned from *sysinfo.dwProcessorType*.

uptime

This is the uptime of the machine. This must be sent as an integer representing the number of seconds that the operating system has been running. The CupTime class is used to provide this value, as the normal API call to do this wraps after approximately 47 days uptime.

userCount

This specifies the number of users logged on to the machine. We obtain this using the *wksta* object.

percent_idle

This specifies the percentage of idle CPU time on the machine. This is obtained from the CupTime class.

cpu_time

This specifies the percentage of user CPU time on the machine. This is obtained from the CupTime class.

memTotal

The total amount of physical memory available to the machine. This value is sent in the packet in megabytes.

memFree

The amount of unused physical memory available to the machine. This value is sent in the packet in megabytes.

swapTotal

The total amount of swap space available to the machine. This value is sent in the packet in megabytes.

swapFree

The amount of free swap space available to the machine. This value is sent in the packet in megabytes.

Sending the UDP packet

Once the contents of the XML string have been built, this string is printed to the text area so that the user may view the data being sent. This helps to assure users that we are not collecting any sensitive data. The contents of the XML string are then sent to the filter address (obtained by configuring with the filter manager) in a single UDP packet. It is important to send the whole string in a single packet, as the i-scream monitoring system will reject any packet if the whole packet does not contain valid XML.