How to PING

by Randy Charles Morin

I have often been asked how to replicate the PING command-line utility. I often ignored the request and went on to subject that I was more familiar with. But the question has been increasing in frequency over the last few months. I family sat down and tried to hack something together that could be re-used.

The usefulness of the class is limited. The only use I've been able to ascertain is to determine if a particular server is responding to ICMP requests. So, I don't think I'll be able to talk long about how useful this re-usable code is. What's that? ICMP? You don't know what ICMP is? Ok! I'll fill the article with random gibberish about ICMP and its use on the Internet.

Let's start with the basics. What exactly is a PING? A little known fact is the PING is actually a short-form acronym for Packet Internet (or Inter-Network) Groper. The PING utility is most often used in diagnosing network problems. The utility sends one packet of data to the target host and waits for an identical (or echoed) reply from the host. The idea behind the PING is that responding to one packet of data is generally the lowest common denominator in diagnosing network presence. If the host can reply to one packet, then you can be reassured that the host exist. Imagine the situation where a user is having difficulty navigating the corporate website. The user can determine minimal network presence by trying to PING the website.

The PING utility can generally send more than one request and this can also be used to determine if the network is intermittently dropping packets. Last, the PING utility can also be used to determine if the naming service is properly translating names into IP addresses.

ICMP

Now I'm pretty certain you already knew what the PING utility did. You are more likely interested in finding out the how than the what. Here 'tis in black and white.

The PING utility does not use TCP (Transmission Control Protocol) like most other network traffic. Neither does it use UDP (User Datagram Protocol). Rather it uses an often forgotten Internet protocol called ICMP (Internet Control Message Protocol). The intent of ICMP has always been in diagnosing IP-network issues. The entire protocol is a quick low-level access point into the IP-network. If you are familiar with the OSI Reference Model, then skip the next couple of paragraphs.

The OSI Reference Model divides the task of communicating over a network into seven layers, application, presentation, session, transport, network, data-link and physical. The TCP, UDP and ICMP protocol are transport layers, whereas the IP protocol is network layer. Much of the IP traffic these days is TCP, a connection-oriented transport layer for the Internet. The TCP connection-oriented protocol provides a lot of services not provided by the IP layer, like ports, checksums, framing and sequencing packets. The UDP connectionless protocol on the other hand does not provide framing or sequencing of packets, but does provide minimal services like ports and checksums. ICMP is yet

another step down from UDP and provides only a checksum service. ICMP is really a higher-level tap into the low-level IP-network.

The Class

Now let's get down and dirty and write some code. First off, I have no idea how to send ICMP packets with the socket library. I'll leave that for the pros. My sample code will use a little known ICMP.DLL found in the System32 folder. This DLL exports eight functions. I'll limit our discussion to the three important ones.

```
HANDLE WINAPI IcmpCreateFile (VOID);
BOOL WINAPI IcmpCloseHandle (HANDLE IcmpHandle);
DWORD WINAPI IcmpSendEcho (HANDLE IcmpHandle,
                      IPAddr DestinationAddress, LPVOID RequestData,
                      WORD RequestSize, PIP_OPTION_INFORMATION RequestOptions,
                      LPVOID ReplyBuffer, DWORD ReplySize, DWORD Timeout);
```

The IcmpCreateFile function returns a handle that is used to execute ICMP requests. The IcmpCloseHandle function closes the handle created by IcmpCreateFile. The last function is IcmpSendEcho. This function given a handle will send one ICMP packet to a host and wait a specified timeout period or until a reply is received.

```
HANDLE icmphandle = IcmpCreateFile();

char reply[sizeof(icmp_echo_reply)+8];

icmp_echo_reply* iep = (icmp_echo_reply*)&reply;
iep->RoundTripTime = 0xffffffff;

DWORD dw = IcmpSendEcho(icmphandle,
                        *((u_long*)host->h_addr_list[0]),
                        0,0,NULL,
                        reply,sizeof(icmp_echo_reply)+8,5000);

if (dw == 0)
{
        throw IcmpException("send echo failed");
}

IcmpCloseHandle(icmphandle);
```

The code for sending one ICMP packet with zero content is very minimal. Create the handle, setup the reply buffer, call the IcmpSendEcho method, test for error conditions and close the handle.

The only part that is left is to figure out how to construct the *((u_long*)host->h_addr_list[0]) or host address. The host structure is exactly similar to those I present in previous articles in this series. You can view those articles at http://www.kbcafe.com/ip/.

```
hostent * host;
in_addr inaddr;
inaddr.s_addr = ::inet_addr(strAddress.c_str());
if (inaddr.s_addr == INADDR_NONE)
{
        host = ::gethostbyname(strAddress.c_str());
}
else
{
        host = ::gethostbyaddr((const char *)&inaddr,
                        sizeof(inaddr), AF_INET);
}

if (host == NULL)
```

```
        {
                throw IcmpException("invalid SMTP server");
        }
```

In this code sample, I determine if the address is an IP address or a name that should be resolved to an IP address. Then I construct the host structure from the name or address.

I wrapped the ICMP code in a class named appropriately enough Icmp. I also created an IcmpException class that I use to capture and send exceptions down the call stack.

Listing 1: ICMP Code

```
//////////////////////////////////////////////////////////////////////
//
// icmp.h: implementation of the Icmp class.
// Copyright 2001 by KBCafe
//
//////////////////////////////////////////////////////////////////////

#ifndef KBCAFE_ICMP_H
#define KBCAFE_ICMP_H

#include
#include
#include
#include
#include
#include

namespace
{

class WSAInit
{
public:
        WSAInit()
        {
                WORD w = MAKEWORD(1,1);
                WSADATA wsadata;
                ::WSAStartup(w, &wsadata);
        };

        ~WSAInit()
        {
                ::WSACleanup();
        };

} instance;

};

namespace kbcafe
{

class IcmpException : public std::exception
{
        std::string m_str;
public:
        IcmpException()
        {
                m_str = "ICMP exception.\tWSAGetLastError = "
                        +::WSAGetLastError();
        }

        IcmpException(const std::string & str)
        {
                std::stringstream ss;
                ss << "ICMP exception:"
                        << str
                        << "\tWSAGetLastError = "
```

```
                        << ::WSAGetLastError();
                m_str = ss.str();
        }

        virtual const char * what() const throw()
        {
                return m_str.c_str();
        }
};

class Icmp
{
        static void SendEmptyEcho(const std::string & strAddress);
};

inline void Icmp::SendEmptyEcho(const std::string & strAddress)
{
        hostent * host;
        in_addr inaddr;
        inaddr.s_addr = ::inet_addr(strAddress.c_str());
        if (inaddr.s_addr == INADDR_NONE)
        {
                host = ::gethostbyname(strAddress.c_str());
        }
        else
        {
                host = ::gethostbyaddr((const char *)&inaddr,
sizeof(inaddr), AF_INET);
        }

        if (host == NULL)
        {
                throw IcmpException("invalid SMTP server");
        }

        HANDLE icmphandle = IcmpCreateFile();

        char reply[sizeof(icmp_echo_reply)+8];

        icmp_echo_reply* iep = (icmp_echo_reply*)&reply;
        iep->RoundTripTime = 0xffffffff;

        DWORD dw = IcmpSendEcho(icmphandle,
*((u_long*)host->h_addr_list[0]),
                0,0,NULL,
                reply,sizeof(icmp_echo_reply)+8,5000);

        if (dw == 0)
        {
                throw IcmpException("send echo failed");
        }

        IcmpCloseHandle(icmphandle);

        return;
}

};

#endif
```

When initially testing the class, I found that it would not work for me. It was Friday, so I knew it had to be one of the Friday reasons, like my brain was fried. True enough it was. I forgot to initialize the socket library. So, I cut and paste my trusty WSAInit class and I was back off to the races. The WSAInit class has a performance consequence in that it causes the socket initialization code to be called each time the header file is included in a

source listing. This performance inhibiter should only occur at startup, but if you need a quicker startup, then moving the WSAInit class to a source listing might help a bit.

Finally, I wrote a little script to test my new ICMP code. I conveniently named the code PING.

Listing 2: PING utility

```
// ping.cpp : Defines the entry point for the console application.
//

#include
#include "kbcafe/icmp.h"

int main(int argc, char* argv[])
{
	if (argc != 2)
	{
		std::cerr << "Usage:\tping [address]";
		return 1;
	}

	try
	{
		kbcafe::Icmp::SendEmptyEcho(argv[1]);
		std::cout << "Success" << std::endl;
	}
	catch(kbcafe::IcmpException & e)
	{
		std::cerr << e.what();
	}

	return 0;
}
```

This PING utility is not nearly as useful as the one shipped with most OSes, but at least you know how this utility was coded. Try it out, it's not bad.

You can read more about the ICMP protocol in RFC 792, http://www.ietf.org/rfc/rfc0792.txt?number=792. This article is the latest in a long series of IP protocol articles written by Randy Charles Morin. The bulk of the articles are listed at http://www.kbcafe.com/ip/. You can download the Icmp class along a dozen other useful classes from the File Cabinet of the KBCafe community at MSN, http://communities.msn.ca/KBCafecom/files.msnw?fc_p=%2Fkbcafelib&fc_a=0.

About the Author

Randy Charles Morin is the Lead Architect of SportMarkets Development from Toronto, Ontario, Canada and lives with his wife and two kids in Brampton, Ontario. He is the author of the www.kbcafe.com website, author of Wiley's Programming Windows Services book and co-author of many other programming books.