**Version: 1.0, 3.0, 4.0**

**June 1999**

**Pinging a server**

**by Kent Reisdorph**

<http://bcbjournal.org/articles/vol3/9906/Pinging_a_server.htm>

Internet programming provides many challenges. One of those challenges is detecting whether or not your user is connected to the Internet before attempting to connect to a POP3, SMTP, FTP, or other server. More specifically, you might need to check that the particular remote machine to which you're trying to connect is actually online. This article will explain how to ping a remote machine. We'll explain how to obtain components that allow you to perform a ping. Then, we'll show you how to write code to ping a server using the scarcely documented ICMP.DLL.

**What's ping?**

First of all, PING is a program. One version or another of PING ships with most UNIX systems, as well as with Windows 95/98 and Windows NT. The term *ping* has since been adopted to refer to the process by which a remote machine is sent a packet of data to which it responds. It can be boiled down to, "Hey, machine! Are you there?" To which the remote machine responds, "Yes, I am here," or, if the machine is down or you're not connected to the Internet, doesn't respond at all. The original PING program was written by Mike Muuss more than 15 years ago. Contrary to some statements to the contrary, PING is *not* an acronym. Rather, PING is named after the process used by sonar; a signal is sent out and an echoing signal is received, or not received, in reply. If you want to know more about the PING story, see Mr. Muuss' *The Story of the PING Program* at

[**http://ftp.arl.mil/~mike/ping.html**](http://ftp.arl.mil/~mike/ping.html)

To see how PING works, first connect to the Internet, if necessary. Next, open a command prompt box in Windows and type the following:

ping www.borland.com

If your ISP doesn't have a DNS server running, then you'll need to type the actual IP address of the server:

ping 207.105.83.51

The PING program will respond with something like the following:

Reply from 207.105.83.51: bytes=32 time=91ms TTL=245

Reply from 207.105.83.51: bytes=32 time=70ms TTL=245

Reply from 207.105.83.51: bytes=32 time=70ms TTL=245

Reply from 207.105.83.51: bytes=32 time=90ms TTL=245

The response from PING lists the remote machine's IP address, the number of bytes in the packet sent to the machine, the round trip time in milliseconds, and the time to live (TTL) value. (TTL isn't important for this discussion, but it basically specifies the amount of time this packet is considered to be good.) The PING program is great for testing your connection to the Internet and for checking if a particular machine is up. However, it doesn't help much if you want to perform that task from within a C++Builder program.

**Ping, the easy way**

Without question, the easiest way to implement ping services in your C++Builder applications is to get a component that does the work for you. One such component is Francois Piette's TPing component. TPing is part of Francois' Internet Component Suite (ICS). You can download ICS from the Francois Piette's Web site at

[**www.rtfm.be/fpiette/icsuk.htm**](http://www.rtfm.be/fpiette/icsuk.htm)

Once you've downloaded and installed ICS, you can simply drop a TPing component on a form and write code like this:

if (Ping1->Ping())

// Ping found the remote machine.

ShowMessage("Machine replied");

else

// Machine did not respond.

ShowMessage("No response");

There are other ping components available, but ICS has a good reputation and the components are freeware.

**TNMPing**

One other ping component I want to mention is the TNMPing component from NetMasters. The original NetMasters Internet components that ship with C++Builder Professional doesn't contain a ping component. NetMasters has made an update to their components available for download from their Web site. This update adds the TNMPing component to the FastNet component suite. You can download the update from the NetMasters Web site at

[**www.netmastersllc.com**](http://www.netmastersllc.com/)

**Writing your own ping routine**

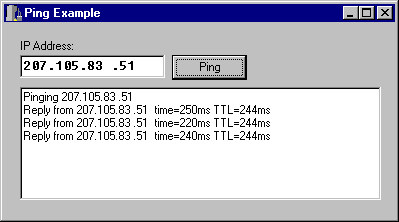
If you need more control over the ping process, or if you simply want to delve into the mysteries of Windows, then you can write your own ping routine. There are essentially two ways to go about writing a ping routine:

|  |  |
| --- | --- |
|  | Use the functions found in Microsoft's ICMP.DLL. |
|  | Use Winsock 2 raw sockets (SOCK\_RAW). |

We'll focus on the first of these methods in this article. If you want to find out more about using raw sockets, you can find information on the Microsoft Developer's Network (MSDN) CD-ROM. There you'll find an example program that shows how to implement raw sockets. If you don't have MSDN, you can check the Microsoft Web site for information on this topic. In either case, search for the article entitled *Ping: SOCK\_RAW in Winsock 2.0*.

**Ping using ICMP.DLL**

The remainder of this article will explain how to use the services found in ICMP.DLL to ping a remote machine. Figure A shows our sample program running.

**Figure A:** The example program shows the results of a ping.   


This technique comes with a caveat. Microsoft documentation on ICMP.DLL (such as it is) includes this warning: *Notice that the functions in icmp.dll are not considered part of the Win32 API and will not be supported in future releases. Once we have a more complete solution in the operating system, this DLL, and the functions it exports, will be dropped.* Granted, this is a warning that shouldn't be discarded lightly. On the other hand, Windows 2000 will still include ICMP.DLL, so it's not going away anytime soon. In addition, the Windows CE API includes the ICMP functions as part of the standard API. While Windows CE doesn't apply to C++Builder, the fact that the ICMP functions are available in CE would seem to indicate that ICMP.DLL will be around for some time.

Implementing ping using ICMP.DLL requires these steps:

|  |  |
| --- | --- |
|  | Create an import library file for ICMP.DLL. |
|  | Obtain the headers required to use the routines in ICMP.DLL. |
|  | Call the IcmpSendEcho() function to perform the ping. |

Of these steps, the second is the most formidable. We'll explain why in the following sections.

**Creating an import library file**

As with all Windows DLLs, you can choose to load ICMP.DLL either dynamically, using LoadLibrary(), or statically, using an import library. If you choose to load ICMP.DLL dynamically, you can skip this step. This article assumes static loading of the DLL. Creating an import library for ICMP.DLL is a trivial exercise. Use the C++Builder IMPLIB utility to create the import library file. Simply open a command prompt box and enter this line at the command prompt:

implib icmp.lib c:\winnt\system32\icmp.dll

Naturally, if you're using Windows 95 or 98, then you'll need to modify the command line to point to your Windows\System directory:

implib icmp.lib c:\windows\system\icmp.dll

When IMPLIB runs, you'll end up with a file called ICMP.LIB. You'll add this import library file to your C++Builder projects that use the ICMP routines. Adding the import library file to your C++Builder project is as simple as choosing Add to Project, and then choosing ICMP.LIB.

**Locating the ICMP headers**

As I said earlier, this is possibly the hardest step. It's a difficult step because the header files required for ICMP.DLL aren't readily available. The headers you need are the following:

|  |  |
| --- | --- |
|  | IPEXPORT.H |
|  | ICMPAPI.H |

The Microsoft Win32 SDK used to ship with ICMPAPI.H, so if you have an older version of the SDK, you can still obtain the headers from that source. The header could still be available on the latest SDK CD-ROM, so be sure to check there before embarking on a Web search for this header.

The more problematic header is IPEXPORT.H. This header contains structures required by ICMPAPI.H. Curiously enough, the Windows CE Platform SDK references these files so if you have that SDK, you can check to see if you already have the required files.

The quickest way to obtain these files is simply by searching the Internet. At the time of this writing you can find IPEXPORT.H and ICMPAPLH at

[**http://callamer.com/~jscarlet/JimWare/VBicmp/Ipexport.txt**](http://callamer.com/~jscarlet/JimWare/VBicmp/Ipexport.txt)

(There are probably other Web sites that have these files, but this is where I was able to find them.) Once you've obtained the ICMP headers, you can get on to the more serious business of writing your ping routine.

**Writing the ping routine**

Writing the ping routine itself requires less than a dozen lines of code. The first step is to obtain an ICMP handle by calling IcmpCreateFile(). This handle is needed later when you call IcmpSendEcho(). Here's the code for obtaining the file handle:

HANDLE hIcmp = IcmpCreateFile();

The next step requires making an int value out of an IP address. As you probably know, an Internet address is specified in the form 207.105.83.51. This value is translated into an integer value. The easy way to make an int from four byte values is by using Windows' MAKEWORD and MAKELONG macros. The following code illustrates this:

int addr = MAKELONG(

MAKEWORD(207, 105),

MAKEWORD(83, 51));

This assumes that you're starting with the IP address of the machine you want to ping, rather than a host name such as **www.bridgespublishing.com**.

At this point, you have an ICMP handle and an IP address in integer form. You can now proceed to call IcmpSendEcho() to perform the ping. IcmpSendEcho() is declared in ICMPAPI.H as follows:

IcmpSendEcho(

HANDLE IcmpHandle,

IPAddr DestinationAddress,

LPVOID RequestData,

WORD RequestSize,

PIP\_OPTION\_INFORMATION RequestOptions,

LPVOID ReplyBuffer,

DWORD ReplySize,

DWORD Timeout

);

Granted, this declaration looks a bit intimidating. It's not quite as bad as it looks, though, because you can simply set most of the parameters to 0. Here's how the call to IcmpSendEcho() looks:

int size = sizeof(icmp\_echo\_reply) + 8;

char\* buff = new char[size];

DWORD res = IcmpSendEcho(hIcmp, addr, 0, 0, 0, buff, size, 1500);

The first parameter of IcmpSendEcho() is the ICMP handle obtained in the call to IcmpCreateFile(). The second parameter is the IP address of the machine you're pinging. The third, fourth, and fifth parameters are used to specify request parameters. For simple ping operations you can set these parameters to 0. If you wanted to perform advanced ICMP operations, such as performing a trace route, you'd need to set these parameters to meaningful values.

The sixth parameter, ReplyBuffer, is a pointer to a buffer that will receive the reply information returned from the remote machine. The seventh parameter is used to specify the size of the return buffer. We used this code to create the buffer and set the size:

int size = sizeof(icmp\_echo\_reply) + 8;

char\* buff = new char[size];

Note that we set the buffer size to the size of an icmp\_echo\_reply structure, plus eight bytes. This is a safe buffer size. Typically, this is a large enough buffer to contain the reply information, based on the way we're calling IcmpSendEcho(). I've seen example code that allocated an extra eight *kilobytes* of buffer space; that seems like gross overkill, especially for what we're doing.

The final parameter of IcmpSendEcho() is used to specify a timeout value. We used a timeout value of 1500 milliseconds in this example, but you can use any timeout value you like. If you use a longer timeout value, a failed call to IcmpSendEcho() will take longer to complete. If you use a short timeout value, the operation may timeout before the call to IcmpSendEcho() completes, even though the server you're pinging may indeed be up and running.

If the server replies, IcmpSendEcho() will return a non-zero value (usually 1). If the server doesn't reply during the timeout period, IcmpSendEcho() will return 0. Keep in mind that the call to IcmpSendEcho() will fail if the server isn't responding, if you've provided an invalid IP address, or if you're not connected to the Internet. Further, understand that a "bad" IP address may contact a remote machine other than the one you intended to ping. In other words, you may get a reply from a server, but it may not be the server you're looking for!

After calling IcmpSendEcho() you need to free the ICMP handle you obtained in the call to IcmpCreateFile(). You do that by calling IcmpCloseHandle():

IcmpCloseHandle(hIcmp);

At this point the ping operation, whether successful or not, is complete.

**Examining the reply**

If IcmpSendEcho() succeeds, you may want additional information on the result of the ping. You can obtain the reply information by copying the first 24 bytes of the reply buffer to an instance of the icmp\_echo\_reply structure. You can then examine the members of the icmp\_echo\_reply structure for more information. icmp\_echo\_reply is declared as follows:

struct icmp\_echo\_reply {

// Replying address

IPAddr Address;

// Reply IP\_STATUS

unsigned long Status;

// RTT in milliseconds

unsigned long RoundTripTime;

// Reply data size in bytes

unsigned short DataSize;

// Reserved for system use

unsigned short Reserved;

// Pointer to the reply data

void FAR \*Data;

// Reply options

struct ip\_option\_information

Options;

};

Here again, this structure looks like it will take a lot of work to parse. In reality, you're probably only interested in the Status and RoundTripTime data members. You can ignore the rest of the data members unless you have specific needs beyond a simple ping operation. The Options member is a structure that contains the time to live in its Ttl member, so you might be interested in that information, as well.

To begin, we copy the first part of the reply buffer to an instance of icmp\_echo\_reply:

icmp\_echo\_reply reply;

memcpy(&reply, buff, sizeof(reply));

Now we can build a string to report the results of the ping to the user. The string might be constructed like this:

String rtt = reply.RoundTripTime;

String ttl = reply.Options.Ttl;

String S = "Reply from " + IPEdit->Text +

" time=" + rtt + "ms TTL=" + ttl + "ms";

This code assumes that you have an Edit component on a form, and that the Edit contains the IP address of the server you're pinging. When this code executes, the string will contain a value similar to this:

Reply from 207.105.83.51 time=220ms TTL=244ms

It's no coincidence that this string roughly resembles the text returned from the PING program itself. You may also want to examine the value of the Status member of the icmp\_echo\_reply structure to determine the cause of any ping failures. The Status member will be set to 0 if the ping operation succeeds, but will be a value of 11000 or greater if an error occurred. For example, IcmpSendEcho() might return 0, indicating the function was called successfully, but the status value may indicate some error condition. The IPEXPORT.H header defines the error values listed in Table A.

**Table A:** Error values

IP\_BUF\_TOO\_SMALL

IP\_DEST\_NET\_UNREACHABLE

IP\_DEST\_HOST\_UNREACHABLE

IP\_DEST\_PROT\_UNREACHABLE

IP\_DEST\_PORT\_UNREACHABLE

IP\_NO\_RESOURCES

IP\_BAD\_OPTION

IP\_HW\_ERROR

IP\_PACKET\_TOO\_BIG

IP\_REQ\_TIMED\_OUT

IP\_BAD\_REQ

IP\_BAD\_ROUTE

IP\_TTL\_EXPIRED\_TRANSIT

IP\_TTL\_EXPIRED\_REASSEM

IP\_PARAM\_PROBLEM

IP\_SOURCE\_QUENCH

IP\_OPTION\_TOO\_BIG

IP\_BAD\_DESTINATION

IP\_ADDR\_DELETED

IP\_SPEC\_MTU\_CHANGE

IP\_MTU\_CHANGE

IP\_UNLOAD

IP\_ADDR\_ADDED

If you've used the Windows' PING program, you may recognize some of these error constants as relating to messages PING returns in the case of a problem connecting to a remote machine. It's a simple matter to build an array of strings that map to these values (which fall in the range of 11001 to 11023). See Listing A for an example. Figure A shows the example program running. The program's main form contains an Edit component used to specify the IP address, a Button component that starts the ping operation, and a Memo component that displays the result of the ping. We haven't placed the code for this article on our Web site due to the fact that it requires IPEXPORT.H and ICMPAPI.H files, which we can't provide due to copyright considerations.

**Notes**

Listing A contains the code for the main unit of this article's example program.

**Listing A:** PingExU.cpp

#include <vcl.h>

#pragma hdrstop

#include "PingExU.h"

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm1 \*Form1;

// The ICMP.DLL headersyou'll have to obtain

// these from Microsoft or on the Internet.

extern "C" {

#include "ipexport.h""

#include "icmpapi.h"

}

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner)

{

}

// An array of error message strings.

String ErrorStrings[] = {

"Error Base",

"Buffer too small.",

"Destination net unreachable.",

"Destination host unreachable.",

"Destination protocol unreachable.",

"Destination port unreachable.",

"Out of resources.",

"Bad option.",

"Hardware error.",

"Packet too large.",

"Request timed out.",

"Bad request.",

"Bad route.",

"TTL expired in transit.",

"TTL expired REASSEM.",

"Param problem.",

"Source quench.",

"Option too large.",

"Bad destination.",

"Address deleted.",

"Spec MNU change.",

"MTU change.",

"Unload"

};

void \_\_fastcall

TForm1::PingBtnClick(TObject \*Sender)

{

// Obtain an ICMP handle.

HANDLE hIcmp = IcmpCreateFile();

if (!hIcmp) {

ShowMessage("Error getting ICMP handle");

return;

}

// Clear the Memo of any previous text.

Memo1->Lines->Clear();

Memo1->Lines->Add("Pinging " + IPEdit->Text);

// Parse the IP address in the Edit.

String S = IPEdit->Text;

int addr1 = Trim(S.SubString(1, 3)).ToInt();

int addr2 = Trim(S.SubString(5, 3)).ToInt();

int addr3 = Trim(S.SubString(9, 3)).ToInt();

int addr4 = Trim(S.SubString(13, 3)).ToInt();

// Make an int out of the IP address.

int addr = MAKELONG(

MAKEWORD(addr1, addr2),

MAKEWORD(addr3, addr4));

// Allocate a buffer for the reply info.

int size = sizeof(icmp\_echo\_reply) + 8;

char\* buff = new char[size];

// Show the user we'll be busy for a while.

Screen->Cursor = crHourGlass;

// Send the echo request three times to

// emulate what the PING program does.

for (int i=0;i<3;i++) {

Application->ProcessMessages();

// Call IcmpSendEcho().

DWORD res = IcmpSendEcho(hIcmp,

addr, 0, 0, 0, buff, size, 1500);

if (!res) {

Memo1->Lines->Add("Request timed out.");

continue;

}

// Prepare to report the status.

icmp\_echo\_reply reply;

memcpy(&reply, buff, sizeof(reply));

// If the status is non-zero then show the

// corresponding error message from the

// ErrorStrings array to the user.

if (reply.Status > 0)

Memo1->Lines->Add(

ErrorStrings[reply.Status - 11000]);

else {

// Build a string to report the results.

String rtt = reply.RoundTripTime;

String ttl = reply.Options.Ttl;

String S = "Reply from " + IPEdit->Text +

" time=" + rtt +"ms TTL=" + ttl + "ms";

// Add it to the memo.

Memo1->Lines->Add(S);

}

// Pause a second and then loop.

Sleep(1000);

}

// Close the ICMP handle.

IcmpCloseHandle(hIcmp);

// Restore the cursor.

Screen->Cursor = crArrow;

}

**Conclusion**

Pinging a remote machine using ICMP.DLL is a good way to determine whether users of your program are currently connected to the Internet, or whether a particular server you want to access is available. This method allows you to quietly check for the availability of a server without worrying about exceptions that many Internet components throw in the event of a connect failure.