Remote Monitoring

by Paul Warren

I have noticed a growing interest in remote monitoring technologies among my industrial clients. This is for good reason, since remote monitoring is a great labour saver. Recently, I was asked whether it was possible to monitor the output of an expensive instrument control and data acquisition package on an intranet.

Since there was no source code and no built-in web compatibility I was sceptical. I suggested upgrading the software to a version that provided HTML output, or possibly using a web camera to watch the screen (a solution I have actually seen implemented). The client wasn't interested in either suggestion, mainly because of the costs.

Thankfully, I didn't forget the question and a solution came to me while I was working on a completely different project.

Web Animation

There is a poor man's way of creating animations for the web, called server push, where a web application sends sequential output to the client browser. Closely related to server push is a technique called client pull, where the client periodically requests updates.

It was while investigating these two techniques that I realized it may be possible to monitor all kinds of foreign output. Imagine if you could get your data acquisition application to generously draw itself to a bitmap that you could then pass to the client browser by either server push or client pull. Potentially, we have the solution. But will it work in practice?

Server push seems to hold little promise for remote monitoring, since the server application needs to know when to shut itself down. With client pull, on the other hand, the browser controls the server application, which is just what we need for remote monitoring. For that reason I'll not discuss server push any further in this article.

Application Draw Thyself

Every visible window has a device context, a canvas in Delphi terminology. The device context is the window's visual representation. In order to get at this device context, we first need a handle to the window. The Windows API

```
hForeignWin := FindWindow('System Monitor', 'System Monitor');
try
  wDC := GetDC(hForeignWin);
  // copy DC here
finally
  ReleaseDC(hForeignWin, wDC);
end;
```

➤ *Above: Listing 1*

➤ *Below: Listing 2*

```
Bitmap := TBitmap.Create;
with Bitmap do
try
  hForeignWin := FindWindow('System Monitor', 'System Monitor');
  try
    GetClientRect(hForeignWin, SrcRect);
    Width := SrcRect.Right;
    Height := SrcRect.Bottom;
    wDC := GetDC(hForeignWin);
    BitBlt(Canvas.Handle, 0, 0, Width, Height, wDC, 0, 0, SRCCOPY);
  finally
    ReleaseDC(hForeignWin, wDC);
  end;
finally
  Bitmap.Free;
end;
```

➤ *Listing 3*

```
program CliePull;
{$APPTYPE CONSOLE}
uses
  SysUtils, Windows, Classes, Graphics, JPEG;
const
  Head: array[0..26] of char =
    'content-type: image/jpg'#13#10#13#10;
var
  hForeignWin: THandle;
  wDC: HDC;
  SrcRect: TRect;
  Bitmap: TBitmap;
  AStream: TMemoryStream;
  Buffer: array[0..8191] of Byte;
  BytesToSend: Integer;
begin
  Rewrite(Output);
  FileWrite(TTextRec(Output).Handle, Head, SizeOf(Head));
  Bitmap := TBitmap.Create;
  with Bitmap do
  try
    hForeignWin :=
      FindWindow('System Monitor', 'System Monitor');
    try
      SetForegroundWindow(hForeignWin);
      RedrawWindow(hForeignWin, nil, 0,
        RDW_INVALIDATE+RDW_UPDATENOW);
      GetClientRect(hForeignWin, SrcRect);
      Width := SrcRect.Right;
      Height := SrcRect.Bottom;
      wDC := GetDC(hForeignWin);
      BitBlt(Canvas.Handle, 0, 0, Width, Height, wDC, 0, 0,
        SRCCOPY);
      with TJPEGImage.Create do
      try
        Assign(Bitmap);
        AStream := TMemoryStream.Create;
        try
          SaveToStream(AStream);
          AStream.Seek(soFromBeginning, 0);
          while AStream.Position < AStream.Size do begin
            BytesToSend :=
              AStream.Read(Buffer, SizeOf(Buffer));
            FileWrite(TTextRec(Output).Handle, Buffer,
              BytesToSend);
          end;
        finally
          AStream.Free;
        end;
      finally
        Free;
      end;
    finally
      ReleaseDC(hForeignWin, wDC);
    end;
  finally
    Bitmap.Free;
  end;
end.
```

```
<HTML>
<HEAD><TITLE>Client Pull Demo</TITLE>
<META http-equiv="Refresh" content=10>
</HEAD>
<BODY>
<H1>Client Pull Demo</H1>
<! -Note you need to put the address of your server here>
<IMG SRC="http://127.0.0.1/cgi-bin/cliepull.exe?">
</body>
</html>
```

➤ *Listing 4*

function `FindWindow` returns a handle to the window matching the parameters. Assuming we can find either the class type (using Winsight), or the caption, obtaining the handle is a relatively trivial task.

Armed with a valid handle we can obtain the device context (DC) using either `GetDC` or `GetWindowDC`, depending on whether we want the entire area or just the client area. Listing 1 shows the basic steps of grabbing a DC for a window.

We now have to copy the DC so that it can be saved to a file or streamed to the client, whichever is most convenient. There are a number of Windows functions for working with device contexts, but the easiest way is to `BitBlt` the DC to a Delphi-managed canvas.

Listing 2 shows the code to create a `TBitmap` and `BitBlt` the DC to it.

The bitmap has to be converted to a JPEG before it can be used on the intranet. Since a `TJPEGImage` has a `bitmap` property, we can use the `Assign` method to copy our bitmap directly to the JPEG. It's then a simple matter to stream the JPEG to the client.

The complete code is shown in Listing 3. Note the use of the `FileWrite` procedure, which allows us to write blocks of data to the standard output for greatly improved performance.

You'll notice I have used the Windows system monitor for demonstration purposes, since it should be available to everyone. Figure 1 shows my desktop with both the system monitor and browser together. The complete source is available on this month's disk, along with a compiled version.

Enter HTML

Nowhere have I talked about or implemented any form of animation. The HTML document in which I'll embed the link to this application is the animation engine. By including the META tag `<META http-equiv="Refresh" content=10>` in the HTML the client browser will refresh the document automatically every 10 seconds (in this case). Listing 4 shows the HTML for this simple test page.

Limitations

There are some limitations to this technique. The window you are trying to copy must be in the foreground. To be certain it is I added the `SetForegroundWindow` call. This means that the window being copied will move to the front of all other windows and grab the focus. Anyone trying to work on the server while this was happening would soon be very annoyed. For this reason you should only try this on a server that is not being actively used. In a corporate environment where a process must be monitored and where the alternatives are expensive, it is quite useful.

One other effect shows up when a screen saver or other graphic intensive program is in the foreground. The window you are trying to copy may not redraw itself before the DC has been copied and streamed to the client. A call to

```
RedrawWindow(
  hForeignWin, nil, 0,
  RDW_INVALIDATE+RDW_UPDATENOW);
```

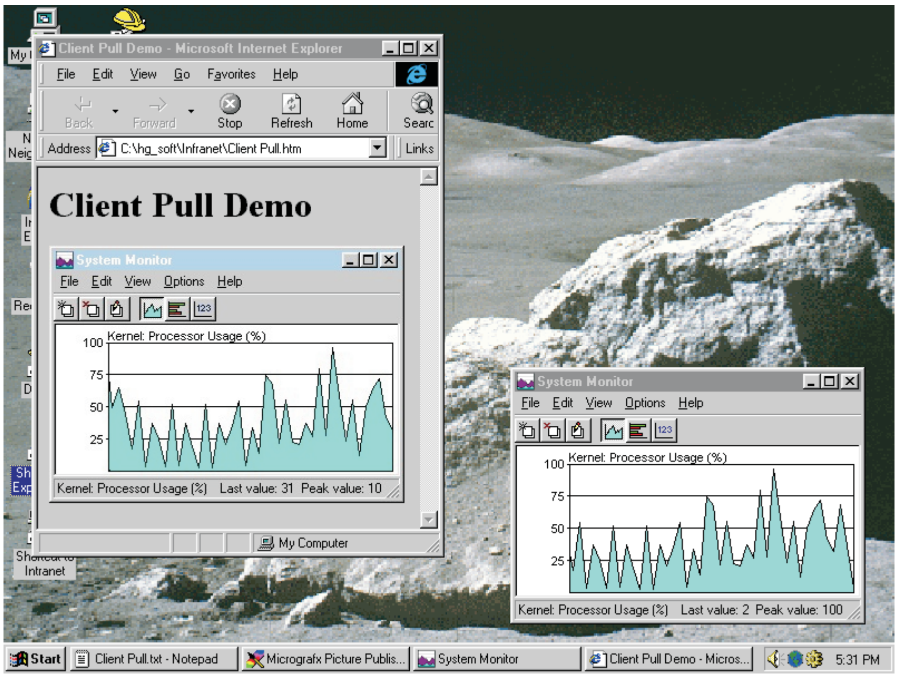
seems to eliminate this problem by forcing a repaint before returning.

Conclusions

There are many sophisticated methods of remote monitoring, but most are expensive in that they require new software, hardware or both. Using client pull and Delphi you can easily monitor the output of any visible window making many remote monitoring tasks feasible. It certainly satisfied the aforementioned client, who can now check the instrument output via his intranet 24 hours a day, 7 days a week, from any location with dial-up or cable access.

Paul Warren runs HomeGrown Software Development in Langley, British columbia, Canada and can be contacted at hg_soft@uniserve.com

➤ *Figure 1*


Client Pull Demo - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search
Address C:\hg_soft\Infranet\Client Pull.htm
Links
Client Pull Demo
System Monitor
File Edit View Options Help
Kernel: Processor Usage (%)
100
75
50
25
Kernel: Processor Usage (%) Last value: 31 Peak value: 10
My Computer
System Monitor
File Edit View Options Help
Kernel: Processor Usage (%)
Kernel: Processor Usage (%) Last value: 2 Peak value: 100
Start
Client Pull.txt - Notepad
Micrografx Picture Publis...
System Monitor
Client Pull Demo - Micros...
5:31 PM