How to get operating system version information (part 2 of 6)

Common Windows API OS information

Common Windows API OS Information

Windows provides the *GetVersionEx* API function that fills a structure with version information about the underlying operating system. This function works for all 32 bit versions of Windows. In Delphi the structure that contains the version information is defined in the `Windows` unit as *TOSVersionInfo* – see *Listing 1* below.

```
type
  ...
  _OSVERSIONINFOA = record
    dwOSVersionInfoSize: DWORD;
    dwMajorVersion: DWORD;
    dwMinorVersion: DWORD;
    dwBuildNumber: DWORD;
    dwPlatformId: DWORD;
    szCSDVersion: array[0..127] of AnsiChar;
  end;
  ...
  _OSVERSIONINFO = _OSVERSIONINFOA;
  TOSVersionInfoA = _OSVERSIONINFOA;
  ...
  TOSVersionInfo = TOSVersionInfoA;
```

Listing 1

The structure's fields are explained in *Table 1*:

Fields of TOSVersionInfo

Field	Description
dwOSVersionInfoSize	Size of the structure in bytes.
dwMajorVersion	Major version of the operating system. Possible values are: 4 Windows NT 4.0, Windows Me, Windows 98 or Windows 95. 5 Windows Server 2003, Windows XP or Windows 2000.
dwMinorVersion	Minor version of the operating system. Possible values are: 0 Windows 2000, Windows NT 4.0 or Windows 95. 1 Windows XP. 2 Windows Server 2003. 10 Windows 98. 90 Windows Me.
dwBuildNumber	Build number of the operating system. On Windows 9x, the low order word contains the build number and the high order word contains the major and minor version numbers.

dwPlatformId	Operating system platform. This member can take one of the following values: *VER_PLATFORM_WIN32_NT* Windows NT OSs, i.e. Windows Server 2003, Windows XP, Windows 2000 or Windows NT. *VER_PLATFORM_WIN32_WINDOWS* Windows 9x OSs, i.e. Windows Me, Windows 98 or Windows 95. *VER_PLATFORM_WIN32s* Win32s on Windows 3.1. (This will not occur on Delphi since Delphi applications will not run on Win32s).
szCSDVersion	Buffer containing a null terminated string identifying any installed service pack. On Windows NT OSs this string is the actual service pack name while on Windows 9x it is a single letter code that is interpreted as follows: B or C Represents Windows 95 OSR2. A Represents Windows 98 SE (Second Edition).

Table 1

Listing 2 shows how to use *GetVersionEx* to fill a *TOSVersionInfo* structure with operating system information:

```
var
  OSVI: TOSVersionInfo;
begin
  ...
  FillChar(OSVI, SizeOf(OSVI), 0);
  OSVI.dwOSVersionInfoSize := SizeOf(OSVI);
  if not GetVersionEx(OSVI) then
    raise Exception.Create('Error calling GetVersionEx');
  ...
end;
```

Listing 2

Here we zero the *TOSVersionInfo* structure then set the *dwOSVersionInfoSize* field to the size of the structure – a common Windows API idiom. We then call *GetVersionEx*, raising an exception if the function returns false. If the function returns true then the information stored in the structure is valid.

We would normally go on to use the fields of *TOSVersionInfo* in our program. However, Delphi steps in to make this process easier, as can be seen below.

Delphi RTL Support

Delphi's `SysUtils` unit provides some global variables that make accessing basic operating system version information very straightforward, saving the need to call *GetVersionEx* ourselves. The variables provide the same information as the *TOSVersionInfo* structure, as can be seen in *Table 2*.

Delphi's Win32XXX Global Variables

Variable	Field	Notes
Win32Platform	*dwPlatformId*	Operating system platform. Can take one of the *VER_PLATFORM_** values noted in *Table 1*.
Win32MajorVersion	*dwMajorVersion*	Major version of the operating system.
Win32MinorVersion	*dwMinorVersion*	Minor version of the operating system.
Win32BuildNumber	*dwBuildNumber*	Build number of the operating system. On Windows 9x systems the high order word is zeroed.
Win32CSDVersion	*szCSDVersion*	Buffer containing a null terminated string identifying any installed service pack. For more information see *Table 1*.

Table 2

Delphi initializes the global variables in the `SysUtils` unit's initialization section. It does this by calling the *InitPlatformId* procedure which is shown in *Listing 3* below:

```
procedure InitPlatformId;
var
  OSVersionInfo: TOSVersionInfo;
begin
  OSVersionInfo.dwOSVersionInfoSize := SizeOf(OSVersionInfo);
  if GetVersionEx(OSVersionInfo) then
    with OSVersionInfo do
    begin
      Win32Platform := dwPlatformId;
      Win32MajorVersion := dwMajorVersion;
      Win32MinorVersion := dwMinorVersion;
      if Win32Platform = VER_PLATFORM_WIN32_WINDOWS then
        Win32BuildNumber := dwBuildNumber and $FFFF
      else
        Win32BuildNumber := dwBuildNumber;
      Win32CSDVersion := szCSDVersion;
    end;
end;
```

Listing 3

Setting up *TOSVersionInfo* and calling *GetVersionEx* should be familiar from our previous discussion. Once Delphi has the version information the global variables are simply assigned the value of the associated *TOSVersionInfo* field. The only exception to this is with *Win32BuildNumber* which, as noted in *Table 2*, has its high word zeroed in Windows 9x systems.

Platform

Listing 3 detects the platform by testing *Win32Platform*. We will learn more about detecting the platform below.

OS Version Information Class

Having learned about the basic Windows and Delphi support for getting OS version information we are now ready to create a new static class, *TOSInfo*, that exposes OS version information in a user friendly way. We will use only the information provided by Delphi's global variables in this class's methods.

We begin by starting a new unit named `UOSInfo` and declaring a new, empty, class in the unit's interface section:

```
type
  TOSInfo = class(TObject)
  public
  end;
```

Listing 4

In the following subsections we will add methods to the class.

Finding the OS Platform

The most basic distinction between Windows operating systems is that between *platforms*. We can find the platform by examining Delphi's *Win32Platform* global variable. The value of the variable will be one of the integer constants described in the *dwPlatformId* entry in *Table 1*. These constants are defined in Delphi's `Windows` unit.

Testing platforms

Users should not assume that when *IsWin9x* returns false we must have an NT platform – to be sure the *IsWinNT* method should be tested instead and vice versa.

Our class will have two methods to test the platform – one to detect Windows 95 (*IsWin9x)* and one to test for the NT platform (*IsWinNT)*. Since Delphi programs will not run on Win32s we will not provide a method to detect that platform.

We can add the appropriate declarations to *TOSInfo*'s interface section and implement the methods as in *Listing 5*:

```
class function TOSInfo.IsWin9x: Boolean;
begin
  Result := (Win32Platform = VER_PLATFORM_WIN32_WINDOWS);
end;

class function TOSInfo.IsWinNT: Boolean;
begin
  Result := (Win32Platform = VER_PLATFORM_WIN32_NT);
end;
```

Listing 5

OS Product

Now we have code that can check the platform we can move on to detecting which operating system product our applications are running on. There are various combinations of major and minor version numbers that, along with the platform, determine the product. These are summarised in *Table 3*.

Mapping of OS version numbers to Windows products

Major Version	Minor Version	Windows 9x Platform	Windows NT Platform
4	0	Windows 95	Windows NT 4
4	10	Windows 98	-
4	90	Windows Me	-
5	0	-	Windows 2000
5	1	-	Windows XP
5	2	-	Windows Server 2003

Table 3

We will design our new method, *Product*, to return an enumerated value that indicates the underlying operating system. We define this enumeration in the interface section of `UOSInfo.pas` as follows:

```
type
  TOSProduct = (
    osUnknown,          // Unknown OS
    osWin95,            // Windows 95
    osWin98,            // Windows 98
    osWinMe,            // Windows Me
    osWinNT4,           // Windows NT 4.0
    osWin2000,          // Windows 2000
    osWinXP,            // Windows XP
    osWinServer2003     // Windows Server 2003
  );
```

Listing 6

Now we can add our *Product* method to *TOSInfo*. The method's implementation is shown in *Listing 7*:

```
class function TOSInfo.Product: TOSProduct;
begin
  Result := osUnknown;
  if IsWin9x then
  begin
    case Win32MajorVersion of
      4:
      begin
        case Win32MinorVersion of
          0: Result := osWin95;
          10: Result := osWin98;
          90: Result := osWinMe;
        end;
      end;
    end;
  end
```

```
  else if IsWinNT then
  begin
    case Win32MajorVersion of
      4:
      begin
        case Win32MinorVersion of
          0: Result := osWinNT4;
        end;
      end;
      5:
      begin
        case Win32MinorVersion of
          0: Result := osWin2000;
          1: Result := osWinXP;
          2: Result := osWinServer2003;
        end;
      end;
    end;
  end;
end;
```

Listing 7

This method is quite simple. It first sets an "unknown" return value in case the method fails to find the OS. We then take different branches according to the platform. Within each branch we test for valid combinations of major and minor versions, per *Table 3*, and return the value representing the OS.

Build Number

Once we know which product we're dealing with we may also want to know its build number We will therefore add a *BuildNumber* method to our class. Its implementation is provided in *Listing 8* and simply returns the value of Delphi's *Win32BuildNumber* variable. (Remember, Delphi took care of masking out the most significant word when on the Windows 9x platform, so we don't need to do that here).

```
class function TOSInfo.BuildNumber: Integer;
begin
  Result := Win32BuildNumber;
end;
```

Listing 8

Service Pack

The final method we will implement here is *ServicePack*. Earlier, in *Table 1*, it was noted that Windows provides the full name of NT platform service packs but only supplies a code letter on the Windows 9x platform. *Listing 9* show how we use Delphi's *Win32CSDVersion* variable to help us implement the method.

```
class function TOSInfo.ServicePack: string;
begin
  Result := '';
  if IsWin9x then
  begin
    if Win32CSDVersion <> '' then
    begin
      case Product of
        osWin95:
          if UpCase(Win32CSDVersion[1]) in ['B', 'C'] then
            Result := 'OSR2';
        osWin98:
          if UpCase(Win32CSDVersion[1]) = 'A' then
            Result := 'SE';
      end;
    end;
  end
  else if IsWinNT then
    Result := Win32CSDVersion;
end;
```

Listing 9

This method begins by assuming there is no service pack installed, so it sets the default result to the empty string. We then detect the platform. For Windows 9x we look for the characters 'A', 'B' or 'C' in *Win32CSDVersion* and return the required service pack name if they are found. For the NT platform we simply return the value of *Win32CSDVersion* since this will either be the empty string or will contain the service pack name in full.

NT 4 Complications

This is not quite the whole story – this method won't detect Windows NT 4 Service Pack 6a. We'll address this problem *later in the article*.

We have now got all the information we can by using the Windows API that is common to all versions of Windows. In the *next section* we will investigate further information that is specific to certain Windows platforms and OS versions.
