How to get operating system version information (part 5 of 6)

Windows XP Tablet, Media Center and 64 bit editions

More XP Related OS Information

We now turn our attention to providing a little more information about Windows XP editions. XP comes in several flavours, some of which we have already dealt with and others that we haven't examined yet. In particular we will see how to detect:

- XP Media Centre Edition
- XP Tablet Edition
- 64 bit versions of XP

XP Media Center and Tablet Editions

Surprisingly, we find out about these systems using the *GetSystemMetrics()* API call by passing the *SM_MEDIACENTER* or *SM_TABLETPC* flags to the function. A non-zero return from *GetSystemMetrics()* indicates the specified edition is present.

64 Bit Versions of XP

With the advent of 64 bit versions of Windows we may need to detect if our application is running on such an operating system.

Any application compiled with current versions of Delphi (up to Delphi 2006 at the time of writing) is 32 bit. 64 bit Windows runs 32 bit applications in the WOW64 subsystem. So, we need a way to find out if our application is running in this subsystem. Windows XP provides a function to 32 bit applications named *IsWow64Process* that provides this information.

IsWow64Process is not available to OSs earlier than Windows XP. Consequently we must link to this function dynamically. Static linking would cause our application to fail on all OSs but XP.

OS Version Information Class: Version 4

Having explored how we can detect XP Media Center and Tablet editions and find if we are running on 64 bit Windows, we can now update our *TOSInfo* static class to take account of these XP specific issues.

Supporting XP Media Center and Tablet Editions

We will add two new public static methods to *TOSInfo* that detect XP Media Center and Tablet editions – *IsMediaCenter* and *IsTablet*. We will also modify the *Edition* method to detect these editions.

Listing 28 shows the implementation of the new *IsMediaCenter* and *IsTablet* methods. They simply call *GetSystemMetrics* with the appropriate flag.

```
class function TOSInfo.IsMediaCenter: Boolean;
begin
  Result := GetSystemMetrics(SM_MEDIACENTER) <> 0;
end;

class function TOSInfo.IsTablet: Boolean;
begin
```

```
  Result := GetSystemMetrics(SM_TABLETPC) <> 0;
end;
```

Listing 28

Unfortunately Delphi does not define *SM_MEDIACENTER* and *SM_TABLETPC*, so we will need to define these in our code. We may as well make these publicly available, so add the constants to the interface section of `UOSInfo.pas` as shown in *Listing 29*.

```
const
  SM_MEDIACENTER = 87;  // Detects XP Media Center Edition
  SM_TABLETPC = 86;     // Detects XP Tablet Edition
```

Listing 29

Armed with the two new *IsMediaCenter* and *IsTablet* methods we can now modify *TOSInfo.Edition* to detect XP Media Center and Tablet editions. *Listing 30* shows the changes that need to be made to the method.

```
class function TOSInfo.Edition: string;
begin
  Result := '';
  if IsWinNT then
  begin
    if Win32HaveExInfo then
    begin
      // Test for edition on Windows NT 4 SP6 and later
      if IsServer then
      begin
        ...
      end
      else
      begin
        // Workstation type
        case Product of
          ...
          osWinXP:
          begin
            if IsMediaCenter then
              Result := 'Media Center Edition'
            else if IsTablet then
              Result := 'Tablet PC Edition'
            else if CheckSuite(VER_SUITE_PERSONAL) then
              Result := 'Home Edition'
            else
              Result := 'Professional';
          end;
        end;
      end;
    end
    else
    begin
      ...
    end;
  end;
end;
```

Listing 30

Here we simply update the Windows XP workstation section of the code to test for Media Center and Tablet editions. If we find neither we check for XP Home and Professional as before.

Detecting 64 Bit Windows

Our approach will be to create a new method of our *TOSInfo* static class – *IsWOW64* – that is a wrapper round the *IsWow64Process* API function we discussed above. The new method, shown in *Listing 31*, returns true when running on 64 bit windows and false otherwise.

```
class function TOSInfo.IsWOW64: Boolean;
type
  TIsWow64Process = function( // Type of IsWow64Process API fn
    Handle: THandle;
    var Res: BOOL
```

```
  ): BOOL; stdcall;
var
  IsWow64Result: BOOL;              // result from IsWow64Process
  IsWow64Process: TIsWow64Process;  // IsWow64Process fn reference
begin
  // Try to load required function from kernel32
  IsWow64Process := GetProcAddress(
    GetModuleHandle('kernel32'), 'IsWow64Process'
  );
  if Assigned(IsWow64Process) then
  begin
    // Function is implemented: call it
    if not IsWow64Process(GetCurrentProcess, IsWow64Result) then
      raise Exception.Create('Bad process handle');
    // Return result of function
    Result := IsWow64Result;
  end
  else
    // Function not implemented: can't be running on Wow64
    Result := False;
end;
```

Listing 31

TIsWow64Process prototypes the *IsWow64Process* API function. The return value of this function indicates whether the it succeeded or failed – it does not indicate whether the WOW64 subsystem is present. In fact the function's *Res* parameter is used for this purpose – it is set true if running on WOW64 and false otherwise.

Our method tries to load *IsWow64Process* from `kernel32.dll` and stores a pointer to the function in the *IsWow64Process* local variable. If we succeed in loading the function we call it, passing the current process handle and storing the function's output value in *IsWow64Result*. In the unlikely event that the function fails (i.e. returns false) we raise an exception. Otherwise the value of *IsWow64Result* is returned.

Should we fail to load *IsWow64Process* we know that we are running on an OS prior to Windows XP that does not have a 64 bit support. Therefore we return false.

Our examination of how to get operating system version information is now complete. In the *last section* we will wrap up the article.
