64-bit Application Development

The two mainstream 64-bit processor architectures available today are:

- The Intel Itanium (IA64) processor family, which is based on Explicitly Parallel Instruction Computing (EPIC)
  There are variations of this architecture but they all commonly allow simultaneous execution of multiple instructions. Thus while designing application for such platform it should implement SMT (Simultaneous Multiple Threaded) architecture for efficient performance.

- x64 processors like the AMD Opteron and Intel Xeon with Extended Memory 64 Technology (EM64T) which are based on extensions to the x86 instruction set. The x86 instruction set was intended for 32 processors (which can execute 32 bit instructions) but with x64 processors having ability to execute 64 bit instructions; the x86 instruction set was extended with introduction of 64 bit instructions which can replace multiple 32 bit instructions.

In order to support 32-bit personal productivity applications needed by software developers and administrators in 64-bit versions of Windows, Microsoft provides WOW64 as an integral part of all its 64-bit OS (Windows XP 64-bit Editions, Windows Server 2003 SP1 x64 Edition and Windows Vista 64-bit Editions).

WOW64 (Windows-on-Windows 64-bit) (understood as Windows 32-bit on Windows 64-bit) allows the 64-bit operating system to seamlessly and transparently redirect requests from 32-bit applications to 32-bit-specific resources. WoW64 isolates 32-bit applications from 64-bit applications, including preventing file and registry collisions. It supports console, GUI applications and service applications. Although it does provides interoperability across the 32/64 boundary for scenarios such as cut and paste and COM, but**, 32-bit processes cannot load 64-bit DLLs and 64-bit processes cannot load 32-bit DLLs**.

The WoW64 subsystem directs all 32-bit applications to a separate File System and Registry. This guarantees that a 32 bit application can not access a 64 bit application registry key or system directory for resources. Similarly, 64 bit applications can not access 32-bit file systems or registry entries.

File System Redirection

%systemroot%\system32\ is mapped to %systemroot%\syswow64

Registry Redirection

HKEY_LOCAL_MACHINE\Software is mapped to
HKEY_LOCAL_MACHINE\Software\Wow6432Node

What benefits?
In general, 64-bit architecture would double the amount of data a CPU can process per clock cycle. Users would note a performance increase because a 64-bit CPU can handle more memory and larger files. One of the most attractive features of 64-bit processors is the amount of memory the system can support. 64-bit architecture will allow systems to address up to 16 terabyte of memory. In 32-bit desktop systems, you can have up to 4GB of RAM (provided your motherboard that can handle that much RAM). Additionally, using a 64-bit server means organizations can support more simultaneous users on each server potentially removing the need for extra hardware as one 64-bit server could replace the use of several 32-bit servers on a network.

Porting Database
Some of the major database management systems, such as Oracle and SQL Server, to name just two, have 64-bit compatible versions.

Porting .Net Application
One of the advantages of using a managed environment is that porting your application is relatively simple. In many cases, all you need to do is make some changes to your application configuration. Of course, you have to have the 64-bit version of the .NET Framework and a few other things to make the application run under the 64-bit environment, but you don't have to do nearly as much as you might expect for a native code application.

.NET Framework Considerations
Most developers know that compiling a .NET application doesn't produce native code. What you get instead is a file containing tokens that the Common Language Runtime (CLR) reads and turns into native code. Consequently, you don't have to worry about cross-compilation when working with managed applications, but you do need to worry about having the right runtime support.

Developing for the .NET Framework
You might wonder why you should have to do anything at all to make your .NET application work in the 64-bit environment. The short answer is that your old 32-bit version will very likely run fine as long as it doesn't include anything odd such as PInvoke calls or reliance on Component Object Model (COM) components. However, using the 32-bit version in the 64-bit environment isn't the best way to do things because you miss a lot of the benefits of working with 64 bits.

A managed application is actually a series of tokens that CLR interprets, but the tokens that the compiler creates are important. The 64-bit environment provides a lot of advantages to your application such as access to a 16 TB memory space and larger integers.

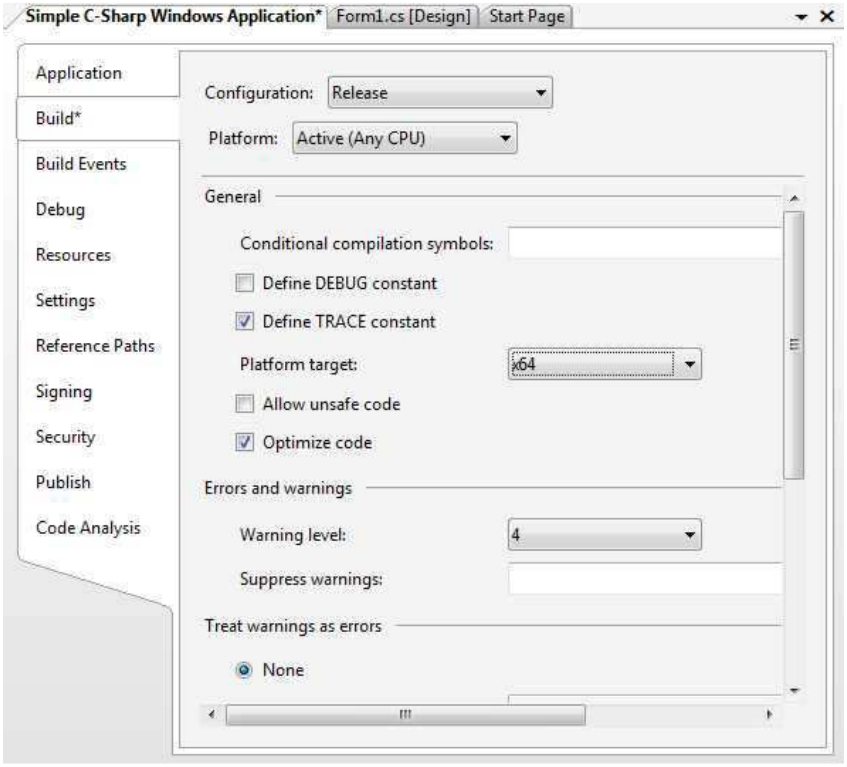

Figure 1. Set the .NET Build Environment to Use 64 Bits

To make use of these features, you must as a minimum optimize the compiler output by changing your application settings. You can perform this task by right clicking the solution in Solution Explorer and choosing Properties from the context menu. Choose the Build tab and you'll see settings similar to those shown in Figure 1. Notice that the Platform Target field contains x64 and that the Optimize Code option is checked.

As an alternative to changing the release settings for your project, you can always create a new configuration. Simply click Configuration Manager in the Solution Configurations drop down list box on the Standard toolbar. When you see the Configuration Manager window, add a new entry for the 64-bit version of your application as shown in Figure 2. To perform this task, select <new…> from the Active Solution Configuration dialog box and type a Name for the new configuration. You can save time by copying the current settings from the Debug or Release versions and changing just the settings you need for the 64-bit version of your application.

Obtaining the 64-bit Version of the .NET Framework

When working with 64-bit applications, you need a 64-bit version of the .NET Framework, which doesn't come with the Visual Studio 2005 package. You can download the 64-bit version of the .NET Framework from the 64-bit .NET Framework Web site..

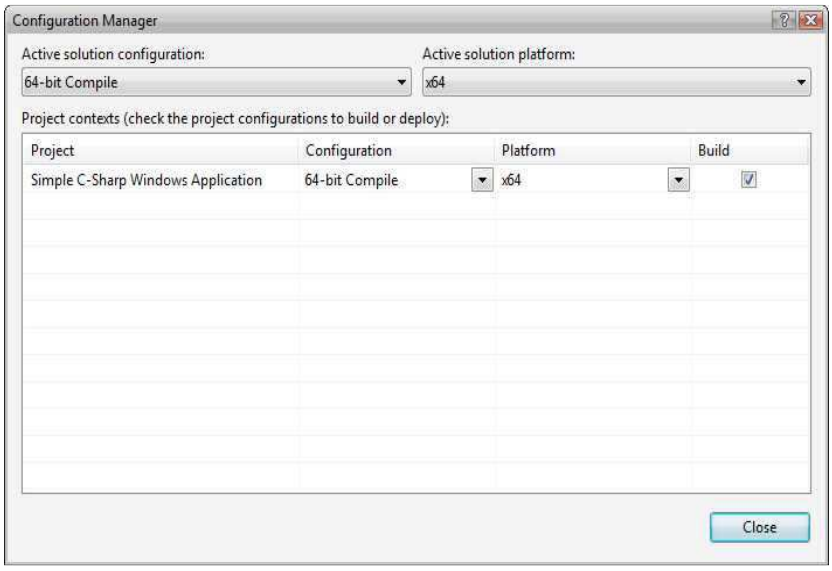

Figure 2. Create One or More 64-bit Configurations for Your Application

You might have noticed that you must use the .NET Framework 2.0 for 64-bit development. Microsoft made changes between the 1.0, 1.1, and the 2.0 versions of the .NET Framework that could cause problems for your 64-bit application that have nothing to do with running in a 64-bit environment. Make sure you read about the breaking changes in the .NET Framework before you assume that an error is due to working in the 64-bit environment.

Obtaining 64-bit .NET Development Tools

A good many of the development tools you use in the 32-bit environment will work fine in the AMD 64-bit environment. However, they'll generally have to run under Windows-on-Windows 64-bit (WOW64). Depending on the utility, you might not notice much of a difference in performance and probably won't see any difference in functionality. That's because most applications spend their time waiting for the developer to do something. Of course, your 32-bit tools don't consider special 64-bit needs such as emulation. Fortunately, Microsoft has answered this problem by providing a rich assortment of 64-bit tools.

Obfuscating an Assembly

Obfuscating--the process of making your application code unreadable to prying eyes--works the same in the 64-bit environment as it does in the 32-bit environment. The application that performs the obfuscation will run under WOW64 in most cases, but that doesn't affect the output. Consequently, you shouldn't experience any difficulties using obfuscation with your 64-bit application.

Special Problems for 64-bit .NET Applications

Just because the Visual Studio IDE makes it so easy to create a version of your application that is optimized for the AMD 64-bit environment doesn't mean you'll have clear sailing. In fact you'll encounter a number of potential problems with your

application whenever you use a special feature. The following sections describe some of the major issues you need to consider.

PInvoke Requirements
Generally, you aren't going to have many problems working with PInvoke in the 64-bit environment. Although you should change 32-bit calls to 64-bit calls whenever possible to improve performance, the 32-bit calls will still work fine with a small performance penalty. You can still perform all of the tricks that you did in the past when using Win32 API calls because the Win32 API hasn't changed--it simply goes through the WOW64 layer. You can read more about the ramifications of WOW64 and how it affects .NET applications.

However, everything changes when a device driver comes into play. If you're making any direct calls whatsoever, you need to consider changes that the 64-bit environment requires. "First Look: Windows XP 64-Bit Edition for AMD64," although posted at a gaming Web site, demonstrates a very simple reason for this change. Even though WOW64 shields your application from Win32 API call changes, it can't shield you from device driver changes. Consequently, when your application makes a call to a device driver such as a Universal Serial Bus (USB) device, you have to make changes to your code.

COM Requirements
Because COM components rely on native code, you may find that you need 64-bit equivalents. In fact, it's a good idea to check for the 64-bit equivalent or take the opportunity to wean off the COM components when making the move to a 64-bit application. As a general rule of thumb, if the COM component comes as part of Windows or as part of an application that runs under the 64-bit version of Windows without any problem, you should probably be safe. However, stand-alone components or those that provide special functionality are very likely going to fail. Even if you do get the application to run, you need to perform extensive testing.

If you need additional information about moving your managed application to the 64-bit environment, you can find it at Getting Started with the 64-Bit .NET Framework.

Setting up the environment for 64 bit development

Although 64 bit applications can be build on 32 bit platforms using cross platform development functionality of VS 2005 but the application can neither be run nor debugged on such platform. Although, remote debugging can serve as a workaround for such platform but that will still require running the application on a 64 bit environment. Building 64 bit application of 32 bit platform will require installation of VS 2005 with x64 development tools option selected. Further change in project configuration (selecting x64 debug or release option) for compilation for 64 bit platform and another change in solution settings selecting x64 as the target platform for linking for 64 bit platform.

For enabling remote debugging option refer: http://msdn2.microsoft.com/en-us/library/ms184681.aspx

Developing 64 bit application on a 64 bit platform will be fairly simple except for the fact that Microsoft has not yet released any virtualization application for 64-bit guest operating system. The 64 bit Microsoft Virtual PC 2007 although installs on a 64-bit host but fails to load a 64-bit guest operating system. So that leaves us with the option of developing 64 bit applications on the host machine directly.