Convert Integers to Byte Arrays

by Mattias Sjögren and Juval Löwy

Technology Toolbox

- ☐ VB.NET
- ☑ C#
- ☐ SQL Server 2000
- ☐ ASP.NET
- ☐ XML
- ☐ VB6

Go Online!

Use these Locator+ codes at **www.visualstudiomagazine.com** to go directly to these resources.

Download
VS0303QA Download the code for this article. It includes the WndProc solution, with two WinForms applications—MessageTracer and MessageSender—as well as the Util class in the WinFormsEx class library.

Discuss
VS0303QA_D Discuss this article in the C# forum.

Read More
VS0303QA_T Read this article online.

VS0211FF_T "Deploy Your .NET App" by Fabio Claudio Ferracchiati

VSEP0112CS_T "Dig Into C#" by Bill Wagner

VSEP0112JP_T "Discussion with Jeff Prosise" by Bill Wagner

Q: Convert Integers to Byte Arrays

How can I convert an int to a byte array, and vice versa, in C#?

A:

You can accomplish this in at least three ways (see Listing 1). A class called BitConverter in the System namespace of the .NET Framework class library is dedicated to this task. It has a GetBytes method, with overloads for most simple types, which returns a byte array with the in-memory representation of the value you pass in. It also has a number of To*TypeName* methods for doing the opposite—converting a byte array to a primitive type:

```
byte[] b = BitConverter.GetBytes(
   0xba5eba11 );
//{0x11,0xba,0x5e,0xba}
uint u = BitConverter.ToUInt32(
   new byte[] {0xfe, 0x5a, 0x11,
   0xfa},0 ); // 0xfa115afe
```

An important thing to keep in mind when you use the BitConverter class is that its behavior depends on the *endianness* of the hardware architecture the code runs on—that is, the order in which the bytes of an integer are stored in memory. Problems can arise if you persist the bits to a file format that should be readable on many different platforms. The BitConverter has a public IsLittleEndian field that you can check to determine how it will behave, but unfortunately it doesn't provide a way for you to change it.

It's also possible to do without the BitConverter class and do the work manually with some bit shifting instead:

```
b = new byte[] {0xfe,0x5a,0x11,0xfa};
u = (uint)(b[0] | b[1] << 8 |
   b[2] << 16 | b[3] << 24);
b[0] = (byte)(u);
b[1] = (byte)(u >> 8);
b[2] = (byte)(u >> 16);
b[3] = (byte)(u >> 24);
```

This way you can get around the endian problem, because you have full control over which byte ends up where.

Finally—if you don't mind using unsafe code—you can do a direct memory copy by casting a pointer to the byte array to a pointer to the integer type, then dereferencing it:

```
unsafe {
  fixed ( byte* pb = b )
  u = *((uint*)pb);
}
```

The exact result of this operation depends on the hardware the code runs on, as it does with the BitConverter.

If you're going to do a lot of this kind of conversion—in a tight loop, say—and want the best performance, I suggest you use one of the last two methods. The BitConverter tends to be somewhat slower, although the difference isn't big. *—M.S.*

Q: Reference Assemblies in Visual Studio

I've used Gacutil.exe to install my assembly in the Global Assembly Cache (GAC), but it's not listed with the other assemblies when I try to reference it from a project in VS.NET. What else must I do to be able to reference it, without using the Browse button?

A:

The list of assemblies that appears in Visual Studio's Add Reference dialog box isn't retrieved from the GAC; that's a common mis-

C# • Convert Between Integers and Byte Arrays

```
using System;

// Compile with /unsafe option

class Test
{
  static void Main()
  {
    uint u = 0xba5eball;
    byte[] b = new byte[] {0xfe, 0x5a, 0x11, 0xfa};

    Console.WriteLine( "Using BitConverter,
      IsLittleEndian=" +
      BitConverter.IsLittleEndian );
    byte[] b2 = BitConverter.GetBytes( 0xba5eball );
    PrintByteArray( b2 );
    uint u2 = BitConverter.ToUInt32( b, 0 );
    Console.WriteLine( "0x{0:x}", u2 );

    Console.WriteLine( "Using shift operators" );
    b2 = new byte[4];
    b2[0] = (byte)(u);
    b2[1] = (byte)(u >> 8);
    b2[2] = (byte)(u >> 16);
    b2[3] = (byte)(u >> 24);
    PrintByteArray( b2 );
    u2 = (uint)(b[0] | b[1] << 8 | b[2] << 16 | b[3] <<
      24);
    Console.WriteLine( "0x{0:x}", u2 );

    Console.WriteLine( "Using pointers in unsafe code"
      );
    unsafe {
      b2 = new byte[4];
      fixed ( byte* pb2 = b2 )
        *((uint*)pb2) = u;
      PrintByteArray( b2 );
      fixed ( byte* pb = b )
        u2 = *((uint*)pb);
      Console.WriteLine( "0x{0:x}", u2 );
    }
  }

  static void PrintByteArray(byte[] ba)
  {
    foreach ( byte b in ba )
      Console.Write( "0x{0:x2} ", b );
    Console.WriteLine();
  }
}
```

Listing 1 You can convert an int to a byte array, and vice versa, in three ways: with the BitConverter class, with manual bit shifting, or by doing a direct memory copy.

conception. If you look in the Path column, you'll see that most of the assemblies listed are found in the .NET Framework directory, %WINDIR%\Microsoft .NET\Framework\v1.x.yyyy (see Figure 1).

The GAC is primarily a deployment feature. Even if you intend to deploy the assembly as a shared component, it might not be necessary to have it installed in the GAC during development. If you choose to have it installed in the GAC, you should also keep a copy in another directory to make it easy to reference.

You shouldn't copy your own assemblies to the Framework directory in order to include them. A better way is to put them in a separate directory, then add a key to the Windows Registry to tell VS.NET where to find them. In addition to the core Framework assemblies, VS.NET also displays any assemblies it finds in directories listed under these Registry keys:

```
HKEY_CURRENT_USER\Software\Microsoft\.NETFramework\AssemblyFolders
HKEY_LOCAL_MACHINE\Software\Microsoft\.NETFramework\AssemblyFolders
HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\7.0\AssemblyFolders
HKEY_LOCAL_MACHINE\Software\Microsoft\VisualStudio\7.0\AssemblyFolders
```

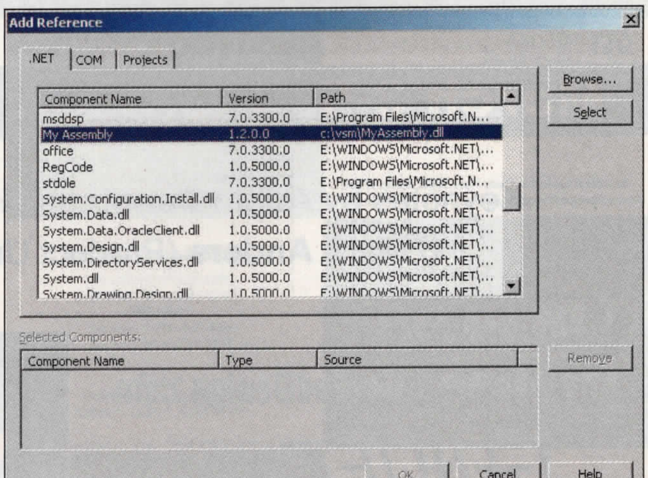
Add your own sub key to one of these, then set the *(Default)* value to the directory path where your assembly is located (see Figure 2). If you're using Visual Studio .NET 2003, change 7.0 to 7.1 in the key names. The next time you start VS.NET, you should see your assembly in the Add Reference dialog. —*M.S.*

Q: Handle Windows Messages

I have a WinForms application that needs to interoperate with a legacy Microsoft Foundation Class (MFC) application. The WinForms application replaces another MFC application. The problem is that the MFC application I still have broadcasts custom user messages that I used to handle in my MFC code. How can I handle Windows messages directly in WinForms?

A:

By default, WinForms abstracts away and encapsulates from the developer the underlying message processing. The operating system still sends messages to the application window. .NET intercepts these messages and converts them to delegate-based event invocations. However, you can provide your own message-processing logic and handle messages directly. You typically want to do that when you deal with custom messages that would be discarded otherwise. The base class of all WinForms controls and forms is the Control



Component Name	Version	Path
msddsp	7.0.3300.0	E:\Program Files\Microsoft.N...
My Assembly	1.2.0.0	c:\vsm\MyAssembly.dll
office	7.0.3300.0	E:\WINDOWS\Microsoft.NET\...
RegCode	1.0.5000.0	E:\WINDOWS\Microsoft.NET\...
stdole	7.0.3300.0	E:\Program Files\Microsoft.N...
System.Configuration.Install.dll	1.0.5000.0	E:\WINDOWS\Microsoft.NET\...
System.Data.dll	1.0.5000.0	E:\WINDOWS\Microsoft.NET\...
System.Data.OracleClient.dll	1.0.5000.0	E:\WINDOWS\Microsoft.NET\...
System.Design.dll	1.0.5000.0	E:\WINDOWS\Microsoft.NET\...
System.DirectoryServices.dll	1.0.5000.0	E:\WINDOWS\Microsoft.NET\...
System.dll	1.0.5000.0	E:\WINDOWS\Microsoft.NET\...
System.Drawing.Design.dll	1.0.5000.0	E:\WINDOWS\Microsoft.NET\...

Component Name	Type	Source

Figure 1 Simplify Component Names. You can use the System.-Reflection.AssemblyTitle attribute to give your assembly a more readable display name in the Component Name column in the Add Reference dialog box.

class. Control provides the virtual WndProc method:

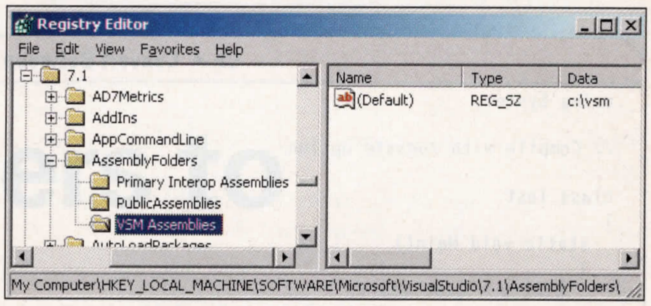
```
protected virtual void WndProc(ref
   Message m);
```

Every Control-derived class overrides WndProc. You can override it in your forms to handle messages directly. You can use the Class Wizard in C# to add the implementation for you. (You have to do it manually in VB.NET.) Open the Class Wizard, then expand the namespace containing your form and its Classes folder. Find your form, expand its Classes and Interfaces folder, find the Form class, then expand its Methods folder. The last method is WndProc. Right-click on the method, select Add from the context menu, then click on Override. VS.NET generates a stubbed-out override implementation of WndProc in your class:

```
protected override void WndProc(ref
   Message m)
{}
```

Almost without exception, you must call your base class in the override implementation, before or after your message handling. Nothing in the form will work otherwise, because WinForms won't be doing message processing. Add a call to the base class's WndProc:

```
protected override void WndProc(ref
   Message m)
```



Name	Type	Data
(Default)	REG_SZ	c:\vsm

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\7.1\AssemblyFolders\

Figure 2 Add Your Own Key. Tell VS.NET where your assembly is by adding a key to the Windows Registry. The key can have any name; only its default value matters. Not all AssemblyFolders parent keys mentioned in the text exist by default. If you can't find one, simply create it.

```
{
   base.WndProc(ref m);
}
```

Build and test your form. Everything should work the same, except now you have your hook in place. For example, you can trace every message to the output window:

```
protected override void WndProc(ref
   Message m)
{
```

application → technical → data → deployment

.NET Architecture Summit

March 3-5, Microsoft Campus

Keynotes by **Adam Denning** (Director of Application Architecture) and **Mark Anders** (Product Unit manager, .NET Framework)

1-800-848-5523

http://summit.GuidedDesign.com

GUIDED DESIGN

Chris Kinsman Juval Lowy Fernando Guerrero Keith Pleas

```
    Trace.WriteLine(m.ToString());
    base.WndProc(ref m);
}
```

The preceding code results in output similar to this:

```
msg=0x135 (WM_CTLCOLORBTN) hwnd=0x50500
    wparam=0x1010054 lparam=0x40488
    result=0x0
msg=0x2b (WM_DRAWITEM) hwnd=0x50500
    wparam=0x40488 lparam=0x12ec08 result=0x0
    msg=0x135 (WM_CTLCOLORBTN) hwnd=0x50500
    wparam=0x1010054 lparam=0x40488
    result=0x0
```

You can also handle custom messages. You can define custom messages in Windows apps in two ways, and use them for cross-application communication. Both ways are also available to your .NET apps. The first is to define a message in a range greater than WM_USER (0x400):

```
public class Util
{
    public const int WM_USER = 0x400;
}
const int UM_MYMESSAGE = Util.WM_USER +
    77;//Or some other arbitrary number
```

The problem with WM_USER is that the messages aren't guaranteed to be unique; in the case of a broadcast, you might wreck other applications that happen to use the same value for a custom message.

The second way is to use the RegisterWindowMessage Win32 API call to generate a unique message identified by a string, or return the same value if somebody registered a message already with the same string. You need to import the RegisterWindowMessage definition into .NET:

```
public class Util
{
   [DllImport("user32",EntryPoint=
      "RegisterWindowMessage")]
      public static extern int
      RegisterWindowMessage(string
      msgString);
}
```

You can even send messages from .NET to Windows (if you know their Windows handles) by importing the SendMessage API call:

```
public class Util
{
    [DllImport("user32",EntryPoint="Send
    Message")]
    public static extern int
    SendMessage(int hwnd,int msg,int
    wparam,int lparam);
}
```

You can obtain the Windows handle associated with your form by accessing the read-only Handle public property of your form class:

```
public IntPtr Handle {get;}
```

These imported APIs, properties, and hooks are powerful tools when it comes to interacting with a legacy code base, both as message sender and as message receiver.

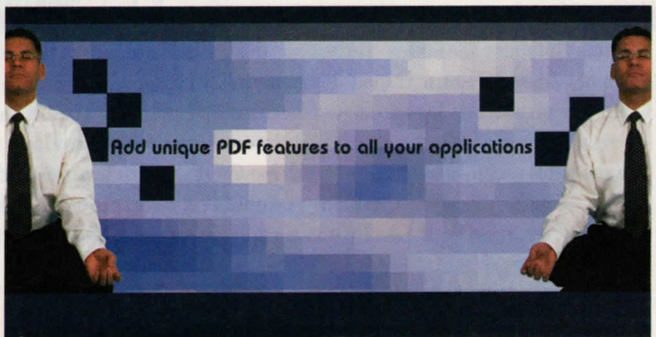
The sample code demonstrates the use of these techniques at both ends (download the code from the *VSM* Web site; see the Go Online box for details). *—J.L.*

Mattias Sjögren lives in southern Sweden, where he tries to combine consulting work with university studies. He is a Microsoft MVP for Visual Basic. Reach him at mattias@mvps.org or visit his Web site at www.msjogren.net.

Juval Löwy is a software architect and the principal of IDesign, a consulting and training company focused on .NET design and .NET migration. Juval is a Microsoft regional director for the Silicon Valley. His latest book is *Programming .NET Components* (O'Reilly & Associates). Contact him at www.idesign.net.

Additional Resources

- System.BitConverter class: http://msdn.microsoft.com/library/en-us/cpref/html/frlrfsystembitconverterclasstopic.asp
- "INFO: How to Display an Assembly in the Add Reference Dialog Box": http://support.microsoft.com/?kbid=306149



Amyuni
PDF CONVERTER
•Convert all your reports and documents to PDF format.
•Concatenate, merge, compress, embed fonts, add bookmarks, hyperlinks and watermarks...
•Create encrypted, secure documents**
•Optimize documents for the Web through PDF Linearization**

AMYUNI Technologies

NEW **Version 1.5 of Amyuni PDF Creator**

For a quick start visit our Technical notes at:
http://www.amyuni.com/en/support/technotes.html

www.amyuni.com
info: sales@amyuni.com
Evaluation:www.amyuni.com
Americas
Toll Free: 1-866-926-9864
Support: (514) 868-9227
Europe
Sales: (+33) 1 30 61 07 97
Support: (+33) 1 30 61 07 98

*Portable Document Format **Professional versions only

Amyuni
PDF CREATOR
•View, Edit and Print PDF* documents
•Create complex documents, forms and reports
•Create encrypted, secure documents**
•Optimize documents for the Web through PDF Linerization**
•ActiveX component for easy integration into your applications
•Link directly to the database...

Amyuni
Client/Server PDF
printing solutions
•Web or application servers delivering reports in PDF format
•Server used to archive or print documents
•Integration with .NET platform

Other formats:
DHTML, RTF, TXT
Available versions
•Single-user license
•Server license
•Royalty-free Developer license
•Unlimited Site licence
•OEM license

All trademarks are property of their respective owners
2002 AMYUNI Technologies All rights reserved.