Trying out all the different ways of recognizing different types of timestamps from quite a long way away

devblogs.microsoft.com/oldnewthing/20150706-00

July 6, 2015

Raymond Chen

Today's Little Program takes a 64-bit integer and tries to interpret it in all the various timestamp formats. This comes in handy when you have extracted a timestamp from a crash dump and want to see it in a friendly format.

```
using System;

class Program
{
 static void TryFormat(string format, Func<DateTime> func)
 {
  try
  {
   DateTime d = func();
   Console.WriteLine("{0} {1}", format, d);
  }
  catch (ArgumentException)
  {
   Console.WriteLine("{0} - invalid", format);
  }
 }
```

The `TryFormat` method executes the passed-in function inside a try/catch block. If the function executes successfully, then we print the result. If it raises an argument exception, then we declare the value as invalid.

```
static DateTime DateTimeFromDosDateTime(long value)
{
 if ((ulong)value > 0x00000000FFFFFFFF) {
  throw new ArgumentOutOfRangeException();
 }
 int intValue = (int)value;
 int year = (intValue >> 25) & 127;
 int month = (intValue >> 21) & 15;
 int day = (intValue >> 16) & 31;
 int hour = (intValue >> 11) & 31;
 int minute = (intValue >> 5) & 63;
 int second = (intValue << 1) & 63;
 return new DateTime(1980 + year, month, day, hour, minute, second);
}
```

The `DateTimeFromDosDateTime` function treats the 64-bit value as a 32-bit date/time stamp in MS-DOS format. Assuming the value fits in a 32-bit integer, we extract the bitfields corresponding to the year, month, day, hour, minute, and second, and construct a `DateTime` from it.

```
public static void Main(string[] args)
{
 if (args.Length < 1) return;

 long value = ParseLongSomehow(args[0]);

 Console.WriteLine("Timestamp {0} (0x{0:X}) could mean", value);

 TryFormat("Unix time",
   () => DateTime.FromFileTimeUtc(10000000 * value + 116444736000000000));
 TryFormat("UTC FILETIME",
   () => DateTime.FromFileTimeUtc(value));
 TryFormat("Local FILETIME",
   () => DateTime.FromFileTime(value));
 TryFormat("UTC DateTime",
   () => new DateTime(value, DateTimeKind.Utc));
 TryFormat("Local DateTime",
   () => new DateTime(value, DateTimeKind.Local));
 TryFormat("Binary DateTime",
   () => DateTime.FromBinary(value));
 TryFormat("MS-DOS Date/Time",
   () => DateTimeFromDosDateTime(value));
 TryFormat("OLE Automation Date/Time",
   () => DateTime.FromOADate(BitConverter.Int64BitsToDouble(value)));
 }
}
```

Once we have parsed out the command line, we pump the value through all the different conversion functions. Most of them are natively supported by the `DateTime` structure, but we had to create a few of them manually.

Raymond Chen

Follow